Statistical Parsing: an Overview

based on a draft chapter 14 of the 2nd edition of “Speech and Language Processing” by Jurafsky and Martin
Jennifer Foster

NCLT Seminar Series 2007-2008

23rd January 2007
Outline

Statistical Parsing: What and Why?

PCFGs
   Probabilistic CKY
   Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
   Collins Parser
   Charniak Parser
   Unlexicalised Parsing

Discriminative Parsing
   Discriminative Reranking
   Discriminative Dynamic Programming

Dependency Parsing

Conclusion
Outline

Statistical Parsing: What and Why?

PCFGs
   Probabilistic CKY
   Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
   Collins Parser
   Charniak Parser
   Unlexicalised Parsing

Discriminative Parsing
   Discriminative Reranking
   Discriminative Dynamic Programming

Dependency Parsing

Conclusion
Statistical Parsing: What and Why?

What is natural language parsing?
Process of analysing the structure of natural language sentences.

What is statistical natural language parsing?
Parsing in which the most likely analysis is selected amongst all possible analyses for an input sentence.

Why are most natural language parsers statistical?
Because syntactic ambiguity makes non-probabilistic or symbolic parsing intractable!
Statistical Parsing: What and Why?

What is natural language parsing?
Process of analysing the structure of natural language sentences.

What is statistical natural language parsing?
Parsing in which the most likely analysis is selected amongst all possible analyses for an input sentence.

Why are most natural language parsers statistical?
Because syntactic ambiguity makes non-probabilistic or symbolic parsing intractable!
Statistical Parsing: What and Why?

What is natural language parsing?
Process of analysing the structure of natural language sentences.

What is statistical natural language parsing?
Parsing in which the most likely analysis is selected amongst all possible analyses for an input sentence.

Why are most natural language parsers statistical?
Because syntactic ambiguity makes non-probabilistic or symbolic parsing intractable!
Statistical Parsing: What and Why?

What is natural language parsing?
Process of analysing the structure of natural language sentences.

What is statistical natural language parsing?
Parsing in which the most likely analysis is selected amongst all possible analyses for an input sentence.

Why are most natural language parsers statistical?
Because syntactic ambiguity makes non-probabilistic or symbolic parsing intractable!
Statistical Parsing: What and Why?

What is natural language parsing?
Process of analysing the structure of natural language sentences.

What is statistical natural language parsing?
Parsing in which the most likely analysis is selected amongst all possible analyses for an input sentence.

Why are most natural language parsers statistical?
Because syntactic ambiguity makes non-probabilistic or symbolic parsing intractable!
What is natural language parsing?
Process of analysing the structure of natural language sentences.

What is statistical natural language parsing?
Parsing in which the most likely analysis is selected amongst all possible analyses for an input sentence.

Why are most natural language parsers statistical?
Because syntactic ambiguity makes non-probabilistic or symbolic parsing intractable!
Benefits of Statistical Parsing

Jurafsky and Martin identify the following two benefits of probabilistic parsing:

1. Syntactic Disambiguation (main motivation)
2. Language Modelling
Outline

Statistical Parsing: What and Why?

PCFGs
  Probabilistic CKY
  Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
  Collins Parser
  Charniak Parser
  Unlexicalised Parsing

Discriminative Parsing
  Discriminative Reranking
  Discriminative Dynamic Programming

Dependency Parsing

Conclusion
A PCFG $\langle N, \Sigma, R, S \rangle$ is defined as follows:

1. $N$ is the set of non-terminal symbols
2. $\Sigma$ is the terminals (disjoint from $N$)
3. $R$ is a set of rules of the form $A \rightarrow \beta[p]$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$, and $p$ is a number between 0 and 1
4. A start symbol, $S \in N$

A PCFG is a CFG in which each rule is associated with a probability.
A PCFG $\langle N, \Sigma, R, S \rangle$ is defined as follows:

1. $N$ is the set of non-terminal symbols
2. $\Sigma$ is the terminals (disjoint from $N$)
3. $R$ is a set of rules of the form $A \rightarrow \beta[p]$, where $A \in N$ and $\beta \in (\Sigma \cup N)^*$, and $p$ is a number between 0 and 1
4. A start symbol, $S \in N$

A PCFG is a CFG in which each rule is associated with a probability.
More about PCFGs

What does the $p$ associated with each rule express?

It expresses the probability that the LHS non-terminal will be expanded as the RHS sequence.

- $P(A \rightarrow \beta | A)$
- $\sum_\beta P(A \rightarrow \beta | A) = 1$
- The sum of the probabilities associated with all of the rules expanding the non-terminal $A$ is 1.
More about PCFGs

What does the p associated with each rule express?
It expresses the probability that the LHS non-terminal will be expanded as the RHS sequence.

- \( P(A \rightarrow \beta | A) \)
- \( \sum_{\beta} P(A \rightarrow \beta | A) = 1 \)
- The sum of the probabilities associated with all of the rules expanding the non-terminal \( A \) is 1.
More about PCFGs

What does the $p$ associated with each rule express?

It expresses the probability that the LHS non-terminal will be expanded as the RHS sequence.

- $P(A \rightarrow \beta | A)$
- $\sum_{\beta} P(A \rightarrow \beta | A) = 1$

- The sum of the probabilities associated with all of the rules expanding the non-terminal $A$ is 1.
More about PCFGs

What does the $p$ associated with each rule express?

It expresses the probability that the LHS non-terminal will be expanded as the RHS sequence.

- $P(A \rightarrow \beta | A)$
- $\sum_{\beta} P(A \rightarrow \beta | A) = 1$
- The sum of the probabilities associated with all of the rules expanding the non-terminal $A$ is 1.
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.
  \[ P(T, S) = \prod_{i=1}^{n} P(A_i \to \beta_i) \] where \( n \) is the number of rules in \( T \).
- \( P(T, S) = P(T)P(S|T) = P(S)P(T|S) \) by definition
- But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)
- So, \( P(T, S) = P(T) \)
- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.
  \[ \hat{T}(S) = \arg\max P(T|S) \text{ s.t. } S = \text{yield}(T) \]
- \( P(T|S) = \frac{P(T,S)}{P(S)} \)
- \( P(S) \) will be constant over all the trees for \( S \).
  \[ \hat{T}(S) = \arg\max P(T, S) \text{ s.t. } S = \text{yield}(T) \]
  \[ \hat{T}(S) = \arg\max P(T) \text{ s.t. } S = \text{yield}(T) \]
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \]

where \( n \) is the number of rules in \( T \).

- \( P(T, S) = P(T)P(S|T) = P(S)P(T|S) \) by definition
- But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)
- So, \( P(T, S) = P(T) \)

A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \arg\max P(T|S) \text{ s.t. } S = \text{yield}(T) \]

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

- \( P(S) \) will be constant over all the trees for \( S \).
- \( \hat{T}(S) = \arg\max P(T, S) \text{ s.t. } S = \text{yield}(T) \)
- \( \hat{T}(S) = \arg\max P(T) \text{ s.t. } S = \text{yield}(T) \)
PCFGs and Disambiguation

A PCFG assigns a probability to every parse tree or derivation associated with a sentence.

