Intro NLP Tools

Sporleder & Rehbein

WS 09/10

Approaches to POS tagging

- **rule-based**
  - look up words in the lexicon to get a list of potential POS tags
  - apply hand-written rules to select the best candidate tag
- **probabilistic models**
  - for a string of words $W = w_1, w_2, w_3, ..., w_n$
    find the string of POS tags $T = t_1, t_2, t_3, ..., t_n$
    which maximises $P(T|W)$
    ($\Rightarrow$ the probability of tag T given that the word is W)
  - mostly based on (first or second order) **Markov Models**:
    estimate transition probabilities $\Rightarrow$ How probable is it to see POS tag Z after having seen tag Y on position x_{-1} and tag X on position x_{-2}?

Basic idea of ngram tagger:

- current state only depends on previous n states: $p(t_n|t_{n-2}t_{n-1})$

Approaches to POS tagging

- **rule-based**
  - look up words in the lexicon to get a list of potential POS tags
  - apply hand-written rules to select the best candidate tag
- **probabilistic models**
  - for a string of words $W = w_1, w_2, w_3, ..., w_n$
    find the string of POS tags $T = t_1, t_2, t_3, ..., t_n$
    which maximises $P(T|W)$
    (⇒ the probability of tag T given that the word is W)
  - mostly based on (first or second order) **Markov Models**:
    estimate transition probabilities ⇒ How probable is it to see POS tag Z after having seen tag Y on position x_{-1} and tag X on position x_{-2}?

Basic idea of ngram tagger:

- current state only depends on previous n states: $p(t_n|t_{n-2}t_{n-1})$

Approaches to POS tagging

- **rule-based**
  - look up words in the lexicon to get a list of potential POS tags
  - apply hand-written rules to select the best candidate tag
- **probabilistic models**
  - for a string of words $W = w_1, w_2, w_3, ..., w_n$
    find the string of POS tags $T = t_1, t_2, t_3, ..., t_n$
    which maximises $P(T|W)$
    ($\Rightarrow$ the probability of tag T given that the word is W)
  - mostly based on (first or second order) **Markov Models**:
    estimate transition probabilities $\Rightarrow$ How probable is it to see POS tag Z after having seen tag Y on position x_{-1} and tag X on position x_{-2}?

Basic idea of ngram tagger:

- current state only depends on previous n states: $p(t_n|t_{n-2}t_{n-1})$

How to compute transition probabilities?

- How do we get $p(t_n|t_{n-2}t_{n-1})$?
- many ways to do it...
- e.g. Maximum Likelihood Estimation (MLE)
  - $p(t_n|t_{n-2}t_{n-1}) = \frac{F(t_{n-2}t_{n-1}t_n)}{F(t_{n-2}t_{n-1})}$
  - $\frac{F(the/DET\ \ white/ADJ\ \ house/N)}{F(the/DET\ \ white/ADJ)}$
- Problems:
  - zero probabilities (might be ingrammatical or just rare)
  - unreliable counts for rare events

How to compute transition probabilities?

- How do we get $p(t_n|t_{n-2}t_{n-1})$?
- many ways to do it...
- e.g. Maximum Likelihood Estimation (MLE)
  - $p(t_n|t_{n-2}t_{n-1}) = \frac{F(t_{n-2}t_{n-1}t_n)}{F(t_{n-2}t_{n-1})}$
  - $\frac{F(the/DET\ \ white/ADJ\ \ house/N)}{F(the/DET\ \ white/ADJ)}$
- Problems:
  - zero probabilities (might be ingrammatical or just rare)
  - unreliable counts for rare events

How to compute transition probabilities?

- How do we get $p(t_n|t_{n-2}t_{n-1})$?
- many ways to do it...
- e.g. Maximum Likelihood Estimation (MLE)
  - $p(t_n|t_{n-2}t_{n-1}) = \frac{F(t_{n-2}t_{n-1}t_n)}{F(t_{n-2}t_{n-1})}$
  - $\frac{F(the/DET\ \ white/ADJ\ \ house/N)}{F(the/DET\ \ white/ADJ)}$
- Problems:
  - zero probabilities (might be ingrammatical or just rare)
  - unreliable counts for rare events

How to compute transition probabilities?

