Homework I – Languages

- Ukrainian: 7
- Russian: 3
- Estonian: 2
- German: 2
- Georgian: 1
- Azerbaijani: 1
- Urdu: 1
Homework I - Results

- Average points: 9.35
- Minimum points: 8
- Maximum points: 10

- 10 points: everything is done and it is easy to get an overview what and how was done
- 9 points: There are some minor problems with the results and/or the report
- 8 points: There are some minor problems with the results and/or it was somewhat difficult to follow the report
• **Morphology** – internal structure of words
• **Syntax** – internal structure of sentences

http://pixelmonkey.org/pub/nlp-training/img/02_parsetree_white.png
Syntactic ambiguity
The chicken is ready to eat.

https://cdn.drawception.com/images/panels/2012/3-28/52RW5TaRqe-6.png
“One morning I shot an elephant in my pajamas.

How he got in my pajamas, I don’t know.”

Groucho Marx
More ambiguous sentences

• I saw the man with binoculars.
• Look at the dog with one eye.
• I watched her duck.
• The peasants are revolting.
• They are cooking apples.
• Stolen painting found by tree.
• Police help dog bite victim.
Syntactic analysis/parsing

• Shallow parsing
• Phrase structure / constituency parsing
• Dependency parsing
The role of syntax in NLP

- Text generation/summarization/machine translation
- Useful features for various information extraction tasks
- Syntactic structure also reflects the semantic relations between the words
Shallow Parsing
Shallow parsing

• Also called *chunking* or *light parsing*
• Split the sentence into non-overlapping syntactic phrases

The morning flight from Denver has arrived.

NP       PP       NP       VP
NP – Noun phrase
PP – Prepositional Phrase
VP – Verb phrase
BIO tagging

• A labelling scheme often used in information extraction problems, treated as a sequence tagging task

The **morning flight from** Denver **has arrived.**

- B_NP – Beginning of a noun phrase
- I_NP – Inside a noun phrase
- B_VB – Beginning of a verb phrase etc
BIO tagging

• With only noun phrases

The morning flight from Denver has arrived.

B_NP I_NP I_NP O B_NP O O

B_NP – Beginning of a noun phrase
I_NP – Inside a noun phrase
O – Outside of a noun phrase
Sequence classifier

• Need annotated data for training: POS-tagged, phrase-annotated
• Use a sequence classifier of your choice

Figure 12.8: https://web.stanford.edu/~jurafsky/slp3/12.pdf
Evaluation: precision and recall

Precision = \frac{TP}{TP + FP}

Recall = \frac{TP}{TP + FN}

How many selected items are relevant?

How many relevant items are selected?

https://en.wikipedia.org/wiki/Precision_and_recall
Constituency Parsing
Constituency parsing

• Full constituency parsing helps to resolve structural ambiguities

Figure 12.2: https://web.stanford.edu/~jurafsky/slp3/12.pdf
Structural ambiguities

- **Attachment ambiguity** – a constituent/phrase can be attached to different places in the tree (the elephant example)

- **Coordination ambiguity**
  - [old [men and women]]
  - Both men and women are old
  - [old men] and [women]
  - Only men are old
Bracketed style

- The trees can be represented linearly with brackets

(S (Pr I)
   (Aux will)
   (VP (V do)
      (NP (Det my)
         (N homework))_NP
   )_VP
 )_S
Context-free grammars

\[
S \rightarrow NP \ VP \\
VP \rightarrow V \ NP \\
VP \rightarrow V \ NP \ PP \\
NP \rightarrow NP \ NP \\
NP \rightarrow NP \ PP \\
NP \rightarrow N \\
NP \rightarrow e \\
PP \rightarrow P \ NP
\]

\[
N \rightarrow \text{people} \\
N \rightarrow \text{fish} \\
N \rightarrow \text{tanks} \\
N \rightarrow \text{rods} \\
V \rightarrow \text{people} \\
V \rightarrow \text{fish} \\
V \rightarrow \text{tanks} \\
P \rightarrow \text{with}
\]

- \( G = (T, N, S, R) \)
  - \( T \) is a set of terminal symbols
  - \( N \) is a set of nonterminal symbols
  - \( S \) is the start symbol \( (S \in N) \)
  - \( R \) is a set of rules/productions of the form \( X \rightarrow \gamma \)
    - \( X \in N \) and \( \gamma \in (N \cup T)^* \)
- A grammar \( G \) generates a language \( L \).

