Parsing and Machine learning
http://www-rohan.sdsu.edu/~gawron/aisem

Transition-based parsing

Jean Mark Gawron

San Diego State University, Department of Linguistics

2010-08-19

Overview

Dependency parsing

Dependency parsing

Given a string in some language, find the best dependency graph between the words. (Kübler et al. 2009)

- Assumption: "Best" corresponds to a score assigned to each dependency trees by a dependency model.

Dependency parsing

Given a string in some language, find the best dependency graph between the words. (Kübler et al. 2009)

- Assumption: "Best" corresponds to a score assigned to each dependency trees by a dependency model.
- Two basic types of dependency model

Dependency parsing

Given a string in some language, find the best dependency graph between the words. (Kübler et al. 2009)

- Assumption: "Best" corresponds to a score assigned to each dependency trees by a dependency model.
- Two basic types of dependency model
  - Transition-based models: parameterized over parser transitions (in a sense very like that of an LR parser or a shift-reduce parser).

Dependency parsing

Given a string in some language, find the best dependency graph between the words. (Kübler et al. 2009)

- Assumption: "Best" corresponds to a score assigned to each dependency trees by a dependency model.
- Two basic types of dependency model
  - Transition-based models: parameterized over parser transitions (in a sense very like that of an LR parser or a shift-reduce parser).
  - Graph-based models: parameterized over substructures of a dependency tree:

Dependency parsing

Given a string in some language, find the best dependency graph between the words. (Kübler et al. 2009)

- Assumption: "Best" corresponds to a score assigned to each dependency trees by a dependency model.
- Two basic types of dependency model
  - Transition-based models: parameterized over parser transitions (in a sense very like that of an LR parser or a shift-reduce parser).
  - Graph-based models: parameterized over substructures of a dependency tree:
    1. Arc-factored: model limited to predicting individual arcs in graph

Dependency parsing

Given a string in some language, find the best dependency graph between the words. (Kübler et al. 2009)

- Assumption: "Best" corresponds to a score assigned to each dependency trees by a dependency model.
- Two basic types of dependency model
  - Transition-based models: parameterized over parser transitions (in a sense very like that of an LR parser or a shift-reduce parser).
  - Graph-based models: parameterized over substructures of a dependency tree:
    1. Arc-factored: model limited to predicting individual arcs in graph
    2. Arc-factored +: model includes arity: How likely is a given word to have a fixed number of dependents?

Transition functions

A transition function maps from one parse configuration to another.

Transition functions

A transition function maps from one parse configuration to another.

A parse configuration has 3 components: Stack (σ), Buffer (β), Arcs (A)

- Left-Arc$_r$: Add a dependency to A top word of stack depends on top word in buffer. Pop stack.

$$(\sigma|w_i,\ w_j|\beta, \mathrm{A}) \Rightarrow (\sigma,\ w_j|\beta,\ \mathrm{A} \cup \{(w_j, r, w_i)\})$$

Transition functions

A transition function maps from one parse configuration to another.

A parse configuration has 3 components: Stack (σ), Buffer (β), Arcs (A)

- Left-Arc$_r$: Add a dependency to A top word of stack depends on top word in buffer. Pop stack.

$$(\sigma|w_i,\ w_j|\beta, \mathrm{A}) \Rightarrow (\sigma,\ w_j|\beta,\ \mathrm{A} \cup \{(w_j, r, w_i)\})$$

- Right-Arc$_r$: Add a dependency to A: top word in buffer depends on top word of stack. Replace top word in stack with top word in buffer.

$$(\sigma|w_i,\ w_j|\beta, \mathrm{A}) \Rightarrow (\sigma,\ w_i|\beta,\ \mathrm{A} \cup \{(w_i, r, w_j)\})$$

Transition functions

A transition function maps from one parse configuration to another.

