Introduction to Data-Driven Dependency Parsing

Introductory Course, ESSLLI 2007

Ryan McDonald¹  Joakim Nivre²

¹Google Inc., New York, USA
E-mail: ryanmcd@google.com

²Uppsala University and Växjö University, Sweden
E-mail: nivre@msi.vxu.se
Overview of the Course

- Dependency parsing (Joakim)
- Machine learning methods (Ryan)
- **Transition-based models** (Joakim)
- Graph-based models (Ryan)
- Loose ends (Joakim, Ryan):
  - Other approaches
  - Empirical results
  - Available software
Notation Reminder

- Sentence $x = w_0, w_1, \ldots, w_n$, with $w_0 = root$
- $L = \{l_1, \ldots, l_{|L|}\}$ set of permissible arc labels
- Let $G = (V, A)$ be a dependency graph for sentence $x$ where:
  - $V = \{0, 1, \ldots, n\}$ is the vertex set
  - $A$ is the arc set, i.e., $(i, j, k) \in A$ represents a dependency from $w_i$ to $w_j$ with label $l_k \in L$
- By the usual definition, $G$ is a tree
Data-Driven Parsing

- Goal: Learn a good predictor of dependency graphs
- Input: $x$
- Output: dependency graph/tree $G$
- This lecture:
  - Parameterize parsing by transitions
  - Learn to predict transitions given the input and a history
  - Predict new graphs using deterministic parsing algorithm
- Next lecture:
  - Parameterize parsing by dependency arcs
  - Learn to predict entire graphs given the input
  - Predict new graphs using spanning tree algorithms
Lecture 3: Outline

- Transition systems
- Deterministic classifier-based models
  - Parsing algorithm
  - Stack-based and list-based transition systems
  - Classifier-based parsing
- Pseudo-projective parsing
Transition Systems

A transition system for dependency parsing is a quadruple $S = (C, T, c_s, C_t)$, where

1. $C$ is a set of configurations, each of which contains a buffer $\beta$ of (remaining) nodes and a set $A$ of dependency arcs,
2. $T$ is a set of transitions, each of which is a (partial) function $t : C \rightarrow C$,
3. $c_s$ is an initialization function, mapping a sentence $x = w_0, w_1, \ldots, w_n$ to a configuration with $\beta = [1, \ldots, n]$,
4. $C_t \subseteq C$ is a set of terminal configurations.

Note:

- A configuration represents a parser state.
- A transition represents a parsing action (parser state update).
Transition Sequences

Let $S = (C, T, c_s, C_t)$ be a transition system.

A transition sequence for a sentence $x = w_0, w_1, \ldots, w_n$ in $S$ is a sequence $C_{0,m} = (c_0, c_1, \ldots, c_m)$ of configurations, such that

1. $c_0 = c_s(x)$,
2. $c_m \in C_t$,
3. for every $i$ ($1 \leq i \leq m$), $c_i = t(c_{i-1})$ for some $t \in T$.

The parse assigned to $x$ by $C_{0,m}$ is the dependency graph $G_{c_m} = (\{0, 1, \ldots, n\}, A_{c_m})$, where $A_{c_m}$ is the set of dependency arcs in $c_m$. 
Deterministic Parsing

- An oracle for a transition system $S = (C, T, c_s, C_t)$ is a function $o : C \rightarrow T$.
- Given a transition system $S = (C, T, c_s, C_t)$ and an oracle $o$, deterministic parsing can be achieved by the following simple algorithm:

\[
\text{Parse}(x = (w_0, w_1, \ldots, w_n)) \\
1. \quad c \leftarrow c_s(x) \\
2. \quad \text{while } c \not\in C_t \\
3. \quad c = [o(c)](c) \\
4. \quad \text{return } G_c
\]

- NB: Oracles can be approximated by classifiers (cf. lecture 2).
Stack-Based Transition Systems

- A stack-based configuration for a sentence \( x = w_0, w_1, \ldots, w_n \) is a triple \( c = (\sigma, \beta, A) \), where
  1. \( \sigma \) is a stack of tokens \( i \leq m \) (for some \( m \leq n \)),
  2. \( \beta \) is a buffer of tokens \( j > m \),
  3. \( A \) is a set of dependency arcs such that \( G = (\{0, 1, \ldots, n\}, A) \) is a dependency graph for \( x \).

