Transition-based dependency parsing

Syntactic analysis (5LN455)

2015-12-16

Sara Stymne

Department of Linguistics and Philology

Based on slides from Marco Kuhlmann

- Arc-factored dependency parsing

    Collins' algorithm

    Eisner's algorithm

- Evaluation of dependency parsers

- **Transition-based dependency parsing**

    The arc-standard algorithm

- **Projectivity**

Transition-based dependency parsing

Transition-based dependency parsing

- Eisner's algorithm runs in time $O(|w|^3)$.
  This may be too much if a lot of data is involved.

- Idea: Design a dumber but really fast algorithm and let the machine learning do the rest.

- Eisner's algorithm searches over many different dependency trees at the same time.

- A transition-based dependency parser only builds *one* tree, in *one* left-to-right sweep over the input.

Transition-based dependency parsing

- The parser starts in an initial configuration.

- At each step, it asks a guide to choose between one of several transitions (actions) into new configurations.

- Parsing stops if the parser reaches a terminal configuration.

- The parser returns the dependency tree associated with the terminal configuration.

Generic parsing algorithm

```
Configuration c = parser.getInitialConfiguration(sentence)

while c is not a terminal configuration do

    Transition t = guide.getNextTransition(c)

    c = c.makeTransition(t)

return c.getGraph()
```

Variation

Transition-based dependency parsers differ
with respect to the configurations
and the transitions that they use.

The arc-standard algorithm

The arc-standard algorithm

- The arc-standard algorithm is a simple algorithm for transition-based dependency parsing.

- It is very similar to shift–reduce parsing as it is known for context-free grammars.

- It is implemented in most practical transition-based dependency parsers, including MaltParser.

Configurations

A configuration for a sentence $w = w_1 \ldots w_n$ consists of three components:

- a buffer containing words of w

- a stack containing words of w

- the dependency graph constructed so far

Configurations

- Initial configuration:

    - All words are in the buffer.

    - The stack is empty.

    - The dependency graph is empty.

- Terminal configuration:

    - The buffer is empty.

    - The stack contains a single word.

Possible transitions

- shift (sh): push

  the next word in the buffer onto the stack

- left-arc (la): add an arc

  from the topmost word on the stack, s_1,

  to the second-topmost word, s_2, and pop s_2

- right-arc (ra): add an arc

  from the second-topmost word on the stack, s_2,

  to the topmost word, s_1, and pop s_1

Configurations and transitions

- Initial configuration: ([],[0,...,n],[])

- Terminal configuration: ([i],[],A)

- shift (sh):
  $(\sigma,[i|\beta],A) \Rightarrow ([\sigma|i],\beta,A)$

- left-arc (la):
  $([\sigma|i|j],B,A) \Rightarrow ([\sigma|j],B,A\cup\{j,l,i\})$

- right-arc (ra):
  $([\sigma|i|j],B,A) \Rightarrow ([\sigma|i],B,A\cup\{i,l,j\})$

Example run

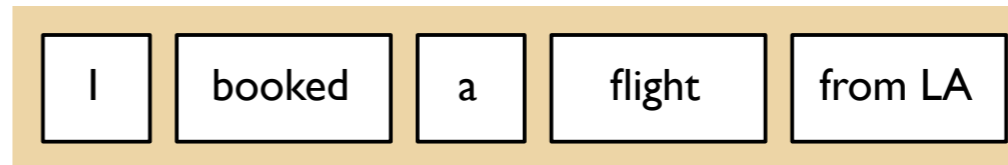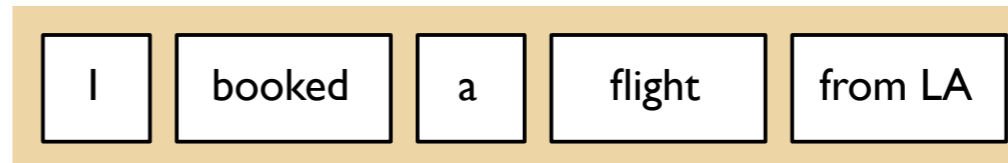Stack