This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \] where \( n \) is the number of rules in \( T \).

\[ P(T, S) = P(T)P(S|T) = P(S)P(T|S) \] by definition

But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)

So, \( P(T, S) = P(T) \)

A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \arg\max P(T|S) \text{ s.t. } S = \text{yield}(T) \]

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

\( P(S) \) will be constant over all the trees for \( S \).

\[ \hat{T}(S) = \arg\max P(T, S) \text{ s.t. } S = \text{yield}(T) \]

\[ \hat{T}(S) = \arg\max P(T) \text{ s.t. } S = \text{yield}(T) \]
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \] where \( n \) is the number of rules in \( T \).

\[ P(T, S) = P(T)P(S|T) = P(S)P(T|S) \] by definition

- But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)
- So, \( P(T, S) = P(T) \)

- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \arg\max P(T|S) \text{ s.t. } S = \text{yield}(T) \]

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

\( P(S) \) will be constant over all the trees for \( S \).

\[ \hat{T}(S) = \arg\max P(T, S) \text{ s.t. } S = \text{yield}(T) \]

\[ \hat{T}(S) = \arg\max P(T) \text{ s.t. } S = \text{yield}(T) \]
A PCFG assigns a probability to every parse tree or derivation associated with a sentence.

This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \] where \( n \) is the number of rules in \( T \).

\[ P(T, S) = P(T)P(S|T) = P(S)P(T|S) \] by definition

But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)

So, \( P(T, S) = P(T) \)

A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\( \hat{T}(S) = \arg\max P(T|S) \text{ s.t. } S = \text{yield}(T) \)

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

\( P(S) \) will be constant over all the trees for \( S \).

\( \hat{T}(S) = \arg\max P(T,S) \text{ s.t. } S = \text{yield}(T) \)

\( \hat{T}(S) = \arg\max P(T) \text{ s.t. } S = \text{yield}(T) \)
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[
P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i)
\]
where \(n\) is the number of rules in \(T\).

- \(P(T, S) = P(T)P(S|T) = P(S)P(T|S)\) by definition
- But \(P(S|T) = 1\) because all the words in \(S\) are in \(T\)
- So, \(P(T, S) = P(T)\)

- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[
\hat{T}(S) = \text{argmax } P(T|S) \text{ s.t. } S = \text{yield}(T)
\]

\[
P(T|S) = \frac{P(T,S)}{P(S)}
\]

- \(P(S)\) will be constant over all the trees for \(S\).

\[
\hat{T}(S) = \text{argmax } P(T, S) \text{ s.t. } S = \text{yield}(T)
\]

\[
\hat{T}(S) = \text{argmax } P(T) \text{ s.t. } S = \text{yield}(T)
\]
PCFGs and Disambiguation

A PCFG assigns a probability to every parse tree or derivation associated with a sentence.

This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \]

where \( n \) is the number of rules in \( T \).

\[ P(T, S) = P(T)P(S|T) = P(S)P(T|S) \]

by definition

But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)

So, \( P(T, S) = P(T) \)

A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \arg\max P(T|S) \text{ s.t. } S = \text{yield}(T) \]

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

\( P(S) \) will be constant over all the trees for \( S \).

\[ \hat{T}(S) = \arg\max P(T, S) \text{ s.t. } S = \text{yield}(T) \]

\[ \hat{T}(S) = \arg\max P(T) \text{ s.t. } S = \text{yield}(T) \]
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \text{ where } n \text{ is the number of rules in } T. \]

- \[ P(T, S) = P(T)P(S|T) = P(S)P(T|S) \text{ by definition} \]
- But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)
- So, \( P(T, S) = P(T) \)
- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \text{argmax } P(T|S) \text{ s.t. } S = \text{yield}(T) \]

- \( P(T|S) = \frac{P(T,S)}{P(S)} \)
- \( P(S) \) will be constant over all the trees for \( S \).

\[ \hat{T}(S) = \text{argmax } P(T, S) \text{ s.t. } S = \text{yield}(T) \]

\[ \hat{T}(S) = \text{argmax } P(T) \text{ s.t. } S = \text{yield}(T) \]
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \]

where \( n \) is the number of rules in \( T \).

- \( P(T, S) = P(T)P(S|T) = P(S)P(T|S) \) by definition
- But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)
- So, \( P(T, S) = P(T) \)
- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \arg\max P(T|S) \text{ s.t. } S = \text{yield}(T) \]

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

- \( P(S) \) will be constant over all the trees for \( S \).
- \( \hat{T}(S) = \arg\max P(T, S) \text{ s.t. } S = \text{yield}(T) \)
- \( \hat{T}(S) = \arg\max P(T) \text{ s.t. } S = \text{yield}(T) \)
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \text{ where } n \text{ is the number of rules in } T. \]

- \[ P(T, S) = P(T)P(S|T) = P(S)P(T|S) \text{ by definition} \]
- But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)
- So, \( P(T, S) = P(T) \)
- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \operatorname{argmax} P(T|S) \text{ s.t. } S = \text{yield}(T) \]

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

- \( P(S) \) will be constant over all the trees for \( S \).
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[ P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i) \] where \( n \) is the number of rules in \( T \).

- \( P(T, S) = P(T)P(S|T) = P(S)P(T|S) \) by definition
- But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)
- So, \( P(T, S) = P(T) \)
- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[ \hat{T}(S) = \text{argmax} \ P(T|S) \text{ s.t. } S = \text{yield}(T) \]

\[ P(T|S) = \frac{P(T,S)}{P(S)} \]

- \( P(S) \) will be constant over all the trees for \( S \).
- \( \hat{T}(S) = \text{argmax} \ P(T, S) \text{ s.t. } S = \text{yield}(T) \)
PCFGs and Disambiguation

- A PCFG assigns a probability to every parse tree or derivation associated with a sentence.
- This probability is the product of the rules applied in building the parse tree.

\[
P(T, S) = \prod_{i=1}^{n} P(A_i \rightarrow \beta_i)
\]
where \( n \) is the number of rules in \( T \).

\[
P(T, S) = P(T)P(S|T) = P(S)P(T|S)
\]
by definition

But \( P(S|T) = 1 \) because all the words in \( S \) are in \( T \)

So, \( P(T, S) = P(T) \)

- A parse disambiguation algorithm picks out the most probable parse tree for a sentence.

\[
\hat{T}(S) = \text{argmax } P(T|S) \text{ s.t. } S = \text{yield}(T)
\]

\[
P(T|S) = \frac{P(T,S)}{P(S)}
\]

- \( P(S) \) will be constant over all the trees for \( S \).

\[
\hat{T}(S) = \text{argmax } P(T, S) \text{ s.t. } S = \text{yield}(T)
\]

\[
\hat{T}(S) = \text{argmax } P(T) \text{ s.t. } S = \text{yield}(T)
\]
PCFGs and Language Modelling

As well as assigning probabilities to parse trees, a PCFG assigns a probability to every sentence generated by the grammar.

This is useful for language modelling.

- The probability of a sentence is the sum of the probabilities of each parse tree associated with the sentence.

\[
P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T, S)
\]

\[
P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T)
\]

When is it useful to know the probability of a sentence?

When ranking the output of speech recognition, machine translation and error correction systems.
PCFGs and Language Modelling

As well as assigning probabilities to parse trees, a PCFG assigns a probability to every sentence generated by the grammar. This is useful for language modelling.

- The probability of a sentence is the sum of the probabilities of each parse tree associated with the sentence.

\[
P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T, S)
\]

\[
P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T)
\]

When is it useful to know the probability of a sentence?

When ranking the output of speech recognition, machine translation and error correction systems.
PCFGs and Language Modelling

As well as assigning probabilities to parse trees, a PCFG assigns a probability to every sentence generated by the grammar. This is useful for language modelling.