A parse configuration has 3 components: Stack (σ), Buffer (β), Arcs (A)

- Left-Arc$_r$: Add a dependency to A top word of stack depends on top word in buffer. Pop stack.

$$(\sigma|w_i,\ w_j|\beta, \mathrm{A}) \Rightarrow (\sigma,\ w_j|\beta,\ \mathrm{A} \cup \{(w_j, r, w_i)\})$$

- Right-Arc$_r$: Add a dependency to A: top word in buffer depends on top word of stack. Replace top word in stack with top word in buffer.

$$(\sigma|w_i,\ w_j|\beta, \mathrm{A}) \Rightarrow (\sigma,\ w_i|\beta,\ \mathrm{A} \cup \{(w_i, r, w_j)\})$$

- Shift: Move an item from the buffer to the stack.

$$(\sigma,\ w_i|\beta, \mathrm{A}) \Rightarrow (\sigma|w_i,\ \beta,\ \mathrm{A})$$

Initial and terminal states

$$S = w_0 w_1 \ldots w_n$$

Assume w_0 is always a dummy word ROOT, which will be the root of all our dependency graphs:

$$\begin{array}{ll} \textit{Initial} & (\langle \textsc{root} \rangle_\sigma,\ [w_1,\ \ldots,\ w_n]_\beta,\ \emptyset) \\ \textit{Terminal} & (\sigma,\ [\,]_\beta,\ A) \end{array}$$

Algorithm

In each parsing configuration, call an ORACLE to find the correct transition function t (Left-Arc, Right-Arc, or Shift), until the parsing configuration is terminal:

Algorithm

In each parsing configuration, call an ORACLE to find the correct transition function t (Left-Arc, Right-Arc, or Shift), until the parsing configuration is terminal:

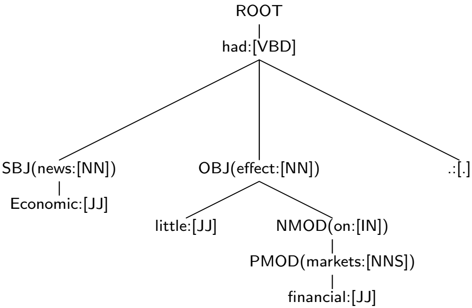
$\textsc{h}(S,\ \textsc{oracle})$

$(\sigma,\ \beta,\ \mathsf{A}\ ) \leftarrow \textsc{initial-configuration}(S)$
while $\textsc{not terminal}(\sigma,\ \beta,\ \mathsf{A}\ )$
do $t \leftarrow \textsc{oracle}(\sigma,\ \beta,\ \mathsf{A}\ )$
    $(\sigma,\ \beta,\ \mathsf{A}\ ) \leftarrow t(\sigma,\ \beta,\ \mathsf{A}\ )$
return A

Economic news had little effect on financial markets

```
init          => < ROOT > [ Economic ... . ]
shift         => < Economic ROOT > [ news ... . ]
left_a(NMOD)  => < ROOT > [ news ... . ]
shift         => < news ROOT > [ had ... . ]
left_a(SBJ)   => < ROOT > [ had ... . ]
shift         => < had ROOT > [ little ... . ]
shift         => < little had ROOT > [ effect ... . ]
left_a(NMOD)  => < had ROOT > [ effect ... . ]
shift         => < effect had ROOT > [ on ... . ]
shift         => < on ... ROOT > [ financial markets . ]
shift         => < financial ... ROOT > [ markets . ]
left_a(NMOD)  => < on ... ROOT > [ markets . ]
right_a(PMOD) => < effect had ROOT > [ on . ]
right_a(NMOD) => < had ROOT > [ effect . ]
right_a(OBJ)  => < ROOT > [ had . ]
shift         => < had ROOT > [ . ]
right_a(P)    => < ROOT > [ had ]
right_a(ROOT) => < > [ ROOT ]
shift         => < ROOT > [ ]
```

Parse



init	⟨ ROOT ⟩	[Economic. . .]
shift	⟨ Economic ROOT ⟩	[news. . .]
left_a	⟨ ROOT ⟩	[news. . .]
shift	⟨ news ROOT ⟩	[had. . .]
left_a	⟨ ROOT ⟩	[had. . .]
shift	⟨ had ROOT ⟩	[little . . .]
shift	⟨ little had ROOT ⟩	[effect . . .]
left_a	⟨ had ROOT ⟩	[effect . . .]
shift	⟨ effect had ROOT ⟩	[on. . .]
shift	⟨ on . . . ROOT ⟩	[financial. . .]
shift	⟨ financial . . . ROOT ⟩	[markets. . .]
left_a	⟨ on . . . ROOT ⟩	[markets. . .]
right_a	⟨ effect had ROOT ⟩	[on .]
right_a	⟨ had ROOT ⟩	[effect .]
right_a	⟨ ROOT ⟩	[had .]
shift	⟨ had ROOT ⟩	[.]
right_a	⟨ ROOT ⟩	[had]
right_a	⟨ ⟩	[ROOT]
shift	⟨ ROOT ⟩	[ ]

Economic news had little effect on financial markets .
Blue: non adjacent attachments

Observations

- Before finding a parent a word must find all its children (The child is vaporized after an attach operation.) In any chain of attachments, the most deeply embedded child must attach first.

shift	⟨ financial . . . ROOT ⟩	[markets. . .]
left_a	⟨ on . . . ROOT ⟩	[markets. . .]
right_a	⟨ effect had ROOT ⟩	[on .]