Buffer

| I | booked | a | flight | from LA |

I   booked   a   flight   from LA

Example run

Stack

Buffer

| I | booked | a | flight | from LA |

I   booked   a   flight   from LA

sh

Example run

Stack

Buffer

| I |

| booked | a | flight | from LA |

I   booked   a   flight   from LA

Example run

Stack

	I

Buffer

booked	a	flight	from LA

I   booked   a   flight   from LA

sh
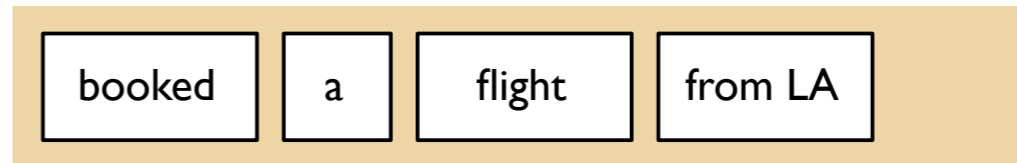
Example run

Stack

	I	booked

Buffer

a	flight	from LA	

I   booked   a   flight   from LA

Example run

Stack

Buffer

| | I | booked |

| a | flight | from LA |

I   booked   a   flight   from LA

la-subj

Example run

Stack
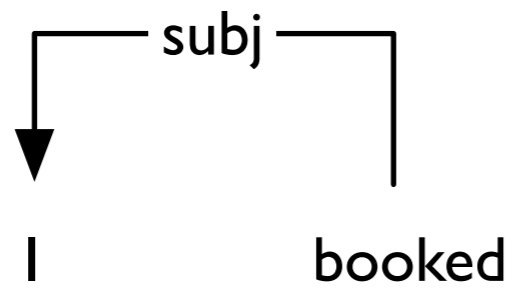
| booked |

Buffer

| a | flight | from LA |

subj

I   booked   a   flight   from LA

Example run

Stack

| booked |

Buffer

| a | flight | from LA |

```
      ┌── subj ──┐
      │          │
      ▼          │
  I       booked       a          flight        from LA
```

sh

Example run

Stack

booked	a

Buffer

flight	from LA

subj

I   booked   a   flight   from LA

Example run

Stack

| booked | a |

Buffer

| flight | from LA |

subj

I   booked   a   flight   from LA

sh

Example run

Stack

booked	a	flight

Buffer

from LA

subj

I   booked   a   flight   from LA

Example run

Stack

| booked | a | flight |

Buffer

| from LA |

subj
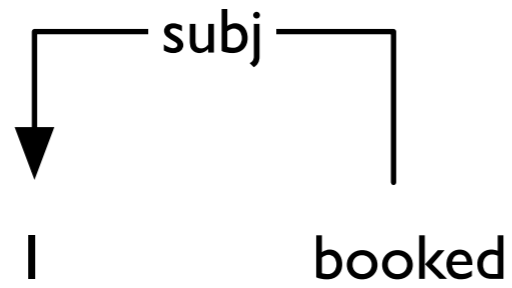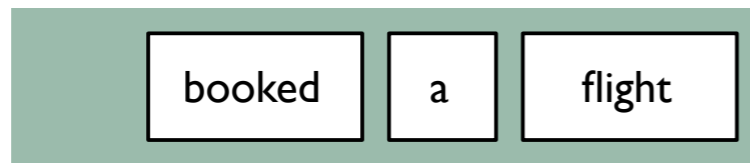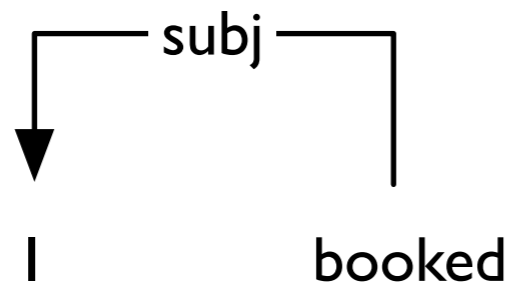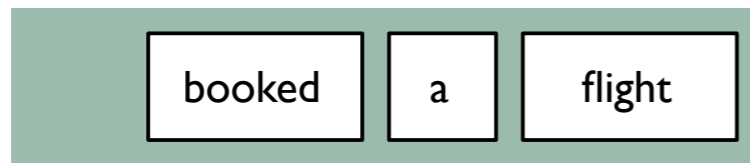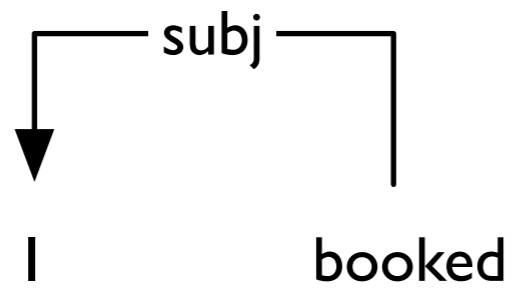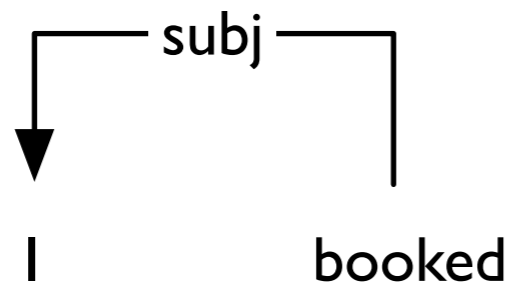
I   booked   a   flight   from LA