- The probability of a sentence is the sum of the probabilities of each parse tree associated with the sentence.

  \[ P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T, S) \]

- The probability of a sentence is the sum of the probabilities of each parse tree associated with the sentence.

  \[ P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T) \]

When is it useful to know the probability of a sentence?

When ranking the output of speech recognition, machine translation and error correction systems.
PCFGs and Language Modelling

As well as assigning probabilities to parse trees, a PCFG assigns a probability to every sentence generated by the grammar. This is useful for language modelling.

- The probability of a sentence is the sum of the probabilities of each parse tree associated with the sentence.

\[ P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T) \]

\[ P(S) = \sum_{T \text{ s.t. } \text{yield}(T)=S} P(T, S) \]

When is it useful to know the probability of a sentence?
When ranking the output of speech recognition, machine translation and error correction systems.

The probability of a sentence is the sum of the probabilities of each parse tree associated with the sentence.
PCFGs and Language Modelling

As well as assigning probabilities to parse trees, a PCFG assigns a probability to every sentence generated by the grammar. This is useful for language modelling.

- The probability of a sentence is the sum of the probabilities of each parse tree associated with the sentence.

\[
P(S) = \sum_{T \text{ s.t. } \text{yield}(T) = S} P(T, S)
\]

\[
P(S) = \sum_{T \text{ s.t. } \text{yield}(T) = S} P(T)
\]

When is it useful to know the probability of a sentence?

When ranking the output of speech recognition, machine translation and error correction systems.
Probabilistic CKY

Probabilistic versions of most parsing algorithms exist.

Many probabilistic parsers use a probabilistic version of the CKY bottom-up chart parsing algorithm (see Chap. 13)

Sentence, $s$, of length $n$ and CFG grammar with $V$ non-terminal symbols

- Normal CKY
  2-d $(n + 1) \times (n + 1)$ array where a value in a cell $(i, j)$ is a list of non-terminals spanning position $i$ through $j$ in $s$

- Probabilistic CKY
  3-d $(n + 1) \times (n + 1) \times V$ array where a value in a cell $(i, j, K)$ is the probability of the non-terminal $K$ spanning position $i$ through $j$ in $s$

As with regular CKY, probabilistic CKY assumes that the grammar is in Chomsky-normal form.

Note that probabilities will also need to be adjusted when transforming a non-CNF PCFG into a CNF PCFG.
Probabilistic CKY

Probabilistic versions of most parsing algorithms exist.
Many probabilistic parsers use a probabilistic version of the CKY bottom-up chart parsing algorithm (see Chap. 13)
Sentence, $s$, of length $n$ and CFG grammar with $V$ non-terminal symbols

- **Normal CKY**
  2-d $(n + 1) \times (n + 1)$ array where a value in a cell $(i, j)$ is a list of non-terminals spanning position $i$ through $j$ in $s$

- **Probabilistic CKY**
  3-d $(n + 1) \times (n + 1) \times V$ array where a value in a cell $(i, j, K)$ is the probability of the non-terminal $K$ spanning position $i$ through $j$ in $s$

As with regular CKY, probabilistic CKY assumes that the grammar is in Chomsky-normal form.
Note that probabilities will also need to be adjusted when transforming a non-CNF PCFG into a CNF PCFG.
Probabilistic CKY

Probabilistic versions of most parsing algorithms exist.

Many probabilistic parsers use a probabilistic version of the CKY bottom-up chart parsing algorithm (see Chap. 13) Sentence, $s$, of length $n$ and CFG grammar with $V$ non-terminal symbols

- **Normal CKY**
  2-d $(n + 1) \times (n + 1)$ array where a value in a cell $(i, j)$ is a list of non-terminals spanning position $i$ through $j$ in $s$

- **Probabilistic CKY**
  3-d $(n + 1) \times (n + 1) \times V$ array where a value in a cell $(i, j, K)$ is the probability of the non-terminal $K$ spanning position $i$ through $j$ in $s$

As with regular CKY, probabilistic CKY assumes that the grammar is in Chomsky-normal form.

Note that probabilities will also need to be adjusted when transforming a non-CNF PCFG into a CNF PCFG.
Probabilistic CKY

Probabilistic versions of most parsing algorithms exist.

Many probabilistic parsers use a probabilistic version of the CKY bottom-up chart parsing algorithm (see Chap. 13)

Sentence, $s$, of length $n$ and CFG grammar with $V$ non-terminal symbols

- **Normal CKY**
  2-d $(n + 1) \times (n + 1)$ array where a value in a cell $(i, j)$ is a list of non-terminals spanning position $i$ through $j$ in $s$

- **Probabilistic CKY**
  3-d $(n + 1) \times (n + 1) \times V$ array where a value in a cell $(i, j, K)$ is the probability of the non-terminal $K$ spanning position $i$ through $j$ in $s$

As with regular CKY, probabilistic CKY assumes that the grammar is in Chomsky-normal form.

Note that probabilities will also need to be adjusted when transforming a non-CNF PCFG into a CNF PCFG.
Obtaining Probabilistic Grammars

In a PCFG every rule is associated with a probability. But where do these rule probabilities come from?

There are two ways to obtain rule probabilities for a PCFG.

1. Use a treebank
   \[ P(A \rightarrow \beta|A) = \frac{count(A \rightarrow \beta)}{count(A)} \]

2. Corpus of sentences + Inside-Outside algorithm (Baker, 1979)
   2.1 Take a CFG and set all rules to have equal probability
   2.2 Parse the corpus with the CFG
   2.3 Adjust the probabilities
   2.4 Repeat steps two and three until probabilities converge

The Inside-Outside algorithm is a type of Expectation Maximisation algorithm. It can also be used to induce a grammar, but only with limited success.
Obtaining Probabilistic Grammars

In a PCFG every rule is associated with a probability. But where do these rule probabilities come from?

There are two ways to obtain rule probabilities for a PCFG.

1. Use a treebank
   \[ P(A \rightarrow \beta | A) = \frac{\text{count}(A \rightarrow \beta)}{\text{count}(A)} \]

2. Corpus of sentences + Inside-Outsides algorithm (Baker, 1979)
   2.1 Take a CFG and set all rules to have equal probability
   2.2 Parse the corpus with the CFG
   2.3 Adjust the probabilities
   2.4 Repeat steps two and three until probabilities converge

The Inside-Outsides algorithm is a type of Expectation Maximisation algorithm. It can also be used to induce a grammar, but only with limited success.
Obtaining Probabilistic Grammars

In a PCFG every rule is associated with a probability. But where do these rule probabilities come from?

There are two ways to obtain rule probabilities for a PCFG.

1. Use a treebank
   \[ P(A \rightarrow \beta | A) = \frac{\text{count}(A \rightarrow \beta)}{\text{count}(A)} \]

2. Corpus of sentences + Inside-Outsides algorithm (Baker, 1979)
   2.1 Take a CFG and set all rules to have equal probability
   2.2 Parse the corpus with the CFG
   2.3 Adjust the probabilities
   2.4 Repeat steps two and three until probabilities converge

The Inside-Outsides algorithm is a type of Expectation Maximisation algorithm. It can also be used to induce a grammar, but only with limited success.
Obtaining Probabilistic Grammars

In a PCFG every rule is associated with a probability. But where do these rule probabilities come from?

There are two ways to obtain rule probabilities for a PCFG.