- How do we get $p(t_n|t_{n-2}t_{n-1})$?
- many ways to do it...
- e.g. Maximum Likelihood Estimation (MLE)
  - $p(t_n|t_{n-2}t_{n-1}) = \frac{F(t_{n-2}t_{n-1}t_n)}{F(t_{n-2}t_{n-1})}$
  - $\frac{F(the/DET \;\; white/ADJ \;\; house/N)}{F(the/DET \;\; white/ADJ)}$
- Problems:
  - zero probabilities (might be ingrammatical or just rare)
  - unreliable counts for rare events

Treetagger

- probabilistic
- uses decision trees to estimate transition probabilities
  $\Rightarrow$ avoid sparse data problems
- How does it work?
  - decision tree automatically determines the context size used for estimating transition probabilities
  - context: unigrams, bigrams, trigrams as well as negations of them (e.g. t_{n-1}=ADJ and $t_{n-2} \neq$ ADJ and t_{n-3} = DET)
  - probability of an n-gram is determined by following the corresponding path through the tree until a leaf is reached
  - improves on sparse data, avoids zero frequencies

Treetagger

- probabilistic
- uses decision trees to estimate transition probabilities
  ⇒ avoid sparse data problems
- How does it work?
  - decision tree automatically determines the context size used for estimating transition probabilities
  - context: unigrams, bigrams, trigrams as well as negations of them (e.g. t_{n-1}=ADJ and $t_{n-2} \neq$ ADJ and t_{n-3} = DET)
  - probability of an n-gram is determined by following the corresponding path through the tree until a leaf is reached
  - improves on sparse data, avoids zero frequencies

Treetagger

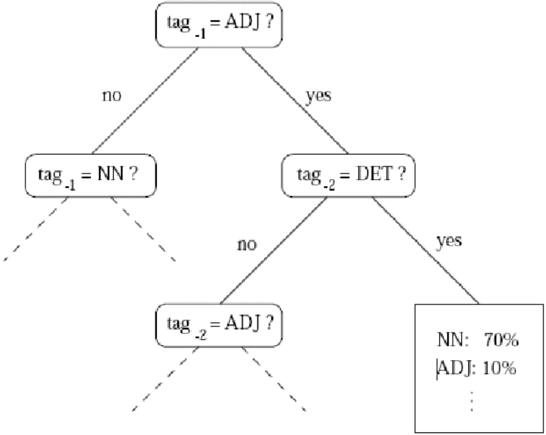


Figure 1: A sample decision tree

Stanford log-linear POS tagger

- ML-based approach based on maximum entropy models
- Idea: improving the tagger by extending the knowledge sources, with a focus on unknown words
- Include linguistically motivated, non-local features:
  - more extensive treatment of capitalization for unknown words
  - features for disambiguation of tense form of verbs
  - features for disambiguating particles from prepositions and adverbs
- Advantage of Maxent: does not assume independence between predictors
- Choose the probability distribution p that has the highest entropy out of those distributions that satisfy a certain set of constraints
- Constraints $\Rightarrow$ statistics from the training data (not restricted to $n-$gram sequences)

C&C Taggers

- Based on maximum entropy models
- highly efficient!
- State-of-the-art results:
  - deleting the correction feature for GIS (Generalised Iterative Scaling)
  - smoothing of parameters of the ME model: replacing simple frequency cutoff by Gaussian prior (form of maximum *a posteriori* estimation rather than a maximum likelihood estimation)
    - penalises models that have very large positive or negative weights
    - allows to use low frequency features without overfitting

The Stanford Parser

- **Factored model**: compute semantic (lexical dependency) and syntactic (PCFG) structures using separate models
- combine results in a new, generative model

$$P(T, D) = P(T)P(D) \tag{1}$$

- Advantages:
  - conceptual simplicity
  - each model can be improved seperately
  - effective A* parsing algorithm (enables efficient, exact inference)

