Part B: Semi-supervised dependency parsing for in-domain texts
Semi-supervised dependency parsing

- Supervised parsing
  - Training: Labeled data
- Semi-supervised parsing
  - Training: Additional unlabeled data + labeled data
In-domain vs out-domain

Annotated data in Domain A

Training

A Parser

Parsing texts in Domain A

Parsing texts in Domain B

In-domain

Out-domain
Semi-supervised dependency parsing

- Typical methods
  - Use whole auto-parsed trees
    - Self-training
    - Co-training

- Other methods?
  - To use partial trees
Semi-supervised dependency parsing

Unlabeled:
- He ate the fish
- The book is mine
- Put it there
- ...

Lexical Level:
- Word clusters, word co-occurrences …

Partial Tree Level:
- Subtree frequencies, subtree likelihood

Whole Tree Level:
- Reliable whole trees as extra training instances

Auto-parsed trees:
- He ate the fish
- The book is mine
- Put it there
- …
Semi-supervised dependency parsing

- Whole tree level
- Partial tree level
- Lexical level
- Whole tree level

- **Approaches:**
  - Self-training
    - Use one parser
    - Select the automatic parses from unlabeled data as extra training examples
  - Co-training
    - Use two parsers
    - Select the automatic parses from two different views from unlabeled data as extra training examples
  - Ambiguity-aware ensemble training
    - Aggregate multi parsers’ outputs into forests
Self-training

- Self-training
  - Step 1: Train a first-stage parser with the human labeled data
  - Step 2: Apply the parser to produce automatic parses for the unlabeled data
  - Step 3: Select some auto-parsed sentences as newly labeled data *(How to select)*
  - Step 4: Train a better parser by combining the human labeled and selected newly auto-parsed data *(How to combine).*
Self-training (McClosky et al., 2006)

- Self-training with a two-stage constituent parser
  - Step 1: train a first-stage generative parser and a second-stage discriminative reranker with the labeled data (Charniak and Johnson, 2005)
  - Step 2: apply the parser and reranker to produce automatic parses for the unlabeled data.
  - Step 3: train a better first-stage parser by combining the labeled and unlabeled data (with corpus weighting).
Self-training (Huang and Harper, 2009)

- Self-training with a single PCFG-LA parser
  - Step 1: train a Berkeley parser with the labeled data (Petrov and Klein, 2007)
  - Step 2: apply the parser to produce automatic parses for the unlabeled data.
  - Step 3: train a better parser by combining the labeled and unlabeled data (with corpus weighting).
Self-training (Huang et al., 2010)

- Self-training with products of PCFG-LA grammars
  - Use the products of PCFG-LA parsers to
    - alleviate the problem that EM tends to get stuck in local maxima;
    - produce better parses for the unlabeled data.
Self-training (dependency parsing)

- For dependency parsing (in-domain)
  - Hard to pick up reliable sentences
    - Use SVM classifier (Kawahara and Uchimoto, 2008) to select reliable sentences
  - Not so successful (so far)
Co-training

- Co-training with two parsers of different views
  (Sarkar, 2001; Steedman et al., 2003; Sagae and Tsujii, 2007)
  - Step 1: select two different parsers, e.g., graph-based and transition-based dependency parsers.
  - Step 2: train the two parsers with the labeled data.
  - Step 3: apply the two parsers to the pool of raw sentences.
  - Step 4: select the reliable parses according to the consistency between the two parsers, and add them into the labeled data.
- Go to Step 2 until no performance improvement on some held-out data.
-Ambiguity-aware ensemble training (Li+ 14)

- Use different parsers to parse the unlabeled data, and aggregate their outputs into forests;
- Use the unlabeled data with forest as extra training data
-Ambiguity-aware ensemble training
(Li+ 14)

- Training objective

$$\mathcal{L}(\mathcal{D}'; \mathbf{w}) = \sum_{i=1}^{M} \log \left( \sum_{d' \in \mathcal{V}_i} p(d'|u_i; \mathbf{w}) \right)$$
Ambiguity-aware ensemble training (Li+ 14)