1. Use a treebank

   \[ P(A \rightarrow \beta | A) = \frac{\text{count}(A \rightarrow \beta)}{\text{count}(A)} \]

2. Corpus of sentences + Inside-Outside algorithm (Baker, 1979)

   2.1 Take a CFG and set all rules to have equal probability
   2.2 Parse the corpus with the CFG
   2.3 Adjust the probabilities
   2.4 Repeat steps two and three until probabilities converge

The Inside-Outside algorithm is a type of Expectation Maximisation algorithm. It can also be used to induce a grammar, but only with limited success.
Obtaining Probabilistic Grammars

In a PCFG every rule is associated with a probability. But where do these rule probabilities come from?

There are two ways to obtain rule probabilities for a PCFG.

1. Use a treebank
   \[ P(A \rightarrow \beta | A) = \frac{\text{count}(A \rightarrow \beta)}{\text{count}(A)} \]

2. Corpus of sentences + Inside-Outside algorithm (Baker, 1979)
   2.1 Take a CFG and set all rules to have equal probability
   2.2 Parse the corpus with the CFG
   2.3 Adjust the probabilities
   2.4 Repeat steps two and three until probabilities converge

The Inside-Outside algorithm is a type of Expectation Maximisation algorithm. It can also be used to induce a grammar, but only with limited success.
Outline

Statistical Parsing: What and Why?

PCFGs
   Probabilistic CKY
   Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
   Collins Parser
   Charniak Parser
   Unlexicalised Parsing

Discriminative Parsing
   Discriminative Reranking
   Discriminative Dynamic Programming

Dependency Parsing

Conclusion
Limitations of PCFG Parsing

Two well-known drawbacks of PCFG parsing are:

1. the *independence* of the rules in a PCFG
2. their failure to fully exploit *lexical* knowledge in resolving ambiguities
PCFG Rule Independence

In PCFG parsing, the application of a rule in a derivation is an independent event.

This means that previous rule applications in the same derivation have no influence over the current rule application.

- The rule independence can be a disadvantage.
- Consider, for example, that, an English subject NP is extremely likely to be realised as a pronoun, and that an English object NP is more likely to be realised as a non-pronominal noun phrase.
- This cannot be reflected in a PCFG which does not distinguish object NPs from subject NPs.
PCFG Rule Independence

In PCFG parsing, the application of a rule in a derivation is an independent event.

This means that previous rule applications in the same derivation have no influence over the current rule application.

- The rule independence can be a disadvantage.
- Consider, for example, that, an English subject NP is extremely likely to be realised as a pronoun, and that an English object NP is more likely to be realised as a non-pronominal noun phrase.
- This cannot be reflected in a PCFG which does not distinguish object NPs from subject NPs.
PCFG Rule Independence

In PCFG parsing, the application of a rule in a derivation is an independent event. This means that previous rule applications in the same derivation have no influence over the current rule application.

▶ The rule independence can be a disadvantage.
▶ Consider, for example, that, an English subject NP is extremely likely to be realised as a pronoun, and that an English object NP is more likely to be realised as a non-pronominal noun phrase.
▶ This cannot be reflected in a PCFG which does not distinguish object NPs from subject NPs.
PCFG Rule Independence

In PCFG parsing, the application of a rule in a derivation is an independent event.
This means that previous rule applications in the same derivation have no influence over the current rule application.

- The rule independence can be a disadvantage.
- Consider, for example, that, an English subject NP is extremely likely to be realised as a pronoun, and that an English object NP is more likely to be realised as a non-pronominal noun phrase.
- This cannot be reflected in a PCFG which does not distinguish object NPs from subject NPs.
PCFG Rule Independence

In PCFG parsing, the application of a rule in a derivation is an independent event. This means that previous rule applications in the same derivation have no influence over the current rule application.

- The rule independence can be a disadvantage.
- Consider, for example, that, an English subject NP is extremely likely to be realised as a pronoun, and that an English object NP is more likely to be realised as a non-pronominal noun phrase.
- This cannot be reflected in a PCFG which does not distinguish object NPs from subject NPs.
Using Lexical Knowledge to Resolve Ambiguity

- Consider, for example, the sentences
  Workers dumped sacks into a bin
  Fisherman caught tons of herring

- The problem of where to attach a PP is a common form of syntactic ambiguity.

- A main motivation for probabilistic parsing is to perform accurate disambiguation.

- People resolve this kind of ambiguity by looking at the actual nouns, verbs and prepositions involved.

- Faced with a choice between two attachment sites, VP and NP, for a constituent PP, a PCFG can only compare the rule probabilities of
  1. $VP \rightarrow VP PP$
  2. $NP \rightarrow NP PP$

  These rules contain no lexical information.
Using Lexical Knowledge to Resolve Ambiguity

- Consider, for example, the sentences
  *Workers dumped sacks into a bin*
  *Fisherman caught tons of herring*

- The problem of where to attach a PP is a common form of syntactic ambiguity.

- A main motivation for probabilistic parsing is to perform accurate disambiguation.

- People resolve this kind of ambiguity by looking at the actual nouns, verbs and prepositions involved.

- Faced with a choice between two attachment sites, *VP* and *NP*, for a constituent *PP*, a PCFG can only compare the rule probabilities of
  1. *VP → VP PP*
  2. *NP → NP PP*

  These rules contain no lexical information.
Using Lexical Knowledge to Resolve Ambiguity

- Consider, for example, the sentences
  
  *Workers dumped sacks into a bin*
  *Fisherman caught tons of herring*

- The problem of where to attach a PP is a common form of syntactic ambiguity.

- A main motivation for probabilistic parsing is to perform accurate disambiguation.

- People resolve this kind of ambiguity by looking at the actual nouns, verbs and prepositions involved.

- Faced with a choice between two attachment sites, *VP* and *NP*, for a constituent *PP*, a PCFG can only compare the rule probabilities of
  
  1. *VP → VP PP* versus
  2. *NP → NP PP*

  These rules contain no lexical information.
Using Lexical Knowledge to Resolve Ambiguity

- Consider, for example, the sentences
  \textit{Workers dumped sacks into a bin}
  \textit{Fisherman caught tons of herring}

- The problem of where to attach a PP is a common form of syntactic ambiguity.

- A main motivation for probabilistic parsing is to perform accurate disambiguation.

- People resolve this kind of ambiguity by looking at the actual nouns, verbs and prepositions involved.

- Faced with a choice between two attachment sites, \textit{VP} and \textit{NP}, for a constituent \textit{PP}, a PCFG can only compare the rule probabilities of
  
  1. \textit{VP} \rightarrow \textit{VP PP} versus  
  2. \textit{NP} \rightarrow \textit{NP PP}

  These rules contain no lexical information.
Using Lexical Knowledge to Resolve Ambiguity

- Consider, for example, the sentences
  
  *Workers dumped sacks into a bin*
  
  *Fisherman caught tons of herring*

- The problem of where to attach a PP is a common form of syntactic ambiguity.

- A main motivation for probabilistic parsing is to perform accurate disambiguation.

- People resolve this kind of ambiguity by looking at the actual nouns, verbs and prepositions involved.

- Faced with a choice between two attachment sites, $VP$ and $NP$, for a constituent $PP$, a PCFG can only compare the rule probabilities of
  
  1. $VP \rightarrow VP PP$ versus
  2. $NP \rightarrow NP PP$

  These rules contain no lexical information.
Outline

Statistical Parsing: What and Why?