- A stack-based transition system is a quadruple \( S = (C, T, c_s, C_t) \), where
  1. \( C \) is the set of all stack-based configurations,
  2. \( c_s(x = w_0, w_1, \ldots w_n) = ([0], [1, \ldots, n], \emptyset) \),
  3. \( T \) is a set of transitions, each of which is a function \( t : C \rightarrow C \),
  4. \( C_t = \{ c \in C | c = (\sigma, [\], A) \} \).

- Notation:
  - \( \sigma|i = \) stack with top \( i \) (| left-associative)
  - \( i|\beta = \) buffer with next token \( i \) (| right-associative)
Shift-Reduce Dependency Parsing

- Transitions:
  - **Left-Arc**
    
    $$(\sigma | i, j | \beta, A) \Rightarrow (\sigma, j | \beta, A \cup \{(j, i, k)\})$$
  - **Right-Arc**
    
    $$(\sigma | i, j | \beta, A) \Rightarrow (\sigma, i | \beta, A \cup \{(i, j, k)\})$$
  - **Shift**
    
    $$(\sigma, i | \beta, A) \Rightarrow (\sigma | i, \beta, A)$$

- Preconditions:
  - **Left-Arc**
    
    $$\neg [i = 0]$$
    $$\neg \exists i' \exists k' [(i', i, k') \in A]$$
  - **Right-Arc**
    
    $$\neg \exists i' \exists k' [(i', j, k') \in A]$$
Example: Shift-Reduce Parsing

\[ \text{root}_0 \sigma [\text{Economic}_1 \text{ news}_2 \text{ had}_3 \text{ little}_4 \text{ effect}_5 \text{ on}_6 \text{ financial}_7 \text{ markets}_8 .9] \beta \]
Example: Shift-Reduce Parsing

\[ [\text{root}_0 \ \text{Economic}_1]_\sigma \ [\text{news}_2 \ \text{had}_3 \ \text{little}_4 \ \text{effect}_5 \ \text{on}_6 \ \text{financial}_7 \ \text{markets}_8 \ .9]_\beta \]

Shift
Example: Shift-Reduce Parsing

\[
\text{[root}_0\text{]}_\sigma \quad \text{Economic}_1 \quad [\text{news}_2 \quad \text{had}_3 \quad \text{little}_4 \quad \text{effect}_5 \quad \text{on}_6 \quad \text{financial}_7 \quad \text{markets}_8 \quad .9]_\beta
\]

\text{Left-Arc}_{nmod}
Example: Shift-Reduce Parsing

\[
[ \text{root}_0 \ \text{Economic}_1 \ \text{news}_2 ]_{\sigma} \ [\text{had}_3 \ \text{little}_4 \ \text{effect}_5 \ \text{on}_6 \ \text{financial}_7 \ \text{markets}_8 .9]_{\beta}
\]

Shift
Example: Shift-Reduce Parsing

\[
\begin{array}{c}
\text{root}_0 \quad \text{Economic}_1 \quad \text{news}_2 \quad \text{[had}_3 \quad \text{little}_4 \quad \text{effect}_5 \quad \text{on}_6 \quad \text{financial}_7 \quad \text{markets}_8 \quad \text{.9}_9 \beta
\end{array}
\]

Left-Arc \_s_{bj}
Example: Shift-Reduce Parsing

[\text{root}_0 \ \text{Economic}_1 \ \text{news}_2 \ \text{had}_3]_\sigma [\text{little}_4 \ \text{effect}_5 \ \text{on}_6 \ \text{financial}_7 \ \text{markets}_8 .9]_\beta

Shift
Example: Shift-Reduce Parsing

\[ \text{root}_0 \ \text{Economic}_1 \ \text{news}_2 \ \text{had}_3 \ \text{little}_4 ]_\sigma \ [\text{effect}_5 \ \text{on}_6 \ \text{financial}_7 \ \text{markets}_8 ]_\beta \]