la-det

Example run

Stack
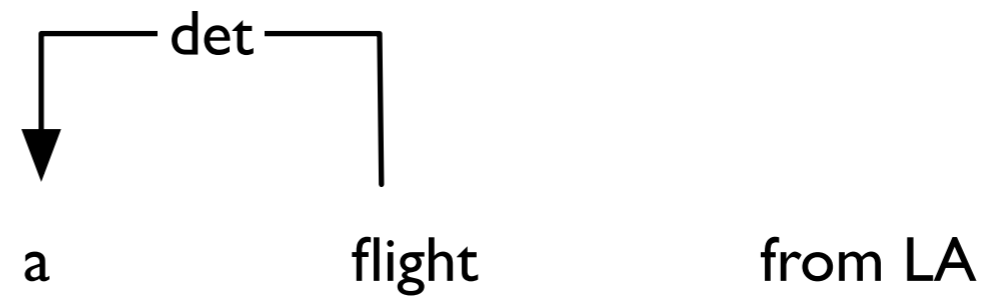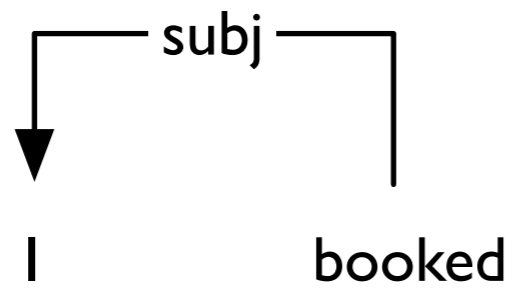
Buffer

| | booked | flight | | from LA | |

subj

det

I booked a flight from LA

Example run

Stack

Buffer

| booked | flight |

| from LA |

subj

det

I   booked   a   flight   from LA

sh

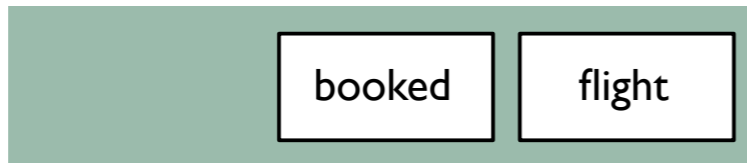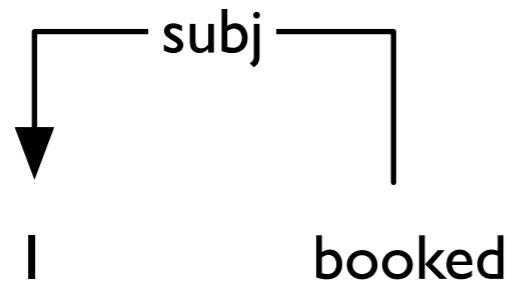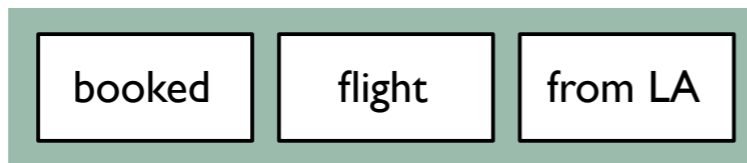# Example run

Stack

| booked | flight | from LA |

Buffer

subj

I ← booked

det

a ← flight   from LA

Example run

Stack

booked | flight | from LA

Buffer

```
    ┌─── subj ───┐              ┌─── det ───┐
    ▼            │              ▼           │
    I          booked           a         flight        from LA
```
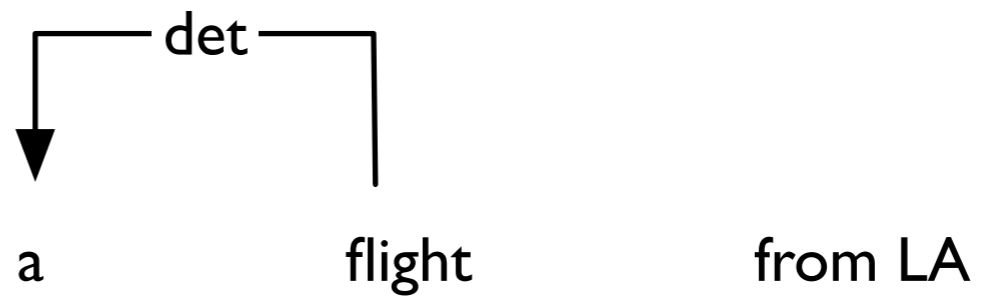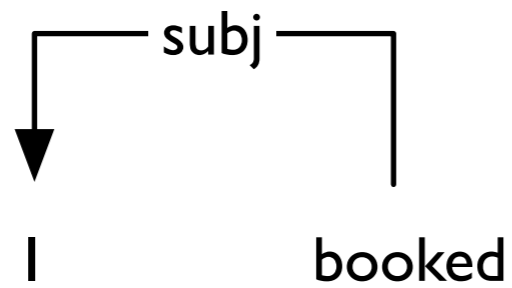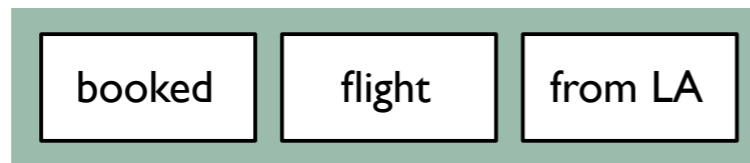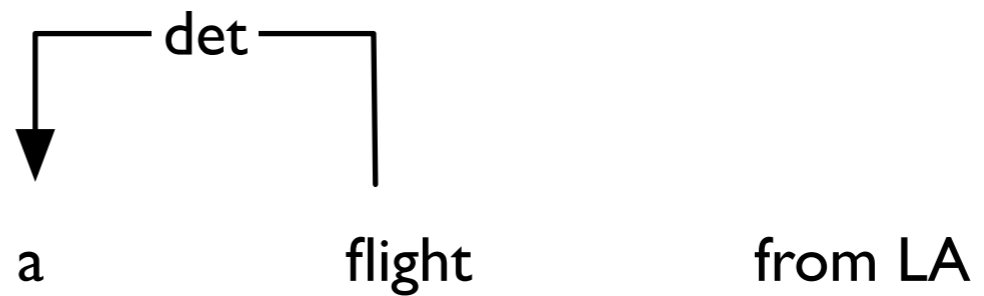
ra-pmod

