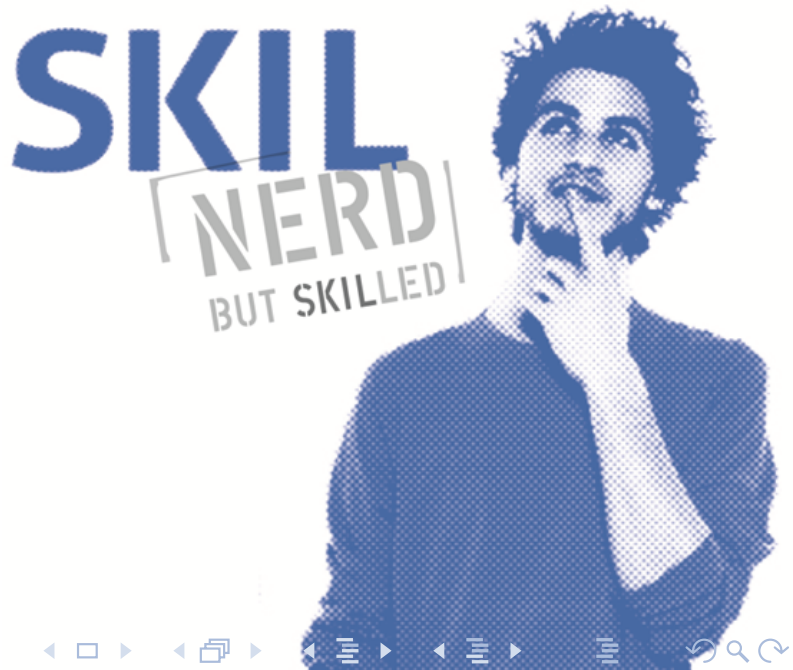
SKIL Studentenkonferenz Informatik Leipzig
25. September 2012

A Lexeme-Clustering Algorithm for Unsupervised Learning of Morphology

Maciej Janicki

September 24, 2012

Introduction



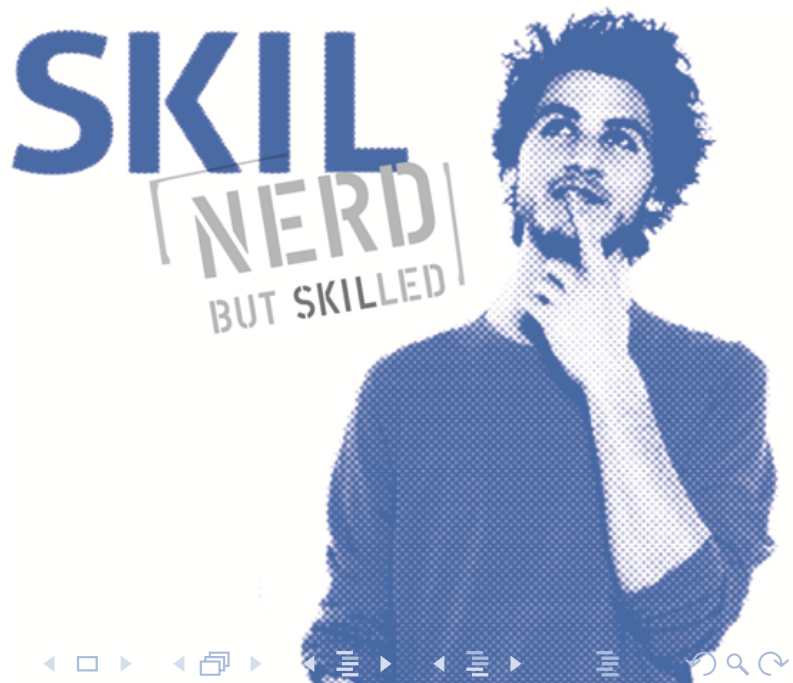
Unsupervised Learning of Morphology

- only plain textual data available
- goal: a monolingual dictionary containing:
  - a list of base forms (infinitives, nominatives etc.)
  - a list of morphological paradigms (declension and conjunction tables etc.)
  - assignment of each base form to a paradigm

The "Group and Abstract" approach

- **group words belonging to the same lexeme**
  e.g. *Haus*, *Hauses*, *Häuser*, *Häusern* are all forms of the word *Haus*
- abstract morphological classes from lexemes
  e.g. "The nouns from a class *X* have forms with suffixes: -, *-es*, *¨-er*, *¨-ern*; the word *Haus* belongs to this class."