Shift
Example: Shift-Reduce Parsing

[\text{root}_0 \text{ Economic}_1 \text{ news}_2 \text{ had}_3[\sigma] \text{ little}_4 \text{ [effect}_5 \text{ on}_6 \text{ financial}_7 \text{ markets}_8\text{ .}_9[\beta]}

\text{Left-Arc_{nmod}}
Example: Shift-Reduce Parsing

\[ \text{Economic news had little effect on financial markets.} \]

Shift
Example: Shift-Reduce Parsing

\[
\text{[root}_0 \text{ Economic}_1 \text{ news}_2 \text{ had}_3 \text{ little}_4 \text{ effect}_5 \text{ on}_6]_\sigma \quad \text{[financial}_7 \text{ markets}_8 \text{ .9]}_\beta
\]

Shift
Example: Shift-Reduce Parsing

\[
\begin{array}{c}
\text{[root}_0 \ \text{Economic}_1 \ \text{news}_2 \ \text{had}_3 \ \text{little}_4 \ \text{effect}_5 \ \text{on}_6 \ \text{financial}_7]_\sigma \ [\text{markets}_8 \ .9]_\beta \\
\end{array}
\]

Shift
Example: Shift-Reduce Parsing

\[ \text{Economic_1 news_2 had_3 little_4 effect_5 on_6}_\sigma \text{ financial_7 [markets_8 .9]}_{\beta} \]

Left-Arc_{nmod}
Example: Shift-Reduce Parsing

[root$_0$ Economic$_1$ news$_2$ had$_3$ little$_4$ effect$_5$]$_\sigma$ [on$_6$ financial$_7$ markets$_8$ .9]$_\beta$

Right-Arc$_{pc}$
Example: Shift-Reduce Parsing

[\text{root}_0 \ \text{Economic}_1 \ \text{news}_2 \ \text{had}_3]_{\sigma} \ \text{little}_4 \ [\text{effect}_5 \ \text{on}_6 \ \text{financial}_7 \ \text{markets}_8]_{\beta}

\text{Right-Arc}_{nmod}
Example: Shift-Reduce Parsing

[\text{root}_0]_\sigma \quad \text{Economic}_1 \quad \text{news}_2 \quad [\text{had}_3 \quad \text{little}_4 \quad \text{effect}_5 \quad \text{on}_6 \quad \text{financial}_7 \quad \text{markets}_8 \cdot 9]_\beta

Right-Arc_{\text{obj}}
Example: Shift-Reduce Parsing

Economic news had little effect on financial markets.

Right-Arc_{pred}
Example: Shift-Reduce Parsing

Right-Arc $p$
Example: Shift-Reduce Parsing

Shift
Theoretical Results

 Complexity:
  - Deterministic shift-reduce parsing has time and space complexity $O(n)$, where $n$ is the length of the input sentence.

 Correctness:
  - For every transition sequence $C_{0,m}$, $G_{c_m}$ is a projective dependency forest (soundness).
  - For every projective dependency forest $G$, there is a transition sequence $C_{0,m}$ such that $G_{c_m} = G$ (completeness).

 Note:
  - A dependency forest is (here) a dependency graph satisfying Root, Single-Head, and Acyclicity (but not Connectedness).
  - A dependency forest $G = (V, A)$ can be transformed into a dependency tree by adding arcs of the form $(0, i, k)$ (for some $l_k \in L$) for every root $i \in V$ ($i \neq 0$).
Variations on Shift-Reduce Parsing

- Empty stack initialization:
  - If we can assume that there is only one node $i$ such that $(0, i, k) \in A$, then we can reduce ambiguity by starting with an empty stack (and adding the arc $(0, i, k)$ after termination).

- Iterative parsing [Yamada and Matsumoto 2003]:
  - Same transition system (with empty stack initialization)$^1$
  - Given a terminal configuration:
    - $(\sigma, [], A) \Rightarrow ([], \sigma, A)$
    - Terminate when $A$ has not been modified in the last iteration.