Example run

Stack
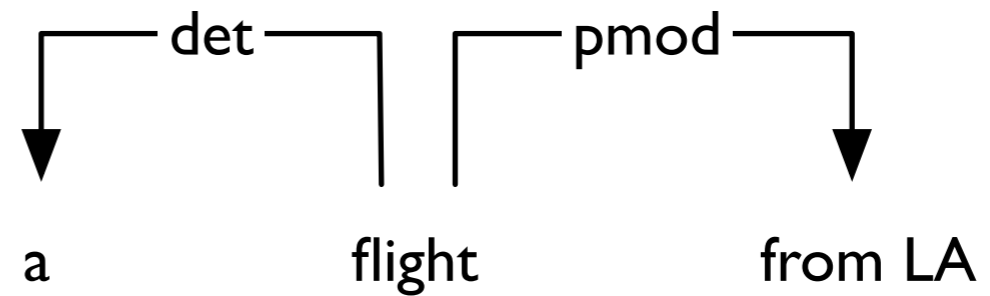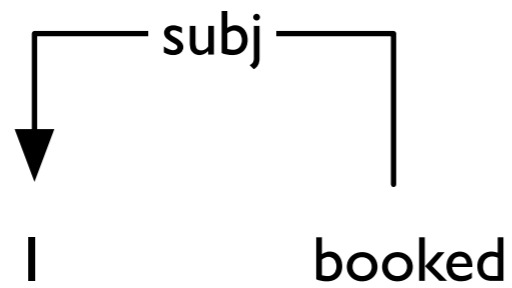
Buffer

| | booked | flight | |

```
    ┌── subj ──┐        ┌── det ──┐   ┌── pmod ──┐
    ↓          │        ↓         │   │          ↓
    I       booked      a      flight      from LA
```