- RIGHT/LEFT ATTACH (RA/LA) is the only way of finding a parent to the left/right.
- The right attaching child is in the buffer; after a right attach, the right attaching parent goes back in the buffer:

ROOT had effect on markets

- After an attach operation, the parent is always in the buffer (immediately eligible for RA, or after a SHIFT, LA)

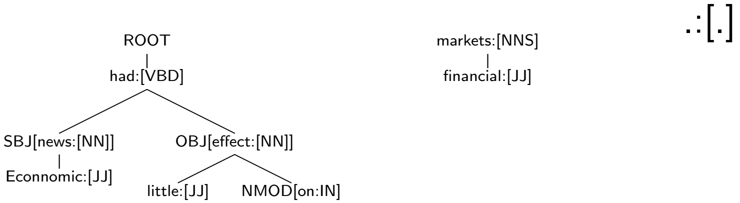
Arbitrary lookahead?

Right		
init	⟨ ROOT ⟩	[Economic. . .]
shift	⟨ Economic ROOT ⟩	[news. . .]
left_a	⟨ ROOT ⟩	[news. . .]
shift	⟨ news ROOT ⟩	[had. . .]
left_a	⟨ ROOT ⟩	[had. . .]
shift	⟨ had ROOT ⟩	[little . . .]
shift	⟨ little had ROOT ⟩	[effect . . .]
left_a	⟨ had ROOT ⟩	[effect . . .]
shift	⟨ effect had ROOT ⟩	[on. . .]
shift	⟨ on . . . ROOT ⟩	[financial. . .]
shift	⟨ financial . . . ROOT ⟩	[markets. . .]
left_a	⟨ on . . . ROOT ⟩	[markets. . .]
right_a	⟨ effect had ROOT ⟩	[on .]
right_a	⟨ had ROOT ⟩	[effect .]
right_a	⟨ ROOT ⟩	[had .]
shift	⟨ had ROOT ⟩	[.]
right_a	⟨ ROOT ⟩	[had]
right_a	⟨ ⟩	[ROOT]
shift	⟨ ROOT ⟩	[ ]

Wrong		
init	⟨ ROOT ⟩	[Economic. . .]
shift	⟨ Economic ROOT ⟩	[news. . .]
left_a	⟨ ROOT ⟩	[news. . .]
shift	⟨ news ROOT ⟩	[had. . .]
left_a	⟨ ROOT ⟩	[had. . .]
shift	⟨ had ROOT ⟩	[little . . .]
shift	⟨ little had ROOT ⟩	[effect . . .]
left_a	⟨ had ROOT ⟩	[effect . . .]
shift	⟨ effect had ROOT ⟩	[on. . .]
right_a	⟨ had ROOT ⟩	[effect. . .]
right_a	⟨ ROOT ⟩	[had . . .]
right_a	⟨ ⟩	[ROOT . . .]
shift	⟨ ROOT ⟩	[financial . . .]
shift	⟨ financial . . . ROOT ⟩	[markets. . .]
left_a	⟨ ROOT ⟩	[markets. . .]
shift	⟨ markets . . . ROOT ⟩	[.]
shift	⟨ . . . . ROOT ⟩	[ ]

Neither *markets* nor . has found a parent. We have a parse forest.

A parse forest



A bad oracle

```
init          => < ROOT > [ Economic ... . ] 1
shift         => < Economic ROOT > [ news ... . ] 1
shift         => < news Economic ROOT > [ had ... . ] 1
shift         => < had ... ROOT > [ little ... . ] 1
shift         => < little ... ROOT > [ effect ... . ] 1
shift         => < effect ... ROOT > [ on ... . ] 1
shift         => < on ... ROOT > [ financial markets . ] 1
shift         => < financial ... ROOT > [ markets . ] 1
shift         => < markets ... ROOT > [ . ] 1
shift         => < . ... ROOT > [ ]
```

Training Problem

Based on features of the current parse state, the oracle should guess what the best next parser action is.