- Modified transition systems:
  - Arc-eager parsing [Nivre 2003]
  - Non-projective parsing [Attardi 2006]

$^1$NB: Left-Arc $\Rightarrow$ Right, Right-Arc $\Rightarrow$ Left
Arc-Eager Parsing

**Transitions:**

- **Left-Arc** \(k\):
  \[
  (\sigma|i, j|\beta, A) \Rightarrow (\sigma, j|\beta, A \cup \{(j, i, k)\})
  \]

- **Right-Arc** \(k\):
  \[
  (\sigma|i, j|\beta, A) \Rightarrow (\sigma|i|j, \beta, A \cup \{(i, j, k)\})
  \]

- **Reduce**:
  \[
  (\sigma|i, \beta, A) \Rightarrow (\sigma, \beta, A)
  \]

- **Shift**:
  \[
  (\sigma, i|\beta, A) \Rightarrow (\sigma|i, \beta, A)
  \]

**Preconditions:**

- **Left-Arc** \(k\):
  
  \[-[i = 0] \]
  \[-\exists i' \exists k'[\{(i', i, k') \in A\}] \]

- **Right-Arc** \(k\):
  
  \[-\exists i' \exists k'[\{(i', j, k') \in A\}] \]

- **Reduce**:
  
  \[\exists i' \exists k'[\{(i', i, k') \in A\}] \]
Example: Arc-Eager Parsing

$$[\text{root}_0]_\sigma [\text{Economic}_1 \ \text{news}_2 \ \text{had}_3 \ \text{little}_4 \ \text{effect}_5 \ \text{on}_6 \ \text{financial}_7 \ \text{markets}_8 .9]_\beta$$
Example: Arc-Eager Parsing

\[
[\text{root}_0 \text{ Economic}_1]_{\sigma} [\text{news}_2 \text{ had}_3 \text{ little}_4 \text{ effect}_5 \text{ on}_6 \text{ financial}_7 \text{ markets}_8 .9]_{\beta}
\]

Shift
Example: Arc-Eager Parsing

\[ \text{[root}_0]_{\sigma} \text{ Economic}_1 \text{ [news}_2 \text{ had}_3 \text{ little}_4 \text{ effect}_5 \text{ on}_6 \text{ financial}_7 \text{ markets}_8 .9]_\beta \]

Left-Arc \text{nmod}
Example: Arc-Eager Parsing

\[
\text{[root}_0 \text{ Economic}_1 \text{ news}_2]_{\sigma} \ [\text{had}_3 \text{ little}_4 \text{ effect}_5 \text{ on}_6 \text{ financial}_7 \text{ markets}_8 .9]_{\beta}
\]

Shift
Example: Arc-Eager Parsing

[\text{root}_0]_\sigma \text{ Economic}_1 \text{ news}_2 \ [\text{had}_3 \text{ little}_4 \text{ effect}_5 \text{ on}_6 \text{ financial}_7 \text{ markets}_8 .9]_\beta

Left-Arc_{\text{sbj}}
Example: Arc-Eager Parsing

\[ \text{pred} \]

\[ \text{nmod} \] \quad \text{subj} \]

\[ \text{[root}_0 \quad \text{Economic}_1 \quad \text{news}_2 \quad \text{had}_3]_\sigma \quad [\text{little}_4 \quad \text{effect}_5 \quad \text{on}_6 \quad \text{financial}_7 \quad \text{markets}_8 \quad .9]_\beta \]

Right-Arc\text{pred}
Example: Arc-Eager Parsing

\[ \text{pred} \]

\[
\begin{array}{c}
\text{nmod} \\
\text{sbj}
\end{array}
\]

\[
[\text{root}_0 \ \text{Economic}_1 \ \text{news}_2 \ \text{had}_3 \ \text{little}_4]_{\sigma} \ [\text{effect}_5 \ \text{on}_6 \ \text{financial}_7 \ \text{markets}_8]_{\beta}
\]

Shift
Example: Arc-Eager Parsing

\[\text{pred} \quad \text{nmod} \quad \text{sbj} \quad \text{nmod} \]

\[
\text{[root}_0 \quad \text{Economic}_1 \quad \text{news}_2 \quad \text{had}_3]_\sigma \quad \text{little}_4 \quad \text{effect}_5 \quad \text{on}_6 \quad \text{financial}_7 \quad \text{markets}_8 \quad .9]_\beta
\]