- Experimental results show
  - Better than co-training/tri-training
  - Diversity of parsers is important!
    - Generative constituent parser is the most useful
Semi-supervised dependency parsing

- Whole tree level
- Partial tree level
- Lexical level
- Partial tree level

- Approaches
  - Use word pairs (Noord, 2007; Chen et al., 2008)
  - Use subtrees (Chen et al., 2009)/DLM (Chen et al., 2012)
  - Use hybrid discriminative and generative models to derive useful cues from unlabeled data (Suzuki et al., 2009).
  - Use meta features (Chen et al., 2013)
- Partial tree level

- With word pairs (Noord, 2007; Chen et.al. 2008)
Word pairs (Noord, 2007; Chen et al. 2008)

- Parsing with word pairs
  - Step 1: Train a baseline parser with the labeled data
  - Step 2: Use the baseline parser to parse the raw sentences
  - Step 3: Collect word pairs (lexical dependencies)
  - Step 4: Represent new features based on word pairs
  - Step 5: Re-train a new parser by combining the base features and new features
Word pairs

• Statistics of word pairs
  • Pointwise mutual information (Noord, 2007)
    \[ I(r(w_1, w_2)) = \log \frac{f(r(w_1, w_2))}{f(r(w_1, \_))f(\_, w_2)} \]
  • Association score between w1 and w2
Word pairs

- Statistics of word pairs
  - Short dependency (Chen et. al. 2008)
    - Word pairs with dependency length=1/2
    - Group the collected word pairs into buckets
- Partial tree level

- With word pairs
  - The information of word pairs is too few.

- With subtrees
With subtrees

- Parsing with subtrees
  - Step1: Train a baseline parser with the labeled data
  - Step2: Use the baseline parser to parse the raw sentences
  - Step3: Extract subtrees from the auto-parsed data
  - Step4: Represent new features based on the extracted subtrees
  - Step5: Re-train a new parser by combining the base features and subtree-based features
Subtree Extraction

- Extract subtrees containing two nodes or three nodes
  - If a subtree contains two nodes, we call it a bigram-subtree.
  - If a subtree contains three nodes, we call it a trigram-subtree.
### Subtree Extraction

**Diagram:**

```
    ROOT
     ↓
    ate
     ↓  ↓
    the fish with
     ↓  ↓  ↓
    a  ate:0-fish:2:1
```

**Table:**

<table>
<thead>
<tr>
<th>Subtree</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>ate</td>
<td>l:1:1-ate:2:0</td>
</tr>
<tr>
<td>ate</td>
<td>ate:1:0-with:2:1</td>
</tr>
<tr>
<td>fish</td>
<td>the:1:1-fish:2:0</td>
</tr>
<tr>
<td>ate</td>
<td>ate:1:0-..:2:1</td>
</tr>
<tr>
<td>fish</td>
<td>ate:1:0-fish:2:1</td>
</tr>
<tr>
<td>fork</td>
<td>a:1:1-fork:2:0</td>
</tr>
<tr>
<td>with</td>
<td>with:1:0-fork:2:1</td>
</tr>
</tbody>
</table>
### Subtree Extraction

<table>
<thead>
<tr>
<th>ate</th>
<th>l:1:1-ate:2:0</th>
<th>ate</th>
<th>ate:1:0-with:2:1</th>
<th>fish</th>
<th>the:1:1-fish:2:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ate</td>
<td>ate:1:0-fish:2:1</td>
<td>ate</td>
<td>ate:1:0-..:2:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fish</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>fork</td>
<td>a:1:1-fork:2:0</td>
<td>with</td>
<td>with:1:0-fork:2:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ate</td>
<td>ate:1:0-fish:2:1-with:3:1</td>
<td>ate</td>
<td>ate:1:0-with:2:1-.:3:1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fish</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>with</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Subtree Labeling

- To group the extracted subtrees into different sets according to their frequencies