PCFGs
  Probabilistic CKY
  Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
  Collins Parser
  Charniak Parser
  Unlexicalised Parsing

Discriminative Parsing
  Discriminative Reranking
  Discriminative Dynamic Programming

Dependency Parsing

Conclusion
Generative, History-Based, Lexicalised Parsers

Generative, history-based lexicalized parsers overcome the limits of PCFGs by employing:

1. lexicalisation
2. more complicated probabilistic models

Two examples are:
1. the Collins parser
2. the Charniak parser
Generative, History-Based, Lexicalised Parsers

Generative, history-based lexicalized parsers overcome the limits of PCFGs by employing:

1. lexicalisation
2. more complicated probabilistic models

Two examples are:

1. the Collins parser
2. the Charniak parser
Lexicalised PCFGs

A PCFG can be lexicalised by associating a word and part-of-speech tag with every non-terminal in the grammar.

It is *head*-lexicalised if the word is the head of the constituent described by the non-terminal.
Lexicalised PCFGs

A non-lexicalised parse tree for the sentence

_Last week IBM bought Lotus_

```
S
  /\  \\
 NP /   \ NP
 |   /   |
 JJ NN NNP VBD NP
|   |   |   |   |
 Last week IBM bought NNP
 |   |   |
    |   |
    Last week IBM bought Lotus
```
Lexicalised PCFGs

A lexicalised parse tree for the sentence
Last week IBM bought Lotus

```
S(bought,VBD)
   / \
NP(week,NN)  NP(IBM,NNP)  VP(bought,VBD)
   |    |    |    |
JJ(Last,JJ) NN(week,NN) NNP(IBM,NNP) VBD(bought,VBD)
     |      |          |        |
Last  week  IBM        bought
          |      |
          |      |
          |      |
NP(Lotus,NNP)  NNP(Lotus,NNP)
              |        |
              |        |
              |        |
              |        |
              |        |
              Lotus
```
Lexicalised PCFGs

Lexicalisation drastically increases the size of the grammar.
This leads to the sparse data problem.

- How do we estimate the probabilities of the rule
  \[ S(bought, VBD) \rightarrow NP(week, NN) \quad NP(IBM, NNP) \quad VP(bought, VBD) \]

- Maximum-likelihood estimation won’t work:
  \[
  \frac{\text{count}(S(bought, VBD) \rightarrow NP(week, NN) \quad NP(IBM, NNP) \quad VP(bought, VBD))}{\text{count}(S(bought, VBD))}
  \]

- Sparse data problem can be tackled by breaking the probability estimation for the right-hand side of the rule into three parts:
  1. predict the probability of the head constituent, in this case, \( VP(bought, VBD) \)
  2. predict the probability of the constituents on the left of the head, in this case, \( NP(IBM, NNP) \) and \( NP(week, NN) \)
  3. predict the probability of the constituents on the right of the head, in this case, none
Lexicalised PCFGs

Lexicalisation drastically increases the size of the grammar.
This leads to the sparse data problem.

- How do we estimate the probabilities of the rule
  \[ S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD) \]

- Maximum-likelihood estimation won’t work:
  \[ \frac{\text{count}(S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD))}{\text{count}(S(bought, VBD))} \]

- Sparse data problem can be tackled by breaking the probability estimation for the right-hand side of the rule into three parts:
  
  1. predict the probability of the head constituent, in this case, \( VP(bought, VBD) \)
  
  2. predict the probability of the constituents on the left of the head, in this case, \( NP(IBM, NNP) \) and \( NP(week, NN) \)
  
  3. predict the probability of the constituents on the right of the head, in this case, none
Lexicalised PCFGs

Lexicalisation drastically increases the size of the grammar.
This leads to the sparse data problem.

▶ How do we estimate the probabilities of the rule

\[
S(bought,VBD) \rightarrow \\
NP(week,NN) \ NP(IBM,NNP) \ VP(bought,VBD)
\]

▶ Maximum-likelihood estimation won’t work:

\[
\frac{\text{count}(S(bought,VBD) \rightarrow NP(week,NN) \ NP(IBM,NNP) \ VP(bought,VBD))}{\text{count}(S(bought,VBD))}
\]

▶ Sparse data problem can be tackled by breaking the probability estimation for the right-hand side of the rule into three parts:

1. predict the probability of the head constituent, in this case, \(VP(bought,VBD)\)
2. predict the probability of the constituents on the left of the head, in this case, \(NP(IBM,NNP)\) and \(NP(week,NN)\)
3. predict the probability of the constituents on the right of the head, in this case, none
Lexicalised PCFGs

Lexicalisation drastically increases the size of the grammar. This leads to the sparse data problem.

- How do we estimate the probabilities of the rule $S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD)$

- Maximum-likelihood estimation won’t work:
  $\frac{\text{count}(S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD))}{\text{count}(S(bought, VBD))}$

- Sparse data problem can be tackled by breaking the probability estimation for the right-hand side of the rule into three parts:
  1. predict the probability of the head constituent, in this case, $VP(bought, VBD)$
  2. predict the probability of the constituents on the left of the head, in this case, $NP(IBM, NNP)$ and $NP(week, NN)$
  3. predict the probability of the constituents on the right of the head, in this case, none
Lexicalised PCFGs

Lexicalisation drastically increases the size of the grammar.
This leads to the sparse data problem.

- How do we estimate the probabilities of the rule
  \[ S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD) \]
- Maximum-likelihood estimation won’t work:
  \[
  \frac{count(S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD))}{count(S(bought, VBD))}
  \]
- Sparse data problem can be tackled by breaking the probability estimation for the right-hand side of the rule into three parts:
  1. predict the probability of the head constituent, in this case, \( VP(bought, VBD) \)
  2. predict the probability of the constituents on the left of the head, in this case, \( NP(IBM, NNP) \) and \( NP(week, NN) \)
  3. predict the probability of the constituents on the right of the head, in this case, none
Lexicalised PCFGs

Lexicalisation drastically increases the size of the grammar. This leads to the sparse data problem.

- How do we estimate the probabilities of the rule
  \[ S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD) \]
  
- Maximum-likelihood estimation won’t work:
  \[
  \frac{\text{count}(S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD))}{\text{count}(S(bought, VBD))}
  \]

- Sparse data problem can be tackled by breaking the probability estimation for the right-hand side of the rule into three parts:
  1. predict the probability of the head constituent, in this case, \( VP(bought, VBD) \)
  2. predict the probability of the constituents on the left of the head, in this case, \( NP(IBM, NNP) \) and \( NP(week, NN) \)
  3. predict the probability of the constituents on the right of the head, in this case, none
Lexicalised PCFGs

Lexicalisation drastically increases the size of the grammar.
This leads to the sparse data problem.

- How do we estimate the probabilities of the rule
  \[ S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD) \]

- Maximum-likelihood estimation won’t work:
  \[
  \frac{\text{count}(S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD))}{\text{count}(S(bought, VBD))}
  \]

- Sparse data problem can be tackled by breaking the probability estimation for the right-hand side of the rule into three parts:
  1. predict the probability of the head constituent, in this case, \( VP(bought, VBD) \)
  2. predict the probability of the constituents on the left of the head, in this case, \( NP(IBM, NNP) \) and \( NP(week, NN) \)
  3. predict the probability of the constituents on the right of the head, in this case, none
The Collins Parser

Model 1

Given a rule of the form:
\[ \text{LHS} \rightarrow L_n L_{n-1} \ldots L_1 H R_1 \ldots R_{n-1} R_n \]

1. Generate the head of the phrase \( H(hw, ht) \) with probability \( P_h(H(hw, ht)|\text{LHS}, hw, ht) \)

2. Generate modifiers to the left of the head with total \( n+1 \) probability:
\[ \prod_{i=1}^{n+1} P_L(L_i(lw_i, lt_i)|\text{LHS}, H, hw, ht) \]
such that \( L_{n+1}(lw_{n+1}, lt_{n+1}) = \text{STOP} \), and we stop generating once we’ve generated a STOP token.