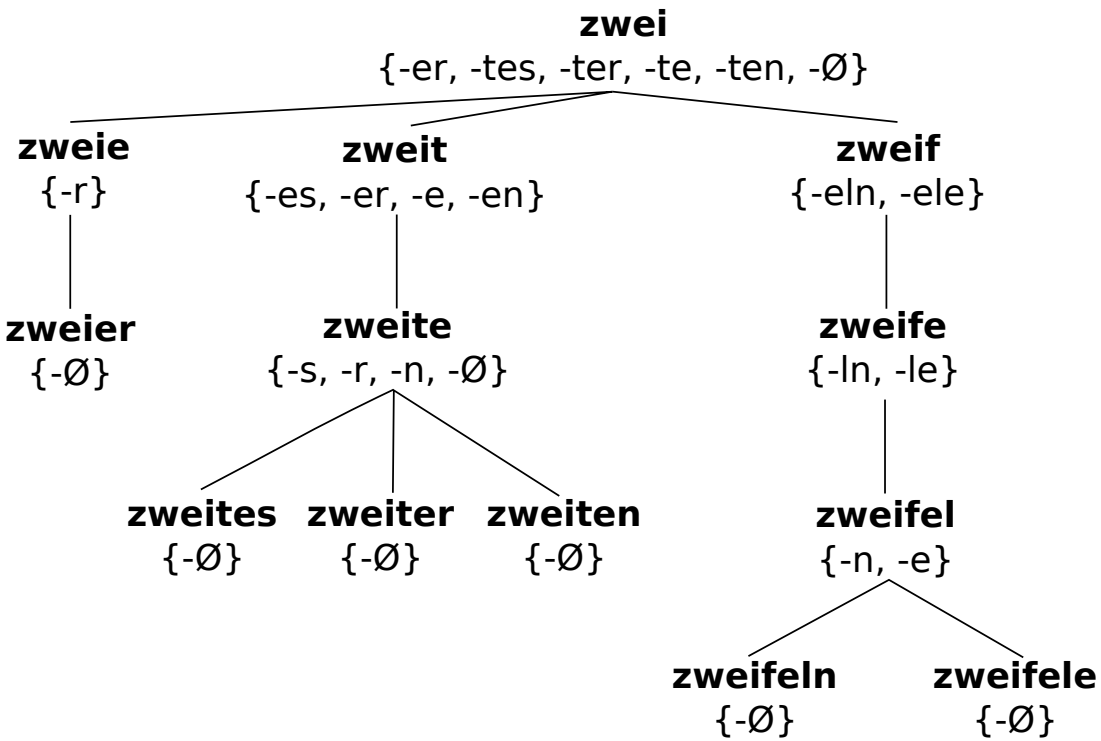
The algorithm



Trie with suffix sets (1)

Idea:

1. Build a trie out of input words.
2. In each node, store possible suffices, that occur with the stem stored in the node (**suffix filtering:** consider only suffices with frequency > 0.001)



Trie with suffix sets (2)

Control questions:

- Why are *feln*, *fele* not in the suffix set of *zwei-*?
- How could we now try to generate all forms of *zweier*?
  - **Proposition:** take a prefix of *zweier* and all its possible suffixes
  - **Problem:** which prefix to take?
    zwei- overgenerates (we consider *ZWEITE* a separate lexeme)
    zweie- undergenerates (we don't capture the base form *zwei*)
  - **Proposition:** split suffix sets into parts, that won't overgenerate a lexeme

Suffix sets partitioning (1)