What's in a state:

1. All the words in the Buffer (in some sequence)
2. All the words in the Stack (in some sequence)
3. All the dependency relations in Arcs

What to ouput (one of):

- Left-Arc$_r$
- Right-Arc$_r$
- Shift

A gap

Our desired training data:

```
init           => < ROOT > [ Economic ... . ]
shift          => < Economic ROOT > [ news ... . ]
left_a(NMOD)   => < ROOT > [ news ... . ]
```

What we've got:

```
(Start
  (S
    (NP (JJ Economic) (NN news))
    (VP
      (VBD had)
      (NP
        (NP (JJ little) (NN effect))
        (PP (IN on) (NP (JJ financial) (NNS markets)))))
    (cat_. .)))
```

A further gap

We want to train a feature-based model. We need to go from a real world event (a sequence of parser actions) to some vector of features that represents it:

```
init          => < ROOT > [ Economic ... . ]
shift         => < Economic ROOT > [ news ... . ]
left_a(NMOD)  => < ROOT > [ news ... . ]
```

(*Economic*, *news*, *had*, *NULL*, *NULL*, *NMOD*, *NULL*)

A final gap

We need a vector of numbers, so we need a binary encoding.

$$(\mathit{Economic}, \mathit{news}, \mathit{had}, \mathit{NULL}, \mathit{NULL}, \mathit{NMOD}, \mathit{NULL})$$

becomes

$$(0, 1, 1, 0, \ldots 1, 1)$$

Needed

1. Pennconverter: A map from WSJ sentences to dependency parses
2. Training oracle: A map from dependency parses to parser action sequences (forthcoming).
3. Feature model: A map from parser actions to feature vectors.

Pennconverter

```
(Start
  (S
    (NP (NNP Mr.) (NNP Vinken))
    (VP
      (VBZ is)
      (NP
        (NP (NN chairman))
        (PP
          (IN of)
          (NP
            (NP (NNP Elsevier) (NNP N.V.))
            (cat_, ,)
            (NP (DT the) (NNP Dutch) (VBG publishing) (NN group))))))
    (cat_. .)))
```

```
1       Mr.     _       NNP     _       _       2       TITLE   _       _
2       Vinken  _       NNP     _       _       3       SBJ     _       _
3       is      _       VBZ     _       _       0       ROOT    _       _
4       chairman        _       NN      _       _       3       PRD     _       _
5       of      _       IN      _       _       4       NMOD    _       _
6       Elsevier        _       NNP     _       _       5       PMOD    _       _
7       N.V.    _       NNP     _       _       6       POSTHON _       _
8       ,       _       ,       _       _       6       P       _       _
9       the     _       DT      _       _       12      NMOD    _       _
10      Dutch   _       NNP     _       _       12      NMOD    _       _
11      publishing      _       VBG     _       _       12      NMOD    _       _
12      group   _       NN      _       _       6       APPO    _       _
13      .       _       .       _       _       3       P       _       _
```

Training Oracle

Let A_d be the graph built by the gold standard dependency parse (a set of $(w_i,\ r,\ w_j)$ triples).

Parse using the algorithm above and the following oracle:

$$o(c = (\sigma, \beta, \mathrm{A})) = \begin{array}{ll} \text{Left-Arc}_r & \text{if } (\beta[0],\ r,\ \sigma[0]) \in \mathrm{A}_d; \\ \text{Right-Arc}_r & \text{if } (\sigma[0],\ r,\ \beta[0]) \in \mathrm{A}_d, \text{ and,} \\ & \text{for all } w, r', \\ & \quad \text{if } (\beta[0],\ r',\ w) \in \mathrm{A}_d, \\ & \quad \text{then } (\beta[0],\ r',\ w) \in \mathrm{A}; \\ \text{Shift} & \text{otherwise.} \end{array}$$

A feature model

Many feature models are possible. Here is a simple one that can do some work (Kübler et al. 2009:29). Let LDP(w) be the (last) *left dependency relation* of word w, RDP(w) be the (last) *right dependency relation* of w, and let STK[i] and BUF[i] be the ith member of the stack and buffer. WORD is type assigned to a word in the sentence and DEPREL is the type for a dependency relation label:

f	Address	Type
1	STK[0]	WORD
2	BUF[0]	WORD
3	BUF[1]	WORD
4	LDP(STK[0])	DEPREL
5	RDP(STK[0])	DEPREL
6	LDP(BUF[0])	DEPREL
7	RDP(BUF[0])	DEPREL