Example run

Stack

Buffer

| booked | flight |

subj

I   booked

det

a   flight

pmod

from LA

ra-dobj

Example run

Stack

Buffer

booked	

dobj

subj   det   pmod

I   booked   a   flight   from LA

Example run

Stack

Buffer

		booked		
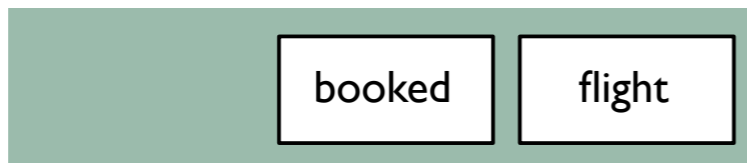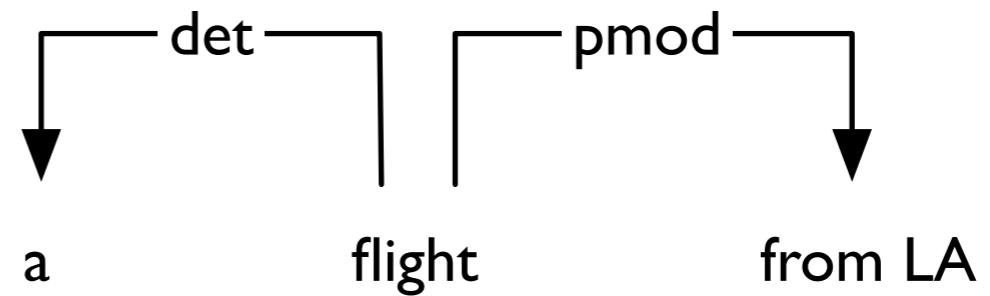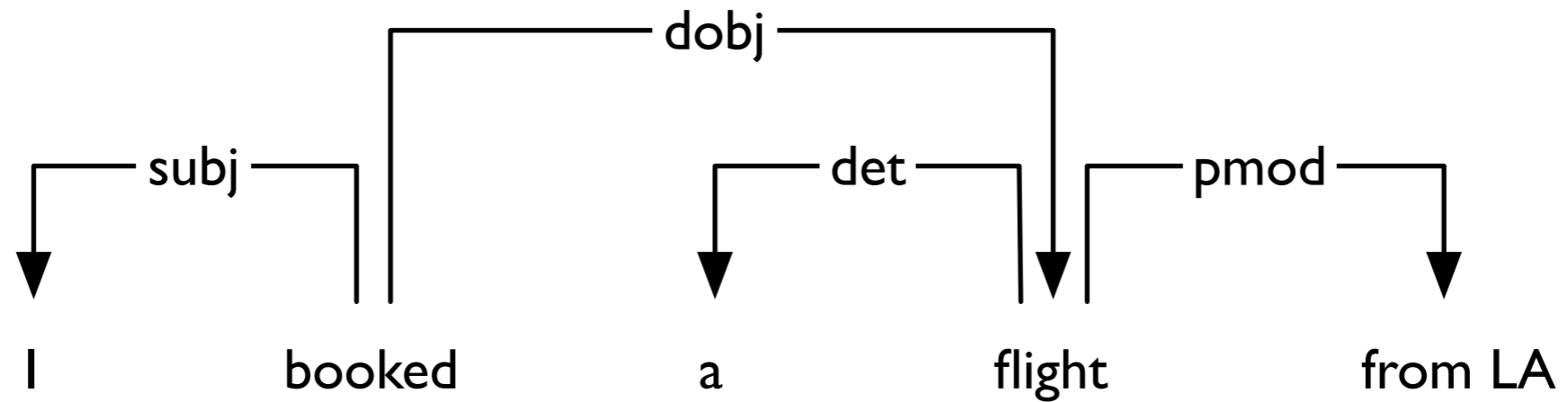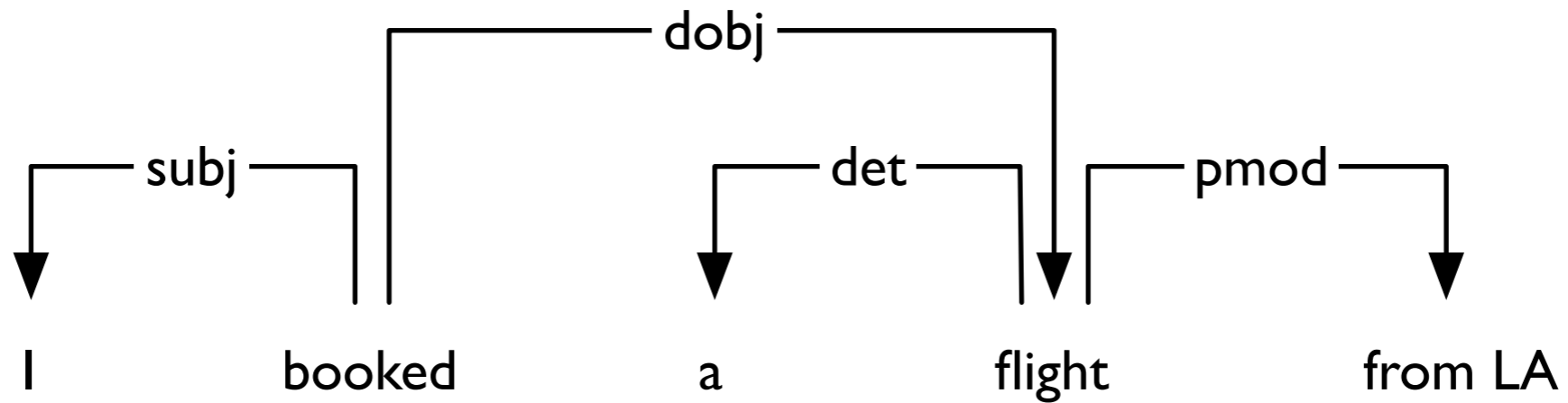
done!

Complexity and optimality

- Time complexity is linear, O(n), since we only have to treat each word once

- This can be achieved since the algorithm is greedy, and only builds one tree, in contrast to Eisner's algorithm, where all trees are explored

- There is no guarantee that we will even find the best tree given the model, with the arc-standard model

- There is a risk of error propagation

- An advantage is that we can use very informative features, for the ML algorithm

Training a guide

Guides

- We need a guide that tells us what the next transition should be.

- The task of the guide can be understood as classification: Predict the next transition (class), given the current configuration.

Training a guide

- We let the parser run on gold-standard trees.

- Every time there is a choice to make, we simply look into the tree and do 'the right thing'™.

- We collect all (configuration, transition) pairs and train a classifier on them.

- When parsing unseen sentences,
  we use the trained classifier as a guide.

Training a guide

- The number of (configuration, transition) pairs is far too large.