3. Generate modifiers to the right of the head with total \( n+1 \) probability:
\[ \prod_{i=1}^{n+1} P_R(R_i(rw_i, rt_i)|\text{LHS}, H, hw, ht) \]
such that \( R_{n+1}(rw_{n+1}, rt_{n+1}) = \text{STOP} \), and we stop generating once we’ve generated a STOP token.
The Collins Parser

Model 1
Given a rule of the form:
\[ LHS \rightarrow L_n \ L_{n-1} \ldots L_1 \ H \ R_1 \ldots R_{n-1} R_n \]

1. Generate the head of the phrase \( H(hw, ht) \) with probability \( P_h(H(hw, ht)|LHS, hw, ht) \)

2. Generate modifiers to the left of the head with total probability:
\[
\prod_{i=1}^{n+1} P_L(L_i(lw_i, lt_i)|LHS, H, hw, ht)
\]
such that \( L_{n+1}(lw_{n+1}, lt_{n+1}) = STOP \), and we stop generating once we’ve generated a STOP token.

3. Generate modifiers to the right of the head with total probability:
\[
\prod_{i=1}^{n+1} P_R(R_i(rw_i, rt_i)|LHS, H, hw, ht)
\]
such that \( R_{n+1}(rw_{n+1}, rt_{n+1}) = STOP \), and we stop generating once we’ve generated a STOP token.
The Collins Parser

Model 1

Given a rule of the form:
$LHS \rightarrow L_n L_{n-1}...L_1 H R_1...R_{n-1}R_n$

1. Generate the head of the phrase $H(hw, ht)$ with probability $P_h(H(hw, ht)|LHS, hw, ht)$

2. Generate modifiers to the left of the head with total probability:
   $$\prod_{i=1}^{n+1} P_L(L_i(lw_i, lt_i)|LHS, H, hw, ht)$$
   such that $L_{n+1}(lw_{n+1}, lt_{n+1}) = STOP$, and we stop generating once we’ve generated a STOP token.

3. Generate modifiers to the right of the head with total probability:
   $$\prod_{i=1}^{n+1} P_R(R_i(rw_i, rt_i)|LHS, H, hw, ht)$$
   such that $R_{n+1}(rw_{n+1}, rt_{n+1}) = STOP$, and we stop generating once we’ve generated a STOP token.
The Collins Parser

Model 1

Given a rule of the form:
$LHS \rightarrow L_n L_{n-1}...L_1 H R_1...R_{n-1}R_n$

1. Generate the head of the phrase $H(hw, ht)$ with probability $P_h(H(hw, ht)|LHS, hw, ht)$

2. Generate modifiers to the left of the head with total probability:
$$\prod_{i=1}^{n+1} P_L(L_i(lw_i, lt_i)|LHS, H, hw, ht)$$

such that $L_{n+1}(lw_{n+1}, lt_{n+1}) = STOP$, and we stop generating once we’ve generated a STOP token.

3. Generate modifiers to the right of the head with total probability:
$$\prod_{i=1}^{n+1} P_R(R_i(rw_i, rt_i)|LHS, H, hw, ht)$$

such that $R_{n+1}(rw_{n+1}, rt_{n+1}) = STOP$, and we stop generating once we’ve generated a STOP token.
The Collins Parser

Model 1 Example

► We want to calculate

\[ P(S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD)) \]

► Work out the following probabilities and multiply the result

1. \[ P(VP(bought, VBD)|S, bought, VBD) \]
2. \[ P(NP(IBM, NNP)|S, VP, bought, VBD) \]
3. \[ P(NP(week, NN)|S, VP, bought, VBD) \]
The Collins Parser

Model 1 Example

- We want to calculate

\[ P(S(bought, VBD) \rightarrow NP(week, NN)\ NP(IBM, NNP)\ VP(bought, VBD)) \]

- Work out the following probabilities and multiply the result

1. \[ P(VP(bought, VBD)|S, bought, VBD) \]
2. \[ P(NP(IBM, NNP)|S, VP, bought, VBD) \]
3. \[ P(NP(week, NN)|S, VP, bought, VBD) \]
The Collins Parser

Model 1 Example

- We want to calculate
  \[ P(S(bought, VBD) \rightarrow NP(week, NN) \; NP(IBM, NNP) \; VP(bought, VBD)) \]
- Work out the following probabilities and multiply the result
  1. \[ P(VP(bought, VBD)|S, bought, VBD) \]
  2. \[ P(NP(IBM, NNP)|S, VP, bought, VBD) \]
  3. \[ P(NP(week, NN)|S, VP, bought, VBD) \]
The Collins Parser

Model 1 Example

- We want to calculate
  \[ P(S(bought, VBD) \rightarrow NP(week, NN) \quad NP(IBM, NNP) \quad VP(bought, VBD)) \]

- Work out the following probabilities and multiply the result
  1. \[ P(VP(bought, VBD)|S, bought, VBD) \]
  2. \[ P(NP(IBM, NNP)|S, VP, bought, VBD) \]
  3. \[ P(NP(week, NN)|S, VP, bought, VBD) \]
The Collins Parser

Model 1 Example

- We want to calculate
  \[ P(S(bought, VBD) \rightarrow NP(week, NN) \ NP(IBM, NNP) \ VP(bought, VBD)) \]

- Work out the following probabilities and multiply the result
  
  1. \[ P(VP(bought, VBD)|S, bought, VBD) \]
  2. \[ P(NP(IBM, NNP)|S, VP, bought, VBD) \]
  3. \[ P(NP(week, NN)|S, VP, bought, VBD) \]
The Collins Parser

Model 1 Example

- We want to calculate
  \[ P(S(bought, VBD) \rightarrow NP(week, NN) \  NP(IBM, NNP) \  VP(bought, VBD)) \]
- Work out the following probabilities and multiply the result
  1. \[ P(VP(bought, VBD)|S, bought, VBD) \]
  2. \[ P(NP(IBM, NNP)|S, VP, bought, VBD) \]
  3. \[ P(NP(week, NN)|S, VP, bought, VBD) \]
The Collins Parser

- In Model 1, a distance function is also included in the conditioning information for the left and right modifiers.
- This is used to measure the number of words between the current modifier and the head.
- It has the effect of preferring right branching structures and dispreferring dependencies which cross a verb.