Left-Arc_{nmod}
Example: Arc-Eager Parsing

[\text{root}_0 \text{ Economic}_1 \text{ news}_2 \text{ had}_3 \text{ little}_4 \text{ effect}_5 \sigma \text{ [on}_6 \text{ financial}_7 \text{ markets}_8 .9\beta]

Right-Arc_{obj}
Example: Arc-Eager Parsing

Right-Arc$_{nmod}$
Example: Arc-Eager Parsing

\[
\begin{array}{c}
\text{pred} \\
\text{nmod} & \text{sbj} \\
\text{obj} \\
\text{nmod} & \text{nmod}
\end{array}
\]

[\text{root}_0 \quad \text{Economic}_1 \quad \text{news}_2 \quad \text{had}_3 \quad \text{little}_4 \quad \text{effect}_5 \quad \text{on}_6 \quad \text{financial}_7]_{\sigma} \ [\text{markets}_8 \quad .9]_{\beta}

Shift
Example: Arc-Eager Parsing

Left-Arc_{nmod}
Example: Arc-Eager Parsing

Right-Arc\textsubscript{pc}
Example: Arc-Eager Parsing

Reduce
Example: Arc-Eager Parsing

[\textbf{root}_0 \quad \text{Economic}_1 \quad \text{news}_2 \quad \textbf{had}_3 \quad \text{little}_4 \quad \textbf{effect}_5 \text{]}_\sigma \quad \text{on}_6 \quad \text{financial}_7 \quad \text{markets}_8 \quad [.9]_\beta]

Reduce
Example: Arc-Eager Parsing

\[
\begin{array}{c}
\text{pred} \\
\text{nmod} \quad \text{sbj} \quad \text{obj} \\
\text{nmod} \quad \text{nmod} \quad \text{nmod} \\
\text{pc} \\
\end{array}
\]

[\text{root}_0 \quad \text{Economic}_1 \quad \text{news}_2 \quad \text{had}_3]_\sigma \quad \text{little}_4 \quad \text{effect}_5 \quad \text{on}_6 \quad \text{financial}_7 \quad \text{markets}_8 \quad [.]_\beta

Reduce
Example: Arc-Eager Parsing

\[
\begin{align*}
\text{pred} & \quad \text{nmod} & \quad \text{sbj} \\
\text{obj} & \quad \text{nmod} & \quad \text{nmod} \\
\text{pc} & \quad \text{nmod} \\
[\text{root}_0] & \quad \text{Economic}_1 & \quad \text{news}_2 & \quad \text{had}_3 & \quad \text{little}_4 & \quad \text{effect}_5 & \quad \text{on}_6 & \quad \text{financial}_7 & \quad \text{markets}_8 \quad [.9]_\beta
\end{align*}
\]

Reduce
Example: Arc-Eager Parsing

Right-Arc$_p$
Non-Projective Parsing

- New transitions:
  - **NP-Left-Arc**
    \[
    (\sigma|i|i', j|\beta, A) \Rightarrow (\sigma|i'|, j|\beta, A \cup \{(j, i, k)\})
    \]
  - **NP-Right-Arc**
    \[
    (\sigma|i|i', j|\beta, A) \Rightarrow (\sigma|i, i'|\beta, A \cup \{(i, j, k)\})
    \]

- Handles most naturally occurring non-projective dependency relations (94% in the Prague Dependency Treebank).

- More expressive extensions are possible [Attardi 2006].

Introduction to Data-Driven Dependency Parsing
Comparing Algorithms

Expressivity:
- Arc-standard and arc-eager shift-reduce parsing is limited to projective dependency graphs.
- Simple extensions can handle a subset of non-projective dependency graphs.

Complexity:
- Space complexity is $O(n)$ for all deterministic parsers (even with simple extensions).
- Time complexity is $O(n)$ for single-pass parsers, $O(n^2)$ for iterative parsers.

More complex extensions to handle non-projective dependency graphs will affect time complexity.
List-Based Transition Systems

▶ A list-based configuration for a sentence $x = w_0, w_1, \ldots, w_n$ is a quadruple $c = (\lambda_1, \lambda_2, \beta, A)$, where

1. $\lambda_1$ is a list of tokens $i_1 \leq m_1$ (for some $m_1 \leq n$),
2. $\lambda_2$ is a list of tokens $i_2 \leq m_2$ (for some $m_2, m_1 < m_2 \leq n$),
3. $\beta$ is a buffer of tokens $j > m_2$,
4. $A$ is a set of dependency arcs such that $G = (\{0, 1, \ldots, n\}, A)$ is a dependency graph for $x$.