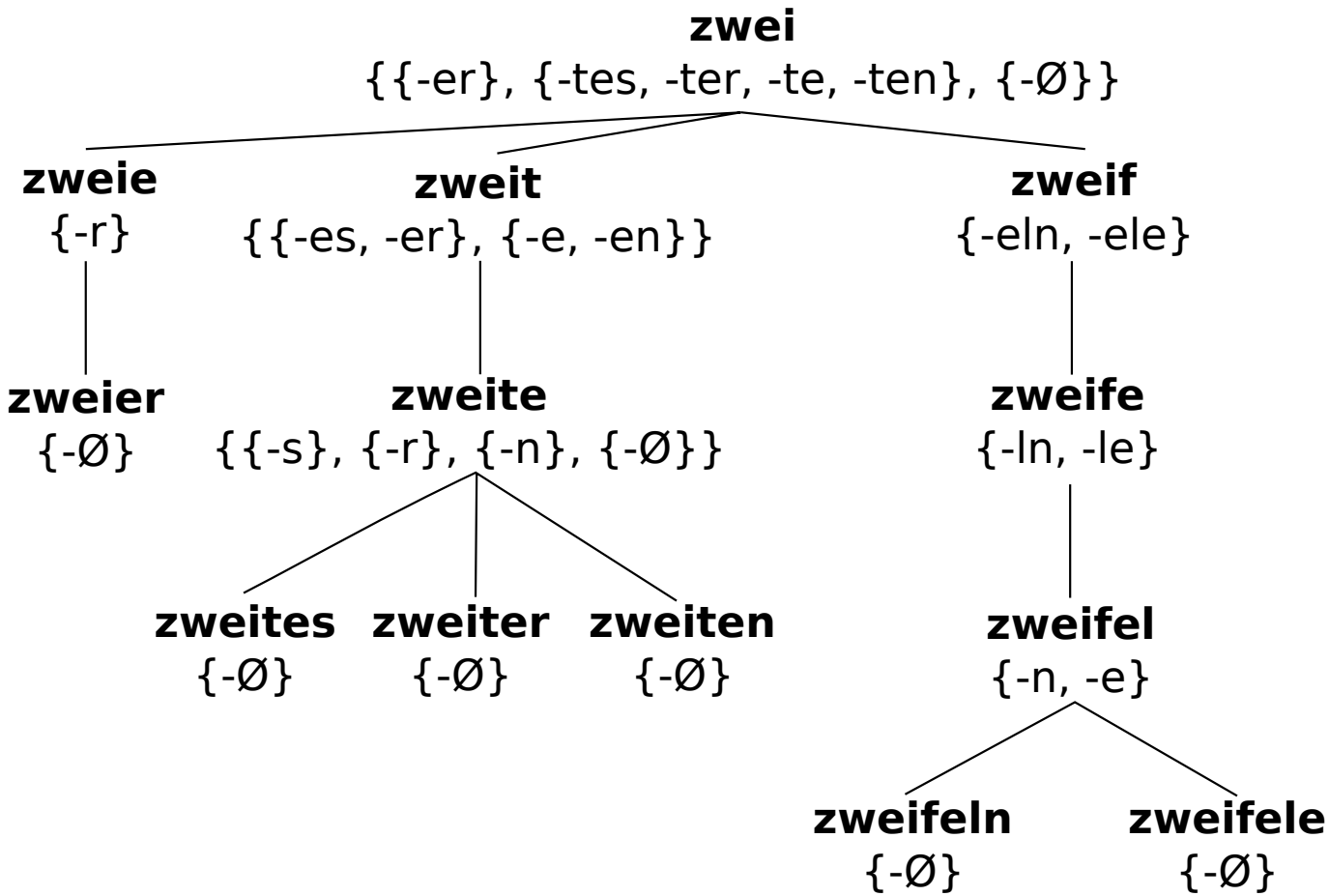
$I(X;Y)$ – *mutual information* of two random variables.