Other Linguistically Motivated Refinements
1. Distinction between recursive and base NPs
2. Special features for coordination and punctuation

- Model 2 incorporates verb subcategorisation information.
- Model 3 incorporates long distance dependency information.
The Collins Parser

- In Model 1, a distance function is also included in the conditioning information for the left and right modifiers.
- This is used to measure the number of words between the current modifier and the head.
- It has the effect of preferring right branching structures and disferring dependencies which cross a verb.
- **Other Linguistically Motivated Refinements**
  1. Distinction between recursive and base NPs
  2. Special features for coordination and punctuation
- Model 2 incorporates verb subcategorisation information.
- Model 3 incorporates long distance dependency information.
The Collins Parser

- In Model 1, a distance function is also included in the conditioning information for the left and right modifiers.
- This is used to measure the number of words between the current modifier and the head.
- It has the effect of preferring right branching structures and disferring dependencies which cross a verb.
- Other Linguistically Motivated Refinements
  1. Distinction between recursive and base NPs
  2. Special features for coordination and punctuation
- Model 2 incorporates verb subcategorisation information.
- Model 3 incorporates long distance dependency information.
The Collins Parser

Smoothing

- Smoothing is carried using a linear interpolation of three models
  \[ \lambda_1 e_1 + (1 - \lambda_1)(\lambda_2 e_2 + (1 - \lambda_2)e_3) \]
- \( e_1 \) is the MLE of the fully lexicalised model, \( e_2 \) the MLE of the model which omits the head word from the conditioning information and \( e_3 \) is the MLE of the model which omits the head word and head tag
- Weights are set using Witten-Bell discounting

Accuracy

- Achieves Parseval f-scores of approx. 88% on WSJ23
- Models 2 and 3 slightly more accurate than Model 1

Parsing Algorithm

- Version of probabilistic CKY
- Complexity of lexicalised CFG chart parsing is \( n^4 \) or \( n^5 \)
The Collins Parser

Smoothing

- Smoothing is carried using a linear interpolation of three models
  \[ \lambda_1 e_1 + (1 - \lambda_1)(\lambda_2 e_2 + (1 - \lambda_2)e_3) \]
- \( e_1 \) is the MLE of the fully lexicalised model, \( e_2 \) the MLE of the model which omits the head word from the conditioning information and \( e_3 \) is the MLE of the model which omits the head word and head tag
- Weights are set using Witten-Bell discounting

Accuracy

- Achieves Parseval f-scores of approx. 88% on WSJ23
- Models 2 and 3 slightly more accurate than Model 1

Parsing Algorithm

- Version of probabilistic CKY
- Complexity of lexicalised CFG chart parsing is \( n^4 \) or \( n^5 \)
The Collins Parser

Smoothing

- Smoothing is carried using a linear interpolation of three models:
  \[ \lambda_1 e_1 + (1 - \lambda_1)(\lambda_2 e_2 + (1 - \lambda_2)e_3) \]
- \( e_1 \) is the MLE of the fully lexicalised model, \( e_2 \) the MLE of the model which omits the head word from the conditioning information and \( e_3 \) is the MLE of the model which omits the head word and head tag
- Weights are set using Witten-Bell discounting

Accuracy

- Achieves Parseval f-scores of approx. 88% on WSJ23
- Models 2 and 3 slightly more accurate than Model 1

Parsing Algorithm

- Version of probabilistic CKY
- Complexity of lexicalised CFG chart parsing is \( n^4 \) or \( n^5 \)
Statistical Parsing: an Overview

based on a draft chapter 14 of the 2nd edition of “Speech and Language Processing” by Jurafsky and Martin
Jennifer Foster

Statistical Parsing: What and Why?

PCFGs

Probabilistic CKY

Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers

Collins Parser

Charniak Parser

Unlexicalised Parsing

 Discriminative Parsing

Discriminative Reranking

Discriminative Dynamic Programming

The Charniak Parser

Main differences between the Charniak Parser and the Collins Parser

1. Breaks down the probability estimation in a similar way to the Collins parser but uses more conditioning information:

1.1 When estimating the probability of a rule $A \rightarrow \beta$, it includes the parent node of $A$ in the conditioning information. This is an important feature.

1.2 Also, includes the grandparent and sister nodes of $A$

1.3 Uses a third-order Markov grammar — a modifier constituent in $\beta$ is conditioned on the head constituent (a la Collins) and the previous two left or right modifiers.

2. Uses the suffix of an unknown word to guess its part-of-speech
The Charniak Parser

Main differences between the Charniak Parser and the Collins Parser

1. Breaks down the probability estimation in a similar way to the Collins parser but uses more conditioning information:
   1.1 When estimating the probability of a rule $A \rightarrow \beta$, it includes the parent node of $A$ in the conditioning information. This is an important feature.
   1.2 Also, includes the grandparent and sister nodes of $A$
   1.3 Uses a third-order Markov grammar — a modifier constituent in $\beta$ is conditioned on the head constituent (a la Collins) and the previous two left or right modifiers.

2. Uses the suffix of an unknown word to guess its part-of-speech
The Charniak Parser

Main differences between the Charniak Parser and the Collins Parser

1. Breaks down the probability estimation in a similar way to the Collins parser but uses more conditioning information:
   1.1 When estimating the probability of a rule $A \rightarrow \beta$, it includes the parent node of $A$ in the conditioning information. This is an important feature.
   1.2 Also, includes the grandparent and sister nodes of $A$
   1.3 Uses a third-order Markov grammar — a modifier constituent in $\beta$ is conditioned on the head constituent (a la Collins) and the previous two left or right modifiers.

2. Uses the suffix of an unknown word to guess its part-of-speech
The Charniak Parser

Main differences between the Charniak Parser and the Collins Parser

1. Breaks down the probability estimation in a similar way to the Collins parser but uses more conditioning information:
   1.1 When estimating the probability of a rule $A \rightarrow \beta$, it includes the parent node of $A$ in the conditioning information. This is an important feature.
   1.2 Also, includes the grandparent and sister nodes of $A$
   1.3 Uses a third-order Markov grammar — a modifier constituent in $\beta$ is conditioned on the head constituent (à la Collins) and the previous two left or right modifiers.

2. Uses the suffix of an unknown word to guess its part-of-speech
The Charniak Parser

Main differences between the Charniak Parser and the Collins Parser

1. Breaks down the probability estimation in a similar way to the Collins parser but uses more conditioning information:
   1.1 When estimating the probability of a rule $A \rightarrow \beta$, it includes the parent node of $A$ in the conditioning information. This is an important feature.
   1.2 Also, includes the grandparent and sister nodes of $A$
   1.3 Uses a third-order Markov grammar — a modifier constituent in $\beta$ is conditioned on the head constituent (ala Collins) and the previous two left or right modifiers.

2. Uses the suffix of an unknown word to guess its part-of-speech
Unlexicalised Parsing

- History-based lexicalised parsers achieve f-scores in the range 87-89% on WSJ23
- Klein and Manning (2003) show that it is possible to achieve an f-score of approximately 86% without lexicalisation:
  - A node in a parse tree is annotated with its parent node - parent annotation
  - This means that a subject noun phrase is annotated with the category $S, NP^S$, and a direct object noun phrase will be annotated with the category $VP, NP^VP$.
  - Other transformations can be carried out, e.g. splitting pre-terminal categories into more fine-grained categories
Outline

Statistical Parsing: What and Why?

PCFGs
  Probabilistic CKY
  Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
  Collins Parser
  Charniak Parser
  Unlexicalised Parsing

Discriminative Parsing
  Discriminative Reranking
  Discriminative Dynamic Programming

Dependency Parsing

Conclusion
Discriminative versus Generative Statistical Parsing

Generative Statistical Parsing
The probabilistic model is based on the generative derivation of a sentence. The model gives us the probability of a sentence by summing the probabilities of the derivations.

Discriminative Statistical Parsing
More flexible probability models which can incorporate information from a variety of sources:

1. Global facts about tree structure
2. Structure of previous sentences
3. Text genre
4. Facts about the speaker (e.g. gender)

Two Types of Discriminative Parsing
1. Discriminative Reranking
2. Discriminative Dynamic Programming
Discriminative versus Generative Statistical Parsing

Generative Statistical Parsing
The probabilistic model is based on the generative derivation of a sentence. The model gives us the probability of a sentence by summing the probabilities of the derivations.