- We define a set of features of configurations that we consider to be relevant
  for the task of predicting the next transition.
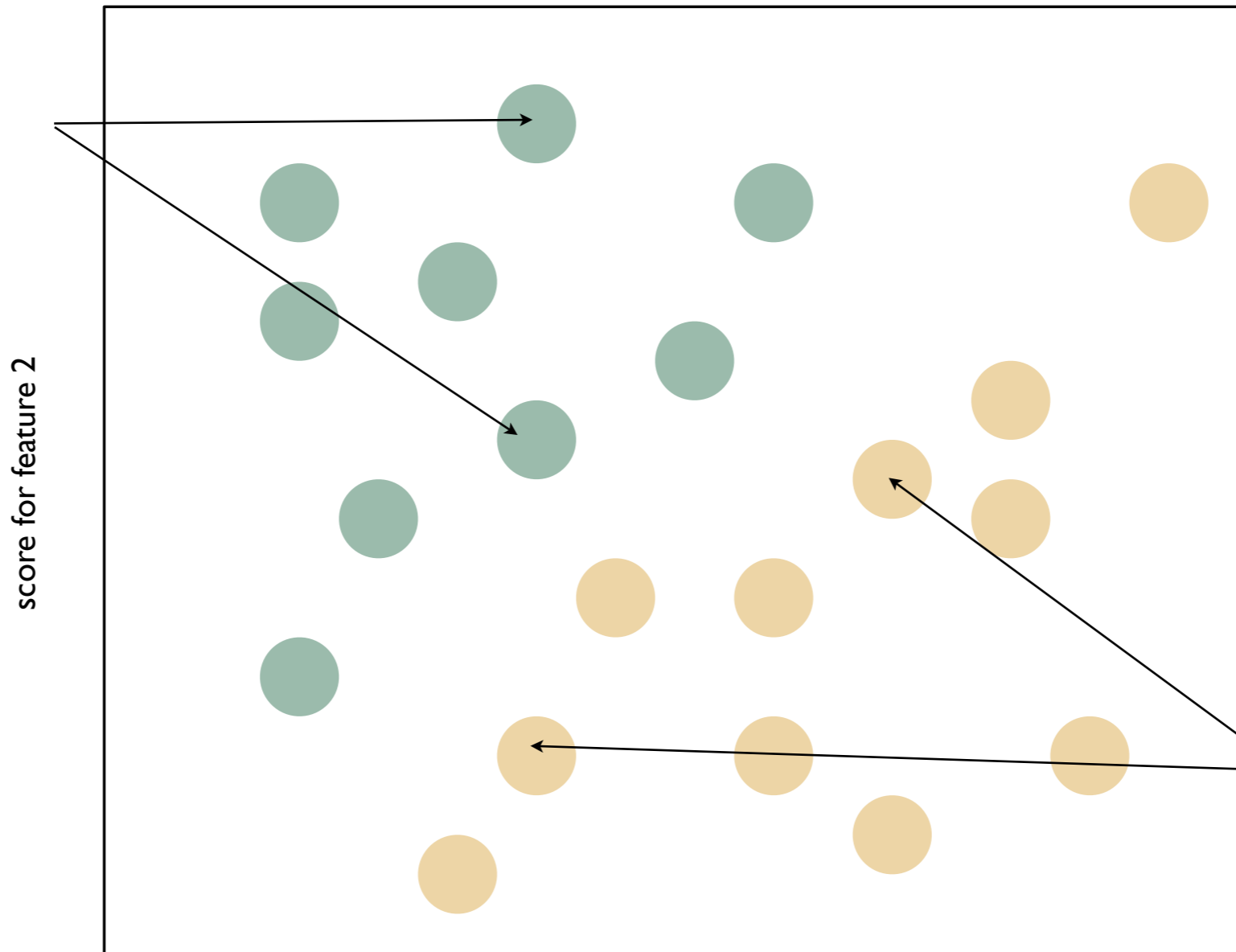
  Example: word forms of the topmost two words
  on the stack and the next two words in the buffer

- We can then describe every configuration
  in terms of a feature vector.