Example

Consider the LAST parse configuration of the following config sequence (the first 3 configurations in the correct parse of *Economic news had little effect on financial markets .*):

init	$\langle$ ROOT $\rangle$	[Economic. . .]	$\emptyset$
shift	$\langle$ Economic . . . $\rangle$	[news. . .]	$\emptyset$
$\mathrm{LA}_{\mathrm{NMOD}}$	$\langle$ ROOT $\rangle$	[news. . .]	{(news, NMOD, Economic)}

1 STK[0]	2 BUF[0]	3 BUF[1]	4 LDP(STK[0])	5 RDP(STK[0])	6 LDP(BUF[0])	7 RDP(BUF[0])
(ROOT,	news,	had,	NULL,	NULL,	NMOD,	NULL)

Thus feat 3 amounts to one-word of lookahead, feat 6 tell us whether the stack word already has a left dependent (it does), and the value NULL is used for feats 4, 5, and 7, because the required dependents do not exist in this parse configuration.

One last mapping

Vectors in vector models must have numbers as values. Assume a vocab-size of 7000. Assume 70 distinct dependency relations

	f	Address	# Dms	Value
	1	STK[0]	7000	Economic
	2	BUF[0]	7000	news
	3	BUF[1]	7000	had
	4	LDP(STK[0])	70	NULL
	5	RDP(STK[0])	70	NULL
	6	LDP(BUF[0])	70	NMOD
	7	RDP(BUF[0])	70	NULL
Total			21280	

$$f_1 \rightarrow \overbrace{\underbrace{(0,\ 0,\ \ldots,\ 1,\ \ldots,\ 0}_{\text{1 in } \textit{Economic} \text{ Dm}}}^{\text{7000 positions}},\ \underbrace{0,\ \ldots, 1, \ldots,}_{\text{1 in } \textit{news} \text{ Dm}}\ \ldots)$$

SVM Light format

Joachims (1999)

```
# Reuters category "corporate aquisitions" (test examples: 300 positive/ 300 neg
ative)
+1 6:0.0342598670723747 26:0.148286149621374 27:0.0570037235976456 31:0.03730864 \
82671729 33:0.0270832794680822 ... 2731:0.071437898589286 2771:0.0706069752753547 \
3553:0.0783933439550538 3589:0.0774668403369963
.
.
.
-1 17:0.21719967570196 91:0.318496281551414 111:0.249272990204113 \
242:0.279299199312461 280:0.417562480514829 \
327:0.335215781708414 647:0.360048508361177 1828:0.543025210572967
```

```
<line> .=. <target> <feature>:<value> <feature>:<value> ... <feature>:<value> # <info>
<target> .=. +1 | -1 | 0 | <float>
<feature> .=. <integer> | "qid"
<value> .=. <float>
<info> .=. <string>
```

1. First term on each line is class. Only binary classification allowed.
2. 0: class unknown (for transductive SVMs)
3. Feature/value pairs must be ordered in increasing feature number.

Multiclass classification

- One-of classification: Combine two-class linear classifiers as follows:
  - Run each classifier separately
  - Rank classifiers (e.g., according to score)
  - Pick the class with the highest score
- Any-of or multilabel classification
  - A document can be a member of 0, 1, or many classes.
  - A decision on one class leaves decisions open on all other classes.
  - A type of "independence" (but not statistical independence)
  - Example: topic classification
  - Usually: make decisions on the region, on the subject area, on the industry and so on "independently"

SVM multiclasses

Crammer and Singer (2002)

1. Train k-classifiers: For k classes,we have decision function

$$\mathrm{H_M}(\vec{x}) = \arg\max_{r=1}^{k} \vec{w}_r \cdot \vec{x}$$

   where w_r is the weight vector for the rth class.

2. Separation: We seek to minimize the sum of the squares of the k weight vectors norms subject to the following constraint, where y_i is the correct class for example x_i:

$$\forall y \neq y_i \, [\, \vec{w}_{y_i}{}^{\mathsf{T}} \vec{x}_i - \vec{w}_y{}^{\mathsf{T}} \vec{x}_i \geq 1 - \zeta_i \,]$$

   That is, ignoring slack vars, the functional margin of the correct class must beat all others by at least 1.