Discriminative Statistical Parsing
More flexible probability models which can incorporate information from a variety of sources:

1. Global facts about tree structure
2. Structure of previous sentences
3. Text genre
4. Facts about the speaker (e.g. gender)

Two Types of Discriminative Parsing
1. Discriminative Reranking
2. Discriminative Dynamic Programming
Discriminative Reranking

- Generative parser outputs an n-best list of parse trees
  - Features are extracted from the parse trees
    - Parse probabilities
    - The CFG rules used
    - Structural properties of trees (e.g. degree of parallelism)
  - Log-linear models are often used
  - Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
  - Disadvantage: final result will depend on the quality of the n-best list
Discriminative Reranking

- Generative parser outputs an n-best list of parse trees
- Features are extracted from the parse trees
  - Parse probabilities
  - The CFG rules used
  - Structural properties of trees (e.g. degree of parallelism)
- Log-linear models are often used
- Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
- Disadvantage: final result will depend on the quality of the n-best list
 Discriminative Reranking

- Generative parser outputs an n-best list of parse trees
- Features are extracted from the parse trees
  - Parse probabilities
  - The CFG rules used
  - Structural properties of trees (e.g. degree of parallelism)
- Log-linear models are often used
- Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
- Disadvantage: final result will depend on the quality of the n-best list
Discriminative Reranking

- Generative parser outputs an n-best list of parse trees
- Features are extracted from the parse trees
  - Parse probabilities
  - The CFG rules used
    - Structural properties of trees (e.g. degree of parallelism)
- Log-linear models are often used
- Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
- Disadvantage: final result will depend on the quality of the n-best list
Discriminative Reranking

- Generative parser outputs an n-best list of parse trees
- Features are extracted from the parse trees
  - Parse probabilities
  - The CFG rules used
  - Structural properties of trees (e.g. degree of parallelism)
- Log-linear models are often used
- Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
- Disadvantage: final result will depend on the quality of the n-best list
Statistical Parsing: an Overview

based on a draft chapter 14 of the 2nd edition of “Speech and Language Processing” by Jurafsky and Martin
Jennifer Foster

Discriminative Reranking

- Generative parser outputs an n-best list of parse trees
- Features are extracted from the parse trees
  - Parse probabilities
  - The CFG rules used
  - Structural properties of trees (e.g. degree of parallelism)
- Log-linear models are often used
  - Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
  - Disadvantage: final result will depend on the quality of the n-best list
Discriminative Reranking

- Generative parser outputs an n-best list of parse trees
- Features are extracted from the parse trees
  - Parse probabilities
  - The CFG rules used
  - Structural properties of trees (e.g. degree of parallelism)
- Log-linear models are often used
- Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
  - Disadvantage: final result will depend on the quality of the n-best list
Discriminative Reranking

- Generative parser outputs an \( n \)-best list of parse trees
- Features are extracted from the parse trees
  - Parse probabilities
  - The CFG rules used
  - Structural properties of trees (e.g. degree of parallelism)
- Log-linear models are often used
- Charniak and Johnson (2005) reranker: boosts f-score 2 percentage points to 91.3%
- Disadvantage: final result will depend on the quality of the \( n \)-best list
Discriminative Dynamic Programming

- All parses are stored (compactly) in the chart
- The best parse is then selected using a discriminative probabilistic model
- Example: CCG parser (Clark and Curran, 2004)
Discriminative Dynamic Programming

- All parses are stored (compactly) in the chart
- The best parse is then selected using a discriminative probabilistic model
- Example: CCG parser (Clark and Curran, 2004)
Discriminative Dynamic Programming

- All parses are stored (compactly) in the chart
- The best parse is then selected using a discriminative probabilistic model
- Example: CCG parser (Clark and Curran, 2004)
Outline

Statistical Parsing: What and Why?

PCFGs
  Probabilistic CKY
  Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
  Collins Parser
  Charniak Parser
  Unlexicalised Parsing

Discriminative Parsing
  Discriminative Reranking
  Discriminative Dynamic Programming

Dependency Parsing

Conclusion
Dependency Parsing

Dependency parsers return a different type of structural analysis to phrase structure parsers. They return dependency graphs:

- **the nodes** are the *words* in the sentence
- **the arcs** are the *dependency relationships* between the words

Phrase structure is not explicitly represented.
Dependency Parsing

We distinguish labelled and non-labelled dependency graphs.

- In a labelled dependency graph, the nature of the dependency relationship is specified.
- Typical dependency relationships: subj, obj, nmod, vmod

We distinguish projective and non-projective dependency graphs.

- Non-projectivity identifiable with crossing dependencies.
- Used for representing certain types of non-distance dependencies, e.g. *What did economic news have little effect on?*
- Most dependency-based linguistic theories assume *non-projectivity*
- Most dependency-based parsing systems assume *projectivity*
Probabilistic Dependency Parsing

Broadly speaking, there are two types of probabilistic dependency parsers.

1. Transition-based systems
2. Graph-based systems

Transition-based systems

- Probability model predicts the next action of the parser
- Deterministic shift-reduce parsing
- Example: Malt Parser (Nivre et al. 2006)

Graph-based systems

- Probability model predicts the entire graph for a sentence from the set of all possible graphs
- Example: MST Parser (Mc Donald et al. 2006)
Probabilistic Dependency Parsing

Broadly speaking, there are two types of probabilistic dependency parsers.

1. Transition-based systems
2. Graph-based systems

Transition-based systems

- Probability model predicts the next action of the parser
- Deterministic shift-reduce parsing
- Example: Malt Parser (Nivre et al. 2006)

Graph-based systems

- Probability model predicts the entire graph for a sentence from the set of all possible graphs
- Example: MST Parser (McDonald et al. 2006)
Probabilistic Dependency Parsing

Broadly speaking, there are two types of probabilistic dependency parsers.

1. Transition-based systems
2. Graph-based systems

Transition-based systems

- Probability model predicts the next action of the parser
- Deterministic shift-reduce parsing
- Example: Malt Parser (Nivre et al. 2006)

Graph-based systems

- Probability model predicts the entire graph for a sentence from the set of all possible graphs
- Example: MST Parser (McDonald et al. 2006)
Outline

Statistical Parsing: What and Why?

PCFGs
  Probabilistic CKY
  Obtaining Probabilistic Grammars

Limitations of PCFG Parsing

Generative, History-Based Lexicalised Parsers
  Collins Parser
  Charniak Parser
  Unlexicalised Parsing

Discriminative Parsing
  Discriminative Reranking
  Discriminative Dynamic Programming

Dependency Parsing

Conclusion
But while there is room for improvement I believe that it is now or will soon be time to stop working on improving labeled precision/recall per se... At some point we should move on. Charniak (1997)

What now?
1. Domain adaptation
2. Task-based evaluation
3. Semantic parsing
Conclusion

But while there is room for improvement I believe that it is now or will soon be time to stop working on improving labeled precision/recall per se... At some point we should move on. Charniak (1997)

What now?

1. Domain adaptation
2. Task-based evaluation
3. Semantic parsing
Charniak, E.: Statistical techniques for natural language parsing.
AI Magazine 18(4) (1997)

Charniak, E.: A maximum-entropy-inspired parser.


Klein, D., Manning, C.: Accurate unlexicalised parsing.

McDonald, R., Lerman, K., Pereira, F.: Multilingual dependency analysis with a two-stage discriminative parser.