Training a guide

configurations in which
we want to do la
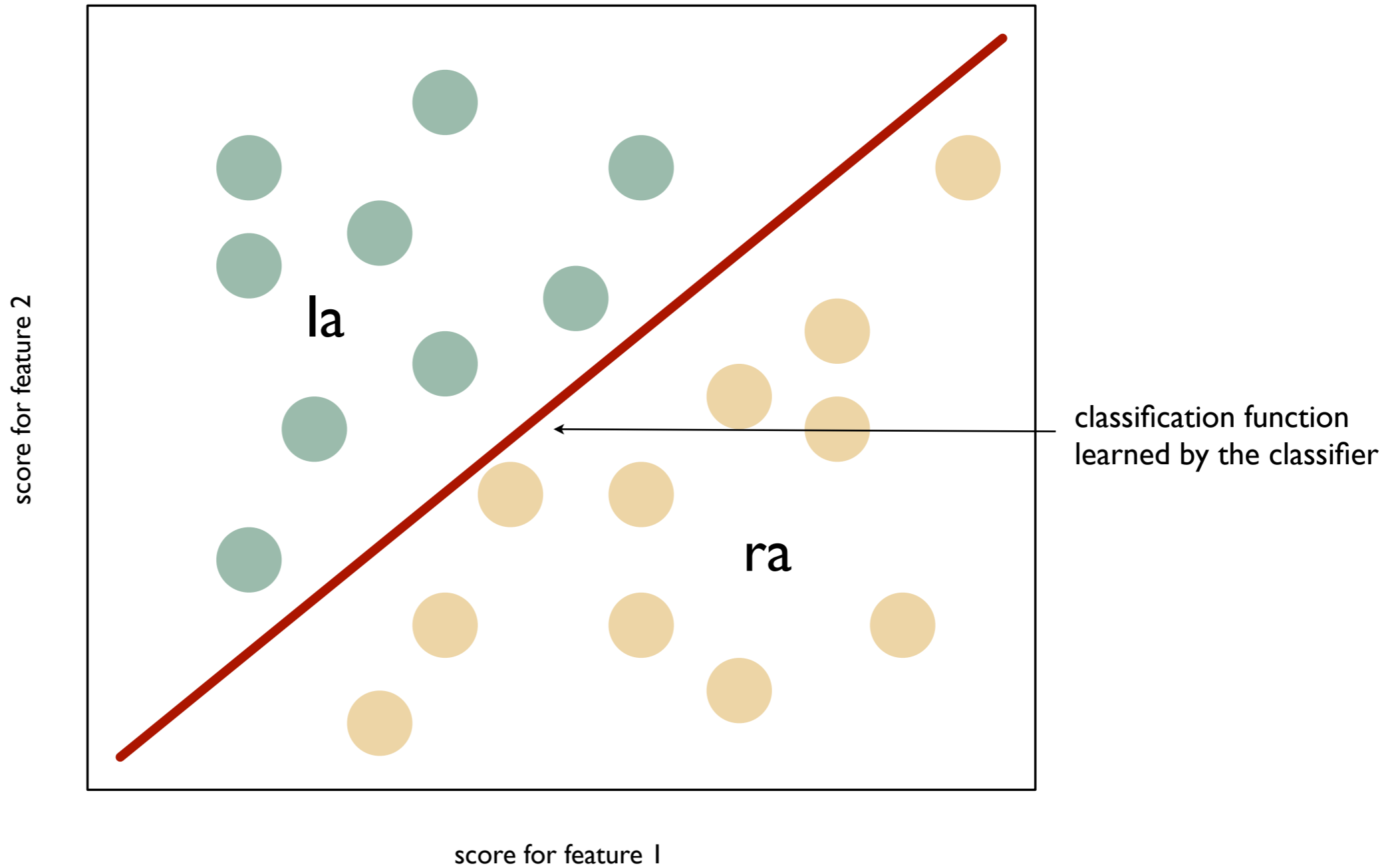
score for feature 2

configurations in which
we want to do ra

score for feature 1

Training a guide

Training a guide

- In practical systems, we have thousands of features and hundreds of transitions.

- There are several machine-learning paradigms that can be used to train a guide for such a task.

  Examples: perceptron, decision trees, support-vector machines, memory-based learning

Example features

Adress	Attributes				
	FORM	LEMMA	POS	FEATS	DEPREL
Stack[0]	X	X	X	X	
Stack[1]	X		X		
Ldep(Stack[0])					X
Rdep(Stack[0])					X
Buffer[0]	X	X	X	X	
Buffer[1]			X		
. . .					

- Combinations of addresses and attributes (e.g. those marked in the table)

- Other features, such as distances, number of children, ...

Alternative transition models

Alternatives

- The arc-standard model as I presented it, is just one example of a transition model

    - In the book you can see another version of the arc-standard model, where arcs are added between the topmost word on the stack and the topmost word on the buffer

- There are many other alternatives

Arc-eager model

- Contains four transitions:

  - Shift

  - Reduce

  - Left-arc

  - Right-arc

- Advantage: not strictly bottom-up, can create arcs earlier than in the arc-standard model

Non-projective transition model

- Allows non-projective parsing by adding a swap transition

- Contains four transitions:

  - Shift

  - Swap

  - Left-arc

  - Right-arc

- Runtime is $O(n^2)$ in the worst case (but usually less in practice)

Transition models in Maltparser

- Nivre family

    - Arcs created between stack and buffer

        - arc-eager model

        - arc-standard (variant from course book)

- Stack family

    - Arcs between two topmost words on stack

        - arc-standard model (variant from slides)

        - models with swap transition

- Other families available as well

Other alternatives

- Parsing with beam search

  - Instead of just keeping the 1-best tree, keep a beam of the k-best trees in each step