The Stanford Parser

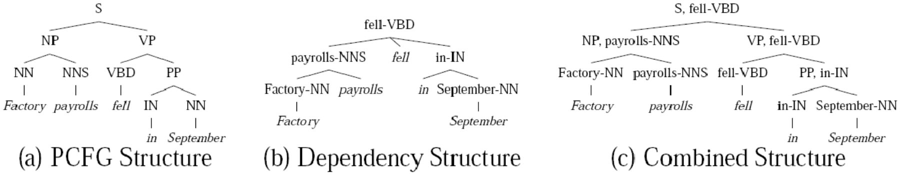


Figure 1: Three kinds of parse structures.

The Stanford Parser

- P(T): use more accurate PCFGs
- annotate tree nodes with contextual markers
  (weaken PCFG independence assumptions)
  - **PCFG-PA**: Parent encoding

    (S (NP (N Man)) (VP (V bites) (NP (N dog))))

    (S (NP^S (N Man)) (VP^S (V bites) (NP^VP (N dog))))
  - **PCFG-LING**: selective parent splitting, order-2 rule markovisation, and linguistically-derived feature splits

The Stanford Parser

- P(T): use more accurate PCFGs
- annotate tree nodes with contextual markers
  (weaken PCFG independence assumptions)
  - **PCFG-PA**: Parent encoding

    (S (NP (N Man)) (VP (V bites) (NP (N dog))))

    (S (NP^S (N Man)) (VP^S (V bites) (NP^VP (N dog))))
  - **PCFG-LING**: selective parent splitting, order-2 rule markovisation, and linguistically-derived feature splits

The Stanford Parser

- P(T): use more accurate PCFGs
- annotate tree nodes with contextual markers
  (weaken PCFG independence assumptions)
  - **PCFG-PA**: Parent encoding

    (S (NP (N Man)) (VP (V bites) (NP (N dog))))

    (S (NP^S (N Man)) (VP^S (V bites) (NP^VP (N dog))))
  - **PCFG-LING**: selective parent splitting, order-2 rule markovisation, and linguistically-derived feature splits

The Stanford Parser

- P(T): use more accurate PCFGs
- annotate tree nodes with contextual markers
  (weaken PCFG independence assumptions)
  - **PCFG-PA**: Parent encoding

    (S (NP (N Man)) (VP (V bites) (NP (N dog))))

    (S (NP^S (N Man)) (VP^S (V bites) (NP^VP (N dog))))
  - **PCFG-LING**: selective parent splitting, order-2 rule markovisation, and linguistically-derived feature splits

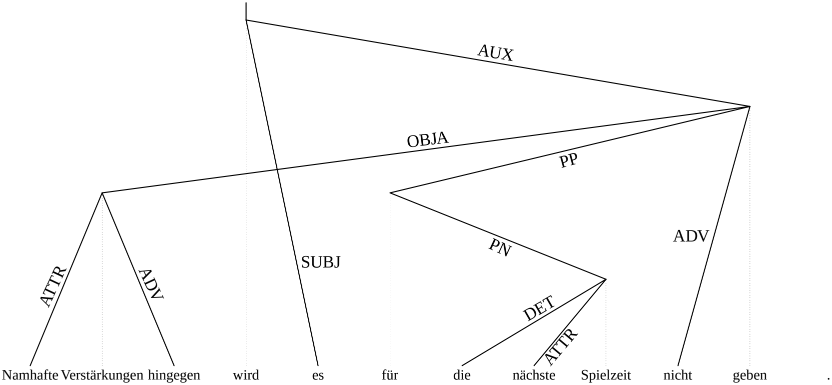
The Stanford Parser

- P(D): lexical dependency models over tagged words
  1. generate head of constituent
  2. generate right dependents until a STOP token is generated
  3. generate left dependents until a STOP token is generated
- word-word dependency models are sparse ⇒ smoothing needed
  - **DEP-BASIC**: generate a dependent conditioned on the head and direction → can capture bilexical selectional preferences, such as the affinity between *payrolls* and *fell*
  - **DEP-VAL**: condition not only on direction, but also on distance and valence