\textit{people fish tanks} \\
\textit{people fish with rods}
Probabilistic CFGs

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

- A grammar $G$ generates a language model $L$. 

http://slideplayer.com/slide/4559350/
A PCFG

<table>
<thead>
<tr>
<th>Rule</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>S → NP VP</td>
<td>1.0</td>
</tr>
<tr>
<td>VP → V NP</td>
<td>0.6</td>
</tr>
<tr>
<td>VP → V NP PP</td>
<td>0.4</td>
</tr>
<tr>
<td>NP → NP NP</td>
<td>0.1</td>
</tr>
<tr>
<td>NP → NP PP</td>
<td>0.2</td>
</tr>
<tr>
<td>NP → N</td>
<td>0.7</td>
</tr>
<tr>
<td>PP → P NP</td>
<td>1.0</td>
</tr>
<tr>
<td>N → people</td>
<td>0.5</td>
</tr>
<tr>
<td>N → fish</td>
<td>0.2</td>
</tr>
<tr>
<td>N → tanks</td>
<td>0.2</td>
</tr>
<tr>
<td>N → rods</td>
<td>0.1</td>
</tr>
<tr>
<td>V → people</td>
<td>0.1</td>
</tr>
<tr>
<td>V → fish</td>
<td>0.6</td>
</tr>
<tr>
<td>V → tanks</td>
<td>0.3</td>
</tr>
<tr>
<td>P → with</td>
<td>1.0</td>
</tr>
</tbody>
</table>

http://slideplayer.com/slide/4559350/
The probability of strings and trees

- \( P(t) \) – The probability of a tree \( t \) is the product of the probabilities of the rules used to generate it.
- \( P(s) \) – The probability of the string \( s \) is the sum of the probabilities of the trees which have that string as their yield

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

\[
= \sum_j P(t)
\]
Exercise

• Compute the probability of a tree for

People fish tanks with rods
PCFG for efficient parsing

• For efficient parsing the rules should be **unary** or **binary**

• Chomsky normal form – all rules have the form:
  • X --> Y Z
  • X --> w
  • X, Y, Z - non-terminal symbols
  • w – terminal symbol
  • No epsilon rules
Before binarization

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

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

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

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

http://slideplayer.com/slide/4559350/
Before and after binarization

http://slideplayer.com/slide/4559350/
Finding the most likely tree: CKY parsing

- Dynamic programming algorithm
- Proceeds bottom-up and performs Viterbi on trees
CKY parsing

• For a full example look at the slides at http://slideplayer.com/slide/4559350/
CKY parsing

S → NP VP 0.9
S → VP 0.1
VP → V NP 0.5
VP → V 0.1
VP → V @VP_V 0.3
VP → V PP 0.1
@VP_V → NP PP 1.0
NP → NP NP 0.1
NP → NP PP 0.2
NP → N 0.7
PP → P NP 1.0
N → people 0.5
N → fish 0.2
N → tanks 0.2
N → rods 0.1
V → people 0.1
V → fish 0.6
V → tanks 0.3
P → with 1.0

http://slideplayer.com/slide/4559350/
CKY parsing

S → NP VP  0.9
S → VP  0.1
VP → V NP  0.5
VP → V  0.1
VP → V @VP_V  0.3
VP → V PP  0.1
@VP_V → NP PP  1.0
NP → NP NP  0.1
NP → NP PP  0.2
NP → N  0.7
PP → P NP  1.0
N → people  0.5
N → fish  0.2
N → tanks  0.2
N → rods  0.1
V → people  0.1
V → fish  0.6
V → tanks  0.3
P → with  1.0

http://slideplayer.com/slide/4559350/
Evaluating constituency parsing

Gold standard brackets:
S-(0:11), NP-(0:2), VP-(2:9), VP-(3:9), NP-(4:6), PP-(6-9), NP-(7,9), NP-(9:10)

Candidate brackets:
S-(0:11), NP-(0:2), VP-(2:10), VP-(3:10), NP-(4:6), PP-(6-10), NP-(7,10)

<table>
<thead>
<tr>
<th>Measure</th>
<th>Calculation</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Labeled Precision</td>
<td>3/7</td>
<td>42.9%</td>
</tr>
<tr>
<td>Labeled Recall</td>
<td>3/8</td>
<td>37.5%</td>
</tr>
<tr>
<td>LP/LR F1</td>
<td></td>
<td>40.0%</td>
</tr>
</tbody>
</table>

F1 = \frac{2 \cdot \text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}
Dependency parsing

- Dependency parse is a directed graph $G = (V, A)$
  - $V$ – the set of vertices corresponding to words
  - $A$ – the set of nodes corresponding to dependency relations