  - Requires scoring and ranking of transition sequences

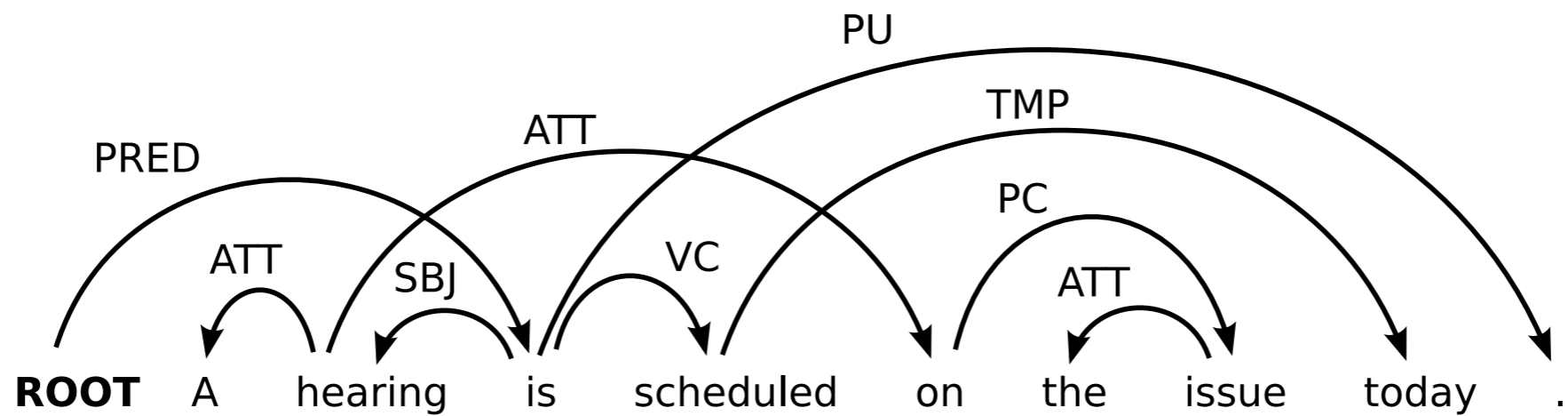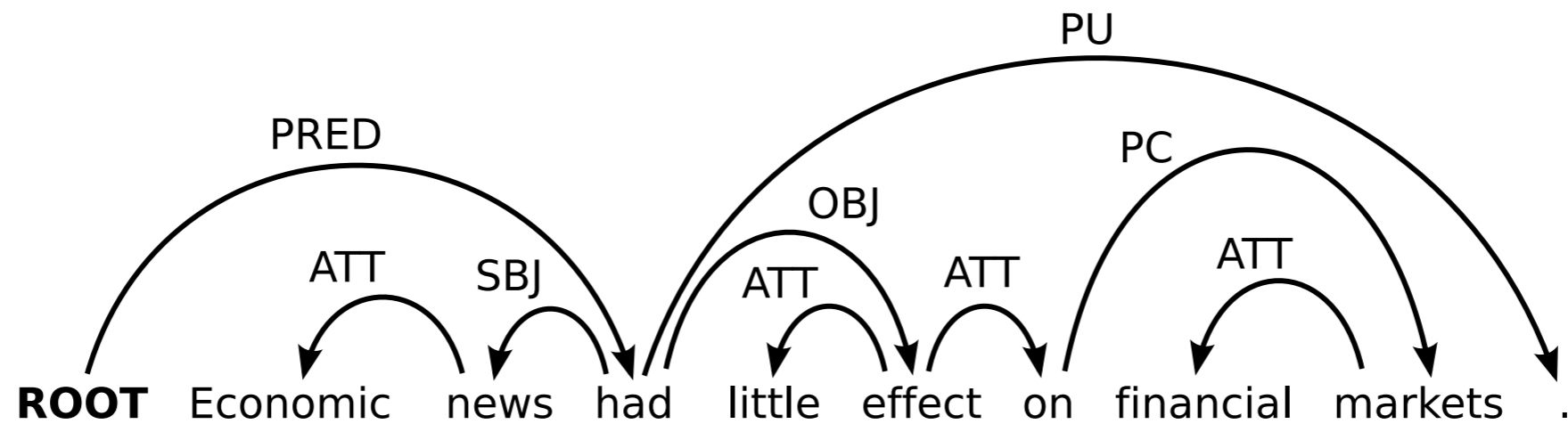 - Complexity: O(nk)

Projectivity

Projectivity

- A dependency tree is projective if:

  - For every arc in the tree, there is a directed path from the head of the arc to all words occurring between the head and the dependent (that is, the arc (i,l,j) implies that i →* k for every k such that min(i, j) < k < max(i, j))

Projective and non-projective trees

Projectivity and dependency parsing

- Many dependency parsing algorithms can only handle projective trees

    - Including all algorithms we have discussed in detail

- Non-projective trees do occur in natural language

    - How often depends on language (and treebank)

Non-projective dependency parsing

- Variants of transition-based parsing

  - Using a swap-transition

  - Using more than one stack (Covington)

- Graph-based parsing

  - Minimum spanning tree algorithms

- Post processing

  - Pseudo-projective parsing (seminar 2)

  - Approximate non-projective parsing

Summary

- In transition-based dependency parsing one does not score graphs but computations, sequences of (configuration, transition) pairs.

- In its simplest form, transition-based dependency parsing uses classification.

- One specific instance of transition-based dependency parsing is the arc-standard algorithm.

The end of the course

- Seminar 2, Pseudo-projective parsing

  - Easier and shorter article than last seminar, some more general questions

- Assignment 3: Disambiguation in arc-factored and transition-based parsing

- Assignment 4: Try and evaluate MaltParser

- Supervision on demand, mainly by email

- Course evaluation in the student portal