The Stanford Parser

- P(D): lexical dependency models over tagged words
  1. generate head of constituent
  2. generate right dependents until a STOP token is generated
  3. generate left dependents until a STOP token is generated
- word-word dependency models are sparse ⇒ smoothing needed
  - **DEP-BASIC**: generate a dependent conditioned on the head and direction → can capture bilexical selectional preferences, such as the affinity between *payrolls* and *fell*
  - **DEP-VAL**: condition not only on direction, but also on distance and valence

Dependency Tree



"However, there won't be considerable reinforcements for the next playing time"

The Stanford Parser

1. Extract the PCFG sub-model and set up the PCFG parser
2. Use the PCFG parser to find outside scores $\alpha_{PCFG}(e)$ for each edge
3. Extract the dependency sub-model and set up the dependency parser
4. Use the dependency parser to find outside scores $\alpha_{DEP}(e)$ for each edge
5. Combine PCFG and dependency sub-models into the lexicalized model
6. Form the combined outside estimate $a(e) = \alpha_{PCFG}(e) + \alpha_{DEP}(e)$
7. Use the lexicalized A* parser, with $a(e)$ as an A* estimate of $\alpha(e)$

The Berkeley Parser

- Observed treebank categories too coarse-grained
- Idea: treebank refinement using latent variables
  - learn an optimally refined grammar for parsing
  - refine the observed trees with latent variables and learn subcategories
  - basic nonterminal symbols are alternately split and merged to maximize the likelihood of the training treebank

The Berkeley Parser

Start with a minimal X-Bar grammar and learn increasingly refined grammars in a hierarchical **split-and-merge** fashion

1. start with a simple X-bar grammar
2. binarise the trees
3. split-and-merge technique:
   - repeatedly split and re-train the grammar
   - use Expectation Maximisation (EM) to learn a new grammar whose nonterminals are subsymbols of the original nonterminals
4. in each iteration, initialize EM with results of the previous round's grammar
5. split every previous symbol in two
6. after training all splits, measure for each one the loss in likelihood incurred by removing (merging) it
   $\Rightarrow$ keep the ones whose removal causes a considerable loss

The Berkeley Parser

Start with a minimal X-Bar grammar and learn increasingly refined grammars in a hierarchical **split-and-merge** fashion

1. start with a simple X-bar grammar
2. binarise the trees
3. split-and-merge technique:
   - repeatedly split and re-train the grammar
   - use Expectation Maximisation (EM) to learn a new grammar whose nonterminals are subsymbols of the original nonterminals
4. in each iteration, initialize EM with results of the previous round's grammar
5. split every previous symbol in two
6. after training all splits, measure for each one the loss in likelihood incurred by removing (merging) it
   ⇒ keep the ones whose removal causes a considerable loss

The Berkeley Parser

Start with a minimal X-Bar grammar and learn increasingly refined grammars in a hierarchical **split-and-merge** fashion

1. start with a simple X-bar grammar
2. binarise the trees
3. split-and-merge technique:
   - repeatedly split and re-train the grammar
   - use Expectation Maximisation (EM) to learn a new grammar whose nonterminals are subsymbols of the original nonterminals
4. in each iteration, initialize EM with results of the previous round's grammar
5. split every previous symbol in two
6. after training all splits, measure for each one the loss in likelihood incurred by removing (merging) it
   $\Rightarrow$ keep the ones whose removal causes a considerable loss

The Berkeley Parser

Start with a minimal X-Bar grammar and learn increasingly refined grammars in a hierarchical **split-and-merge** fashion