Visualization with http://corenlp.run/
Dependency parsing

- More compact grammar formalism than CFG

Figure 14.1: https://web.stanford.edu/~jurafsky/slp3/14.pdf
Dependency relations

- The arrows connect heads and their dependents.
- The main verb is the head or the root of the whole sentence.
- The arrows are labelled with grammatical functions/dependency relations.
Properties of a dependency graph

A dependency tree is a directed graph that satisfies the following constraints:

1. There is a single designated root node that has no incoming arcs
   • Typically the main verb of the sentence
2. With the exception of the root node, each node has exactly one incoming arc
   • Each dependent has a single head
3. There is a unique path from the root node to each vertex in V
   • The graph is acyclic and connected
Projectivity

• Projective trees – there are no arc crossings in the dependency graphs
• Non-projective trees - crossings due to free word order
## Dependency relations

<table>
<thead>
<tr>
<th>Clausal Argument Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nsubj</td>
<td>Nominal subject</td>
</tr>
<tr>
<td>DOBJ</td>
<td>Direct object</td>
</tr>
<tr>
<td>IOBJ</td>
<td>Indirect object</td>
</tr>
<tr>
<td>Ccomp</td>
<td>Clausal complement</td>
</tr>
<tr>
<td>Xcomp</td>
<td>Open clausal complement</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Nominal Modifier Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nmod</td>
<td>Nominal modifier</td>
</tr>
<tr>
<td>Amod</td>
<td>Adjectival modifier</td>
</tr>
<tr>
<td>Nummod</td>
<td>Numeric modifier</td>
</tr>
<tr>
<td>Apos</td>
<td>Appositional modifier</td>
</tr>
<tr>
<td>Det</td>
<td>Determiner</td>
</tr>
<tr>
<td>Case</td>
<td>Prepositions, postpositions and other case markers</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Notable Relations</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conj</td>
<td>Conjunct</td>
</tr>
<tr>
<td>CC</td>
<td>Coordinating conjunction</td>
</tr>
</tbody>
</table>

Figure 14.2: https://web.stanford.edu/~jurafsky/slp3/14.pdf
Universal dependencies

- [http://universaldependencies.org/](http://universaldependencies.org/)
- Annotated treebanks in many languages
- Uniform annotation scheme across all languages:
  - Universal POS tags
  - Universal dependency relations
Dependency parsing methods

• Transition-based parsing
  • stack-based algorithms/shift-reduce parsing
  • only generate projective trees

• Graph-based algorithms
  • can also generate non-projective trees
Transition-based parsing

• Three main components:
  • Stack
  • Buffer
  • Set of dependency relations

• A configuration is the current state of the stack, buffer and the relation set

Figure 14.5: https://web.stanford.edu/~jurafsky/slp3/14.pdf
Arc-standard parsing system

• Initial configuration: \( \sigma = [\text{ROOT}], \beta = [w_1, \ldots, w_n], A = \emptyset \)
  • Stack contains the ROOT symbol
  • Buffer contains all words in the sentence
  • Dependency relation set is empty

• At each step perform either:
  • Shift – move a word from the buffer to the stack:
    \[ \sigma, w_i | \beta, A \rightarrow \sigma | w_i, \beta, A \]
  • LeftArc – left arc between top two words in the stack, pop the second word:
    \[ \sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_j, \beta, A \cup \{w_j, w_i\} \]
  • RightArc – right arc between top two words in the stack, pop the first word:
    \[ \sigma | w_i | w_j, \beta, A \rightarrow \sigma | w_i, \beta, A \cup \{w_i, w_j\} \]
Oracle

• The annotated data is in the form of a treebank
  • Each sentence is annotated with its dependency tree

• The task of the transition-based parser is to predict the correct parsing operation at each step:
  • Input is configuration
  • Output is parsing action: Shift, RightArc or LeftArc

• The role of the oracle is to return the correct parsing operation for each configuration in the training set
Oracle

• Choose LeftArc if it produces a correct head-dependent relation given the reference parse and the current configuration

• Choose RightArc if:
  • It produces a correct head-dependent relation given the reference parse and the current configuration
  • All of the dependents of the word at the top of the stack have already been assigned

• Otherwise choose Shift
Example

\[
\begin{align*}
\text{Shift:} & \quad \sigma, w_i|\beta, A \rightarrow \sigma|w_i, \beta, A \\
\text{LeftArc:} & \quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_j, \beta, A \cup \{w_j, w_i\} \\
\text{RightArc:} & \quad \sigma|w_i|w_j, \beta, A \rightarrow \sigma|w_i, \beta, A \cup \{w_i, w_j\}
\end{align*}
\]
<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
<th>Arc</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Example