▶ A list-based transition system is a quadruple $S = (C, T, c_s, C_t)$, where

1. $C$ is the set of all list-based configurations,
2. $c_s(x = w_0, w_1, \ldots w_n) = ([0], [], [1, \ldots, n], \emptyset),$
3. $T$ is a set of transitions, each of which is a function $t : C \rightarrow C$,
4. $C_t = \{ c \in C | c = (\lambda_1, \lambda_2, [], A) \}$.

▶ Notation:

- $\lambda_1 | i = \text{list with head } i \text{ and tail } \lambda_1$ (left-associative)
- $i | \lambda_2 = i \text{ and tail } \lambda_2$ (right-associative)
Non-Projective Parsing

- **Transitions:**
  - **Left-Arc**:
    \((\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, i|\lambda_2, j|\beta, A \cup \{(j, i, k)\})\)
  - **Right-Arc**:
    \((\lambda_1|i, \lambda_2, j|\beta, A) \Rightarrow (\lambda_1, i|\lambda_2, j|\beta, A \cup \{(i, j, k)\})\)
  - **No-Arc**:
    \((\lambda_1|i, \lambda_2, \beta, A) \Rightarrow (\lambda_1, i|\lambda_2, \beta, A)\)
  - **Shift**:
    \((\lambda_1, \lambda_2, i|\beta, A) \Rightarrow (\lambda_1 \cdot \lambda_2|i, [], \beta, A)\)

- **Preconditions:**
  - **Left-Arc**:
    \(-[i = 0]
    \,- \exists i' \exists k' [(i', k', i) \in A]
    \,- [i \rightarrow^* j]_A
  - **Right-Arc**:
    \,- \exists i' \exists k' [(i', k', j) \in A]
    \,- [j \rightarrow^* i]_A
Projective Parsing

Transitions:

- **Left-Arc** \(k\):
  \[(\lambda_1|i, \lambda_2|j|\beta, A) \Rightarrow (\lambda_1, \lambda_2, j|\beta, A \cup \{(j, i, k)\})\]

- **Right-Arc** \(k\):
  \[(\lambda_1|i, \lambda_2,j|\beta, A) \Rightarrow (\lambda_1|i|j, [], \beta, A \cup \{(i, k, j)\})\]

- **No-Arc**:
  \[(\lambda_1|i, \lambda_2, \beta, A) \Rightarrow (\lambda_1, i|\lambda_2, \beta, A)\]

- **Shift**:
  \[(\lambda_1, \lambda_2, i|\beta, A) \Rightarrow (\lambda_1.\lambda_2|i, [], \beta, A)\]

Preconditions:

- **Left-Arc**:
  \[\neg[i = 0]\]
  \[\neg\exists i' \exists k'[(i', k', i) \in A]\]

- **Right-Arc**:
  \[\neg\exists i' \exists k'[(i', k', j) \in A]\]

- **No-Arc**:
  \[\exists i' \exists k[(i', k, i) \in A]\]
Theoretical Results

- **Complexity:**
  - Deterministic list-based parsing has **time complexity** $O(n^2)$ and **space complexity** $O(n)$, where $n$ is the length of the input sentence.

- **Correctness:**
  - For every transition sequence $C_{0,m}$, $G_{cm}$ is a (projective) dependency forest (**soundness**).
  - For every (projective) dependency forest $G$, there is a transition sequence $C_{0,m}$ such that $G_{cm} = G$ (**completeness**).
Classifier-Based Parsing

- Data-driven deterministic parsing:
  - Deterministic parsing requires an oracle.
  - An oracle can be approximated by a classifier.
  - A classifier can be trained using treebank data.