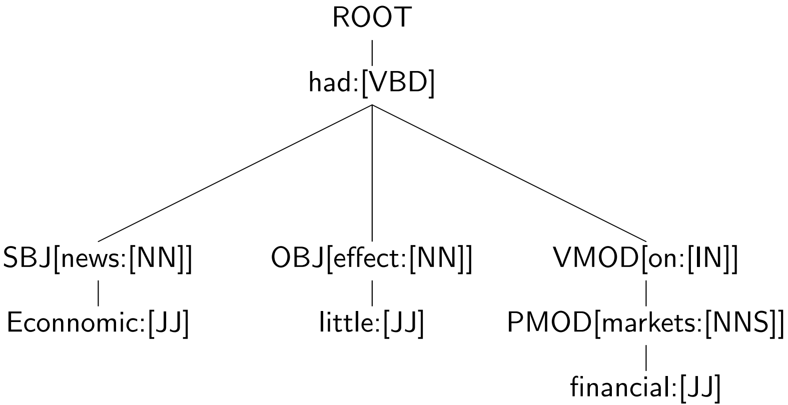
SVM multiclass format

Some sample encoded parse states. Note that successive states may share many features.

```
1 2:1 7909:1 15816:1 23719:1 23764:1 23809:1 23854:1
2 3:1 7910:1 15817:1 23719:1 23764:1 23809:1 23854:1
1 2:1 7910:1 15817:1 23719:1 23764:1 23810:1 23854:1
3 4:1 7911:1 15818:1 23720:1 23764:1 23809:1 23854:1
1 2:1 7910:1 15818:1 23719:1 23764:1 23810:1 23856:1
...
```

Assignment: Part one

Using the example discussed above as a model, show all the parse configurations required to build the following dependency tree.



Given

- One WSJ dependency parse (*Economic news had little impact on financial markets.*)
- Some code for producing a graph from dependency parse.

Produce:

- A transition-based parser.
- With a training oracle
- and with all parse actions implemented.

Testing

*Test your parser on (*Economic news had little impact on financial markets.*). The correct sequence of dependency actions is in these slides.*

Assignment Part two: slide 2 of 3

- You will be given a Python file called dependency_utils.py. Put this in the same directory as your parser file (call it transition_parser.py) and place this line at the top of transition_parser.py.

  ```
  import dependency_utils
  ```

- You will also be given a dependency parse file called wsj_0077_9.gld.

  ```
  >>> gold = 'wsj_0077_9.gld'
  >>> (g_sen, g_dicts) = \
  dependency_utils.collect_dependency_dicts(gold)
  >>> g_dicts
  [{0: [('ROOT', 3)], 2: [('NMOD', 1)], 3: [('SBJ', 2), ('OBJ', 5
  ('P', 9)], 5: [('NMOD', 4), ('NMOD', 6)], 6: [('PMOD', 8)],
  8: [('NMOD', 7)]}]
  >>> g_sen
  ['Economic news had little effect on financial markets .']
  >>> sentence = ['ROOT'] + g_sen[0].split()
  >>> sentence
  ['ROOT', 'Economic', 'news', 'had', 'little', 'effect',
   'on', 'financial', 'markets', '.']
  ```

- Notice `g_dicts` is a list with one dictionary in it. The dictionary uses word indices rather than words. ROOT is the 0th word in the sentence. It is a head bearing the ROOT relation to the dependent at position 3.
- Notice `g_sen` is a list with one sentence string in it. The last line makes the variable `sentence` a list of words. So going back to the ROOT relation in the `g_dict`, to find what word is at index 3 and bears the ROOT relation to 'ROOT' (the word at 0), we do:

```
>>> sentence[3]
'had'
```

Bibliography

Crammer, K., and Y. Singer.
2002.
On the algorithmic implementation of multiclass kernel-based vector machines.
The Journal of Machine Learning Research 2:265–292.

Joachims, T. 1999.
Making large-Scale SVM Learning Practical.
In B. Sch
"olkopf and C. Burges and A. Smola (Ed.), *Advances in Kernel Methods-Support Vector Learning*. MIT-press.

Kübler, S., R. McDonald, and J. Nivre. 2009.
Dependency parsing.
Morgan & Claypool Publishers.
Synthesis Lectures on Human Language Technologies.