1. start with a simple X-bar grammar
2. binarise the trees
3. split-and-merge technique:
   - repeatedly split and re-train the grammar
   - use Expectation Maximisation (EM) to learn a new grammar whose nonterminals are subsymbols of the original nonterminals
4. in each iteration, initialize EM with results of the previous round's grammar
5. split every previous symbol in two
6. after training all splits, measure for each one the loss in likelihood incurred by removing (merging) it
   $\Rightarrow$ keep the ones whose removal causes a considerable loss

The Berkeley Parser

Start with a minimal X-Bar grammar and learn increasingly refined grammars in a hierarchical **split-and-merge** fashion

1. start with a simple X-bar grammar
2. binarise the trees
3. split-and-merge technique:
   - repeatedly split and re-train the grammar
   - use Expectation Maximisation (EM) to learn a new grammar whose nonterminals are subsymbols of the original nonterminals
4. in each iteration, initialize EM with results of the previous round's grammar
5. split every previous symbol in two
6. after training all splits, measure for each one the loss in likelihood incurred by removing (merging) it
   $\Rightarrow$ keep the ones whose removal causes a considerable loss

The Berkeley Parser

Start with a minimal X-Bar grammar and learn increasingly refined grammars in a hierarchical **split-and-merge** fashion

1. start with a simple X-bar grammar
2. binarise the trees
3. split-and-merge technique:
   - repeatedly split and re-train the grammar
   - use Expectation Maximisation (EM) to learn a new grammar whose nonterminals are subsymbols of the original nonterminals
4. in each iteration, initialize EM with results of the previous round's grammar
5. split every previous symbol in two
6. after training all splits, measure for each one the loss in likelihood incurred by removing (merging) it
   $\Rightarrow$ keep the ones whose removal causes a considerable loss

The Berkeley Parser

Start with a minimal X-Bar grammar and learn increasingly refined grammars in a hierarchical **split-and-merge** fashion

1. start with a simple X-bar grammar
2. binarise the trees
3. split-and-merge technique:
   - repeatedly split and re-train the grammar
   - use Expectation Maximisation (EM) to learn a new grammar whose nonterminals are subsymbols of the original nonterminals
4. in each iteration, initialize EM with results of the previous round's grammar
5. split every previous symbol in two
6. after training all splits, measure for each one the loss in likelihood incurred by removing (merging) it

   ⇒ keep the ones whose removal causes a considerable loss

The Berkeley Parser

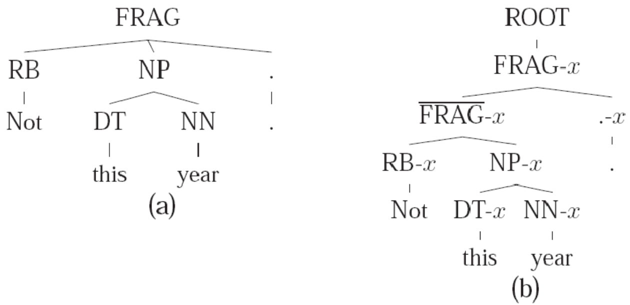


Figure 1: (a) The original tree. (b) The binarized tree with latent variables.

split-and-merge

Splitting provides an increasingly tight fit to the training data, while merging improves generalization and controls grammar size

The Berkeley Parser

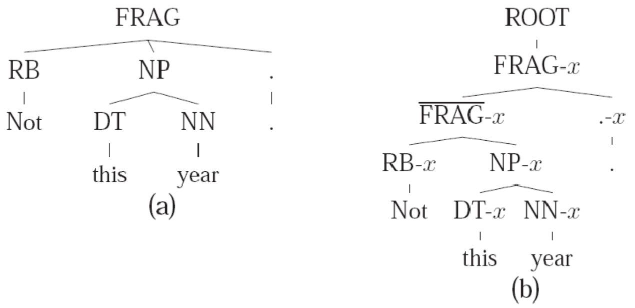


Figure 1: (a) The original tree. (b) The binarized tree with latent variables.

split-and-merge

Splitting provides an increasingly tight fit to the training data, while merging improves generalization and controls grammar size