- Subtree labels:
  - After testing on development data, subtrees are grouped into three sets: HF, MF, and LF
  - Add new set (ZERO) to refer to the subtrees that are not included
  - Finally, we have four labels: HF, MF, LF, and ZERO
Feature representation

• Parsing model
  • A graph-based MST Parsing model proposed by McDonald et al. (2005) and McDonald and Pereira (2006)
    • First-order features are defined over single graph edges
    • Second-order features are defined on adjacent edges

• Represent two types of features on labeled training data
  • First-order features based on bigram-subtrees
  • Second-order features based on trigram-subtrees
First-order subtree-based features

- Generate the features for a head $h$ and a dependent $d$

- The links form temporary bigram-subtrees

- Subtree-based Features
  - The labels of these bigram-subtrees
  - The labels are conjoined with POS tags of head
  - The labels are conjoined with word forms of head
Second-order subtree-based features

- Generate the features for a head $h$ and a dependent $d_1$, $d_1$’s right-leftmost sibing $d_2$

- The links form temporary trigram-subtrees

- Subtree-based Features
  - The labels of these trigram-subtrees
  - The labels are conjoined with POS tags of head
  - The labels are conjoined with word forms of head
How does this approach work?

Training data
... ate ... fish ...

Test data
... ate ... nut ...

Auto-parsed data
... ate ... fish ...
... ate ... nut ...

Feature: MF

Additional features for unseen tuples
Results from Chen et al. 2009

- On PTB

<table>
<thead>
<tr>
<th></th>
<th>UAS</th>
<th>Complete</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ord1</td>
<td>90.95</td>
<td>37.45</td>
</tr>
<tr>
<td>Ord1s</td>
<td>91.76(+0.81)</td>
<td>40.68</td>
</tr>
<tr>
<td>Ord2</td>
<td>91.71</td>
<td>42.88</td>
</tr>
<tr>
<td>Ord2s</td>
<td>92.51(+0.80)</td>
<td>46.19</td>
</tr>
</tbody>
</table>
- Partial tree level
  - With word pairs
    - The information of word pairs is too few.
  - With subtrees
  - With dependency language model (DLM)
Dependency Language Model

- Standard Language Model (N-gram)
  - Predicts the next word based on the N-1 immediate previous words
  - Can not capture long-distance word relations

- Dependency Language Model (Shen et. al, 2008)
  - Predicts the next child of a head based on the N-1 immediate previous children and the head itself
  - Can capture long-distance word relations
  - Has been used in SMT (Shen et. al, 2008)
Dependency Language Model

- Input sentence: \( x = (x_0, x_1, \ldots, x_i, \ldots, x_n, ) \)
- Dependency tree: \( y \)
- Set for the words having at least one dependent: \( H(y) \)
- Dependency structure with head \( x_h : D_h \)

\[
P(y) = \prod_{x_h \in H(y)} P(D_h)
\]
Dependency Language Model

- $D_h$

\[
P(D_h) = P_L(D_h) \times P_R(D_h) \quad (1)
\]

\[
P_L(D_h) \approx P_{Lc}(x_{L1}|x_h) \times P_{Lc}(x_{L2}|x_{L1}, x_h) \times \ldots \times P_{Lc}(x_{Lk}|x_{L(k-1)}, \ldots, x_{L(k-N+1)}, x_h) \quad (2)
\]

\[
P_R(D_h) \approx P_{Rc}(x_{R1}|x_h) \times P_{Rc}(x_{R2}|x_{R1}, x_h) \times \ldots \times P_{Rc}(x_{Rk}|x_{R(k-1)}, \ldots, x_{R(k-N+1)}, x_h)
\]
Dependency Language Model

\[ P(y) = P(D_{ate}) \times P(D_{meat}) \times P(D_{with}) \times P(D_{fork}) \]

\[ P(D_{ate}) = P_L(D_{ate}) \times P_R(D_{ate}) \]

\[ P_L(D_{ate}) = P_{Lc}(he|ate) \]

\[ P_R(D_{ate}) = P_{Rc}(with|ate) \times P_{Rc}(with|meat, ate) \]
Parsing with DLM

- Graph-based parsing model
- Add DLM scores
- DLM-based feature templates
Graph-based model

- Find a maximum spanning tree (MST) (Mcdonald et al., 2005)

\[
y^* = \arg \max_{y \in T(G_x)} \max_{y \in T(G_x)} \sum_{g \in y} \text{score}(w, x, g)
\]

\[
\text{score}(w, x, g) = f(x, g) \cdot w
\]

\(f(x, g)\) is a high-dimensional feature representation which is based on arbitrary features of \(g\) and \(x\).

\(W\) is a weight vector.
Add DLM scores

- Consider the scores of DLM when searching for the MST

\[
y^*_{DLM} = \arg \max \left( \sum_{y \in T(G_x)} \sum_{g \in y} \text{score}(w, x, g) + \text{score}^{DLM}(y) \right)
\]

\[
y^* = \arg \max \text{s}(x, y) = \arg \max \sum_{y \in T(G_x)} \sum_{g \in y} \text{score}(w, x, g)
\]

\[
\text{score}^{DLM}(y) = f^{DLM} \cdot w^{DLM}
\]

\[
P(y) = \prod_{x_h \in H(y)} P(D_h)
\]
DLM-based feature templates

• Define DLM-based features for structure $D_h$.
  • $P_U (ch)$: probability of generating left/right child $ch$
  • TYPE: left/right

$$\Phi(P_u(ch)) = \begin{cases} PH & \text{if } N_o(P_u(ch)) \leq \text{TOP10} \\
PM & \text{if } \text{TOP10} < N_o(P_u(ch)) \leq \text{TOP30} \\
PL & \text{if } \text{TOP30} < N_o(P_u(ch)) \\
PO & \text{if } P_u(ch) = 0 \end{cases}$$

Table 1: DLM-based feature templates
Results from Chen et al. 2012

- On PTB

<table>
<thead>
<tr>
<th>Order1</th>
<th>UAS</th>
<th>Order2</th>
<th>UAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MST1</td>
<td>90.95</td>
<td>MST2</td>
<td>91.71</td>
</tr>
<tr>
<td>MST-DLM1</td>
<td>91.89</td>
<td>MST-DLM2</td>
<td>92.34</td>
</tr>
<tr>
<td>MSTB1</td>
<td>91.92</td>
<td>MSTB2</td>
<td>92.10</td>
</tr>
<tr>
<td>MSTB-DLM1</td>
<td>92.55</td>
<td>MSTB-DLM2</td>
<td>92.76</td>
</tr>
</tbody>
</table>

Table 5: Main results for English
- Partial tree level

- Other approaches
- Partial tree level

- Hybrid discriminative and generative models (Suzuki et al., 2009)
  - No explicit parses are produced for the unlabeled data.
  - Instead, derive useful estimations from the unlabeled data using many generative models, and integrate them in a discriminative model.
    - Many generative models
      - Each estimates the possibility of a given link from the view of one single feature (e.g., the word-bigram feature)
      - Trained with the unlabeled data.
    - A discriminative model (CRF)
      - Combine the basic features and the estimations of the generative models.
      - Trained with the labeled data.
- Suzuki et al., 2009

- Hybrid discriminative and generative models (Suzuki et al., 2009)

- Iteratively Training
  - Step 1: train the discriminative model with the labeled data adopting uniform distributions for the parameters of the generative models.
  - Step 2: apply the EM algorithm to train the generative models where the distributions are determined by the above discriminative model.
  - Step 3: retrain the discriminative model adopting the above generative models. Go to step 2 if desired.
Results from Suzuki et al., 2009

(a) English dependency parsers on PTB

<table>
<thead>
<tr>
<th>dependency parser</th>
<th>test</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>(McDonald et al., 2005a)</td>
<td>90.9</td>
<td>1od</td>
</tr>
<tr>
<td>(McDonald and Pereira, 2006)</td>
<td>91.5</td>
<td>2od</td>
</tr>
<tr>
<td>(Koo et al., 2008)</td>
<td>92.23</td>
<td>1od, 43M ULD</td>
</tr>
<tr>
<td>SS-SCM (w/ CL)</td>
<td>92.70</td>
<td>1od, 3.72G ULD</td>
</tr>
<tr>
<td>(Koo et al., 2008)</td>
<td>93.16</td>
<td>2od, 43M ULD</td>
</tr>
<tr>
<td>2-stage SS-SCM(+MIRA, w/ CL)</td>
<td>93.79</td>
<td>2od, 3.72G ULD</td>
</tr>
</tbody>
</table>
- Partial tree level

- Meta features (Chen et. al, 2013)
  - A base feature represents a kind of partial tree structure
  - The base features may suffer from the data sparseness problem
  - The target is to build connections among base features
Transformation function

- Key
  - Transformation function: $m_{fb} = \Phi(f_b)$
Transformation Functions

- Options
  - PCA-based algorithms
  - Clustering algorithms
  - Rule-based approaches
  - ...

Feature clustering  OK
Transformation Function

- A simple mapping function (Chen et al. 2013)
  - Generate base features from the trees in the auto-parsed data
  - Collect the features and count their frequencies
  - Sort the collected features in decreasing order
  - $R(f_b)$ is the position number of $f_b$ in the list

$$
\Phi(f_b) = \begin{cases} 
H_i & \text{if } R(f_b) \leq \text{TOP10} \\
M_i & \text{if } \text{TOP10} < R(f_b) \leq \text{TOP30} \\
L_i & \text{if } \text{TOP30} < R(f_b) \\
O_i & \text{Others}
\end{cases}
$$
Meta-features

• Generating meta features

I ate the meat with a fork.

\[ T_k: h_w, d_w, c_w, d(h, d, c) \]

\[ F_b: \text{ate, meat, with, RIGHTSIB} \]

\[ \Phi(f_b) = M_k \]

\[ [M_k]; [M_k], VV; [M_k], \text{ate} \]
Results from Chen et al., 2013

- On PTB

<table>
<thead>
<tr>
<th></th>
<th>UAS</th>
<th>COMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>92.76</td>
<td>48.05</td>
</tr>
<tr>
<td>MetaParser</td>
<td>93.77</td>
<td>51.36</td>
</tr>
</tbody>
</table>

Table 7: Main results on English
Semi-supervised dependency parsing

- Whole tree level
- Partial tree level
- Lexical level
Lexical level

• Learn information from words
  • Word clusters as extra features (Koo et al., 2008)

Figure from (Koo et al., 2008)

• Lexical co-occurrences counts from the web
  • PMI (Zhou et al., 2011)
  • Web count (Bansal and Klein, 2011)
Lexical level

• Learn information from words
  • Word clusters as extra features (Koo et al., 2008)

![Diagram showing word clusters](image)

Figure from (Koo et al., 2008)

• Each word can be represented as a bit string
• By using prefixes of various lengths, we can produce different clusters
Word clusters

- Different types of word clusters
  - c4: 4 bit-string prefix
  - c6: 6 bit-string prefix
  - c*: full bit-string. 1,000 distinct bit-strings
Word clusters

- **Examples**
  - ht, mt: head POS + modifier POS
  - hc4, mc4: 4 bit prefix of head + 4 bit prefix of modifier

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Cluster-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>ht, mt</td>
<td>hc4, mc4</td>
</tr>
<tr>
<td>hw, mw</td>
<td>hc6, mc6</td>
</tr>
<tr>
<td>hw, ht, mt</td>
<td>hc*, mc*</td>
</tr>
<tr>
<td>hw, ht, mw</td>
<td>hc4, mt</td>
</tr>
<tr>
<td>ht, mw, mt</td>
<td>ht, mc4</td>
</tr>
<tr>
<td>hw, mw, mt</td>
<td>hc6, mt</td>
</tr>
<tr>
<td>hw, ht, mw, mt</td>
<td>ht, mc6</td>
</tr>
<tr>
<td>...</td>
<td>hc4, mw</td>
</tr>
<tr>
<td>...</td>
<td>hw, mc4</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>ht, mt, st</td>
<td>hc4, mc4, sc4</td>
</tr>
<tr>
<td>ht, mt, gt</td>
<td>hc6, mc6, sc6</td>
</tr>
<tr>
<td>...</td>
<td>ht, mc4, sc4</td>
</tr>
<tr>
<td></td>
<td>hc4, mc4, gc4</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
Results from Koo et al. 2008

- On PTB

<table>
<thead>
<tr>
<th>Sec</th>
<th>dep1</th>
<th>dep1c (+1.09)</th>
<th>dep2</th>
<th>dep2c (+1.01)</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>90.48</td>
<td>91.57</td>
<td>91.76</td>
<td>92.77</td>
</tr>
<tr>
<td>01</td>
<td>91.31</td>
<td>92.43 (+1.12)</td>
<td>92.46</td>
<td>93.34 (+0.88)</td>
</tr>
<tr>
<td>23</td>
<td>90.84</td>
<td>92.23 (+1.39)</td>
<td>92.02</td>
<td>93.16 (+1.14)</td>
</tr>
<tr>
<td>24</td>
<td>89.67</td>
<td>91.30 (+1.63)</td>
<td>90.92</td>
<td>91.85 (+0.93)</td>
</tr>
</tbody>
</table>
Lexical Level

- Word co-occurrences (Zhou et al. 2011)
  - Web-scale resources
    - N-gram counts by search engine google.
    - N-gram counts by google Web 1T 5-gram corpus
  - Association score between two words

\[
\text{PMI}(x, y) = \log \frac{p(“x y”)}{p(“x”)p(“y”)}
\]
## Word co-occurrences

<table>
<thead>
<tr>
<th>N-gram feature templates</th>
</tr>
</thead>
<tbody>
<tr>
<td>hw, mw, PMI(hw,mw)</td>
</tr>
<tr>
<td>hw, ht, mw, PMI(hw,mw)</td>
</tr>
<tr>
<td>hw, mw, mt, PMI(hw,mw)</td>
</tr>
<tr>
<td>hw, ht, mw, mt, PMI(hw,mw)</td>
</tr>
<tr>
<td>...</td>
</tr>
<tr>
<td>hw, mw, sw</td>
</tr>
<tr>
<td>hw, mw, sw, PMI(hw, mw, sw)</td>
</tr>
<tr>
<td>hw, mw, gw</td>
</tr>
<tr>
<td>hw, mw, gw, PMI(hw, mw, gw)</td>
</tr>
</tbody>
</table>
Results from Zhou et al. 2011

- On PTB

<table>
<thead>
<tr>
<th>Sec</th>
<th>dep1</th>
<th>+hits</th>
<th>+V1</th>
<th>dep2</th>
<th>+hits</th>
<th>+V1</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>90.39</td>
<td>90.94</td>
<td>90.91</td>
<td>91.56</td>
<td>92.16</td>
<td>92.16</td>
</tr>
<tr>
<td>01</td>
<td>91.01</td>
<td>91.60</td>
<td>91.60</td>
<td>92.27</td>
<td>92.89</td>
<td>92.86</td>
</tr>
<tr>
<td>23</td>
<td>90.82</td>
<td>91.46</td>
<td>91.39</td>
<td>91.98</td>
<td>92.64</td>
<td>92.59</td>
</tr>
<tr>
<td>24</td>
<td>89.53</td>
<td>90.15</td>
<td>90.13</td>
<td>90.81</td>
<td>91.44</td>
<td>91.41</td>
</tr>
</tbody>
</table>
Semi-supervised dependency parsing (In-domain)

- Three levels
  - Lexical level
  - Partial tree level
  - Whole tree level
End of Part B
References


References

References