<table>
<thead>
<tr>
<th>Stack</th>
<th>Buffer</th>
<th>Action</th>
<th>Arc</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ROOT]</td>
<td>[The, cat, sat, on, the, mat]</td>
<td>Shift</td>
<td></td>
</tr>
<tr>
<td>[ROOT, The]</td>
<td>[cat, sat, on, the, mat]</td>
<td>Shift</td>
<td></td>
</tr>
<tr>
<td>[ROOT, The, cat]</td>
<td>[sat, on, the, mat]</td>
<td>Left-Arc</td>
<td>det(The &lt;-- cat)</td>
</tr>
<tr>
<td>[ROOT, cat]</td>
<td>[sat, on, the, mat]</td>
<td>Shift</td>
<td></td>
</tr>
<tr>
<td>[ROOT, cat, sat]</td>
<td>[on, the, mat]</td>
<td>Left-Arc</td>
<td>nsubj(cat &lt;-- sat)</td>
</tr>
<tr>
<td>[ROOT, sat]</td>
<td>[on, the, mat]</td>
<td>Shift</td>
<td></td>
</tr>
<tr>
<td>[ROOT, sat, on]</td>
<td>[the, mat]</td>
<td>Shift</td>
<td></td>
</tr>
<tr>
<td>[ROOT, sat, on, the]</td>
<td>[mat]</td>
<td>Shift</td>
<td></td>
</tr>
<tr>
<td>[ROOT, sat, on, the, mat]</td>
<td>[]</td>
<td>Left-Arc</td>
<td>det(the &lt;-- mat)</td>
</tr>
<tr>
<td>[ROOT, sat, on, mat]</td>
<td>[]</td>
<td>Left-Arc</td>
<td>case(on &lt;-- mat)</td>
</tr>
<tr>
<td>[ROOT, sat, mat]</td>
<td>[]</td>
<td>Right-Arc</td>
<td>nmod(sat --&gt; mat)</td>
</tr>
<tr>
<td>[ROOT, sat]</td>
<td>[]</td>
<td>Right-Arc</td>
<td>root(ROOT, sat)</td>
</tr>
<tr>
<td>[ROOT]</td>
<td>[]</td>
<td>Done</td>
<td></td>
</tr>
</tbody>
</table>
Typical features

• First word from the stack
• second word from the stack
• The POS of the first word in the stack
• The POS of the second word in the stack
• The first word in the buffer
• The POS of the first word in the buffer
• The word and the POS of the top word in the stack
• …
Exercise

• The next action from the current configuration is Shift. Construct the features.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Word buffer</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[root, canceled, flights]</td>
<td>[to Houston]</td>
<td>(canceled → United)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights → morning)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights → the)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Template</th>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>First word from the stack</td>
<td></td>
</tr>
<tr>
<td>Second word from the stack</td>
<td></td>
</tr>
<tr>
<td>POS of first stack word</td>
<td></td>
</tr>
<tr>
<td>POS of second stack word</td>
<td></td>
</tr>
<tr>
<td>First word from the buffer</td>
<td></td>
</tr>
<tr>
<td>POS of the first buffer word</td>
<td></td>
</tr>
<tr>
<td>Word and POS of the top stack word</td>
<td></td>
</tr>
</tbody>
</table>
Exercise

• The next action from the current configuration is Shift. Construct the features.

<table>
<thead>
<tr>
<th>Stack</th>
<th>Word buffer</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>[root, canceled, flights]</td>
<td>[to Houston]</td>
<td>(canceled → United)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights → morning)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(flights → the)</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
\langle s_1.w = \text{flights}, op = \text{shift} \rangle \\
\langle s_2.w = \text{canceled}, op = \text{shift} \rangle \\
\langle s_1.t = \text{NNS}, op = \text{shift} \rangle \\
\langle s_2.t = \text{VBD}, op = \text{shift} \rangle \\
\langle b_1.w = \text{to}, op = \text{shift} \rangle \\
\langle b_1.t = \text{TO}, op = \text{shift} \rangle \\
\langle s_1.wt = \text{flightsNNS}, op = \text{shift} \rangle
\end{align*}
\]
### Standard feature templates