Let κ be a constant, that we call *independency threshold*.
Let S be a suffix set and $\mathcal{S} = \{S_1, S_2, \ldots, S_n\}$ be a partitioning of S.
We call $\mathcal{S}$ **partitioning into paradigms**, iff:

1. $I(S_j, S_k) < \kappa$ for every $j, k \in \{1, \ldots, n\}$, $j \neq k$
2. $I(A, B) \geq \kappa$ for every j, A, B: $A \cup B = S_j$

(low dependency *between* parts, high dependency *inside* each part)

Suffix sets partitioning (2)

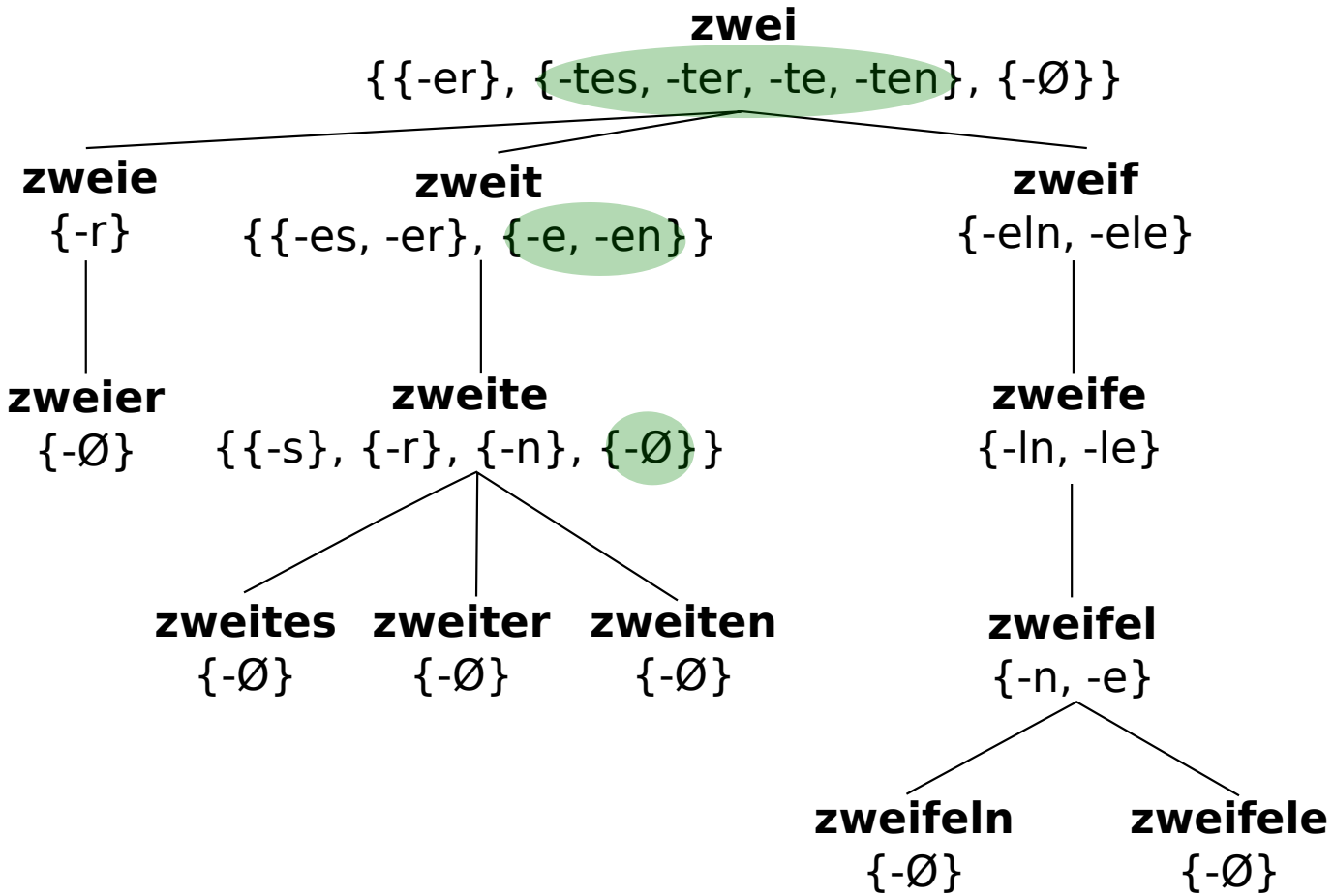


Extracting lexemes (1)