- Learning problem:
  - Approximate a function from configurations (represented by feature vectors) to transitions, given a training set of (gold standard) transition sequences.
  - Three issues:
    - How do we represent configurations by feature vectors?
    - How do we derive training data from treebanks?
    - How do we learn classifiers?
Feature Representations

- A feature representation $\mathbf{f}(c)$ of a configuration $c$ is a vector of simple features $\mathbf{f}_i(c)$.
- Typical features are defined in terms of attributes of nodes in the dependency graph.
  - Nodes:
    - Target nodes (top of $\sigma$, head of $\lambda_1$, $\lambda_2$, $\beta$)
    - Linear context (neighbors in $\sigma$, $\lambda_1$, $\lambda_2$, or $\beta$)
    - Structural context (parents, children, siblings given $A$)
  - Attributes:
    - Word form (and/or lemma)
    - Part-of-speech (and morpho-syntactic features)
    - Dependency type (if labeled)
    - Distance (between target tokens)
A Typical Model [Nivre et al. 2006]
Training Data

Training instances have the form \((f(c), t)\), where
1. \(f(c)\) is a feature representation of a configuration \(c\),
2. \(t\) is the correct transition out of \(c\) (i.e., \(o(c) = t\)).

Given a dependency treebank, we can sample the oracle function \(o\) as follows:
- For each sentence \(x\) with (gold standard) dependency graph \(G_x\), we construct a transition sequence \(C_{0,m} = (c_0, c_1, \ldots, c_m)\) such that
  1. \(c_0 = c_s(x)\),
  2. \(G_{c_m} = G_x\),
- For each configuration \(c_i (i < m)\), we construct a training instance \((f(c_i), t_i)\), where \(t_i(c_i) = c_{i+1}\).
Learning Classifiers

- Learning methods:
  - Support vector machines (SVM)
    - Polynomial kernel \( (d \geq 2) \)
    - Different techniques for multiclass classification
    - Training efficiency problematic for large data sets
  - Memory-based learning (MBL)
    - \( k \)-NN classification
    - Different distance functions
    - Parsing efficiency problematic for large data sets
  - Maximum entropy modeling (MaxEnt)
    [Cheng et al. 2005, Attardi 2006]
    - Extremely efficient parsing
    - Slightly less accurate
Pseudo-Projective Parsing

- Technique for non-projective dependency parsing with a data-driven projective parser [Nivre and Nilsson 2005].

- Four steps:
  1. Projectivize dependency graphs in training data, encoding information about transformations in augmented arc labels.
  2. Train projective parser (as usual).
  3. Parse new sentences using projective parser (as usual).
  4. Deprojectivize output dependency graphs by heuristic transformations guided by augmented arc labels.
Pseudo-Projective Parsing

- Projectivize training data:
  - Projective head nearest permissible ancestor of real head
  - Arc label extended with dependency type of real head

```
root Z nich je jen jedna na kvalitu
(out-of) (them) (is) (only) (one) (to) (quality)
```
Pseudo-Projective Parsing

- Projectivize training data:
  - Projective head nearest permissible ancestor of real head
  - Arc label extended with dependency type of real head

AuxK

AuxP

Pred

AuxP↑Sb

Atr

root

Z

(nich)

(je)

(jen)

(jedna)

(na)

(kvalitu)

(out-of)

(Them)

(is)

(only)

(one)

(to)

(quality)
Pseudo-Projective Parsing

- Deprojectivize parser output:
  - Top-down, breadth-first search for real head
  - Search constrained by extended arc label

```
root Z nich je jen jedna na kvalitu
(out-of) (them) (is) (only) (one) (to) (quality)
```
Pseudo-Projective Parsing

- Deprojectivize parser output:
  - Top-down, breadth-first search for real head
  - Search constrained by extended arc label

Below is a diagram illustrating the parsing process with the words:

- root
- Z
- nich
- je
- jen
- jedna
- na
- kvalitu

The words are connected with the following labels:

- Pred
- AuxP
- AuxP↑Sb
- Atr
- Sb
- AuxZ
- Adv

The diagram shows the dependency relationships between the words, indicating the parsing structure.
Summary – Transition-based Methods

- Transition systems
- Deterministic classifier-based parsing
  - Parsing algorithm
  - Stack-based and list-based transitions systems
  - Classifier-based parsing
- Pseudo-projective parsing
References and Further Reading


References and Further Reading