<table>
<thead>
<tr>
<th>Source</th>
<th>Feature templates</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>One word</strong></td>
<td></td>
</tr>
<tr>
<td>$s_1.w$</td>
<td>$s_1.t$</td>
</tr>
<tr>
<td>$s_2.w$</td>
<td>$s_2.t$</td>
</tr>
<tr>
<td>$b_1.w$</td>
<td>$b_1.w$</td>
</tr>
<tr>
<td><strong>Two word</strong></td>
<td></td>
</tr>
<tr>
<td>$s_1.w \circ s_2.w$</td>
<td>$s_1.t \circ s_2.t$</td>
</tr>
<tr>
<td>$s_1.t \circ s_2.wt$</td>
<td>$s_1.w \circ s_2.w \circ s_2.t$</td>
</tr>
<tr>
<td>$s_1.w \circ s_1.t \circ s_2.t$</td>
<td>$s_1.w \circ s_1.t$</td>
</tr>
</tbody>
</table>

Figure 14.9: https://web.stanford.edu/~jurafsky/slp3/14.pdf
Evaluation

Unlabelled attachment score:
• The proportion of correct head attachments

Labelled attachment score:
• The proportion of correct head attachments labelled with the correct relation

Label accuracy
• The proportion of correct incoming relation labels ignoring the head
Evaluation

UAS =
LAS =
LA =

Figure 14.15: https://web.stanford.edu/~jurafsky/slp3/14.pdf
Evaluation

UAS = 5/6
LAS = 4/6
LA = 4/6

Figure 14.15: https://web.stanford.edu/~jurafsky/slp3/14.pdf
## SyntaxNet

- [https://github.com/tensorflow/models/blob/master/research/syntaxnet/g3doc/universal.md](https://github.com/tensorflow/models/blob/master/research/syntaxnet/g3doc/universal.md)

<table>
<thead>
<tr>
<th>Language</th>
<th>Tokens</th>
<th>UAS</th>
<th>LAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>25096</td>
<td>84.89%</td>
<td>80.38%</td>
</tr>
<tr>
<td>Estonian</td>
<td>23670</td>
<td>83.10%</td>
<td>78.83%</td>
</tr>
<tr>
<td>Finnish</td>
<td>9140</td>
<td>83.65%</td>
<td>79.60%</td>
</tr>
<tr>
<td>German</td>
<td>16268</td>
<td>79.73%</td>
<td>74.07%</td>
</tr>
<tr>
<td>Kazakh</td>
<td>587</td>
<td>58.09%</td>
<td>43.95%</td>
</tr>
<tr>
<td>Chinese</td>
<td>12012</td>
<td>76.71%</td>
<td>71.24%</td>
</tr>
<tr>
<td>Latvian</td>
<td>3985</td>
<td>58.92%</td>
<td>51.47%</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>81.12%</strong></td>
<td><strong>75.85%</strong></td>
<td></td>
</tr>
</tbody>
</table>
Neural Dependency parsers

• Kipperwasser and Goldberg, 2016. **Simple and Accurate Dependency Parsing Using Bidirectional LSTM Feature Representations**
Neural Dependency parsers

• Dyer et al., 2015. Transition-based Dependency Parsing with Stack Long Short-Term Memory
Parsing resources

• Stanford constituency and dependency parser for English: https://nlp.stanford.edu/software/lex-parser.shtml

• Spacy parser for English and German: https://spacy.io/

• MaltParser for morphologically complex languages: http://www.maltparser.org/
Parsing Estonian

• Estnltk has two parsers:
  • A trained MaltParser model
  • A rule-based parser based on Constraint Grammar


<table>
<thead>
<tr>
<th></th>
<th>LAS</th>
<th>UAS</th>
<th>LA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stanford</td>
<td>76.3</td>
<td>80.4</td>
<td>87.6</td>
</tr>
<tr>
<td>spaCy</td>
<td>76.6</td>
<td>82.2</td>
<td>85.5</td>
</tr>
<tr>
<td>SyntaxNet</td>
<td>78.3</td>
<td>83.4</td>
<td>87.1</td>
</tr>
<tr>
<td>MaltParser</td>
<td>80.0</td>
<td>83.6</td>
<td>89.2</td>
</tr>
<tr>
<td>UDPipe</td>
<td>79.1</td>
<td>82.5</td>
<td>90.1</td>
</tr>
</tbody>
</table>
Recap

• Parsing is the task of finding syntactic structure of sentences

• Shallow parsing – find only non-overlapping syntactic phrases
  • Simpler task than full syntactic parsing
  • Useful for information extraction tasks, i.e named entities can only occur in noun phrases

• Constituency parsing – full syntactic analysis that breaks the text into phrases and sub-phrases

• Dependency parsing – simpler grammar formalism that marks the syntactic dependence relation between words
  • More suitable for languages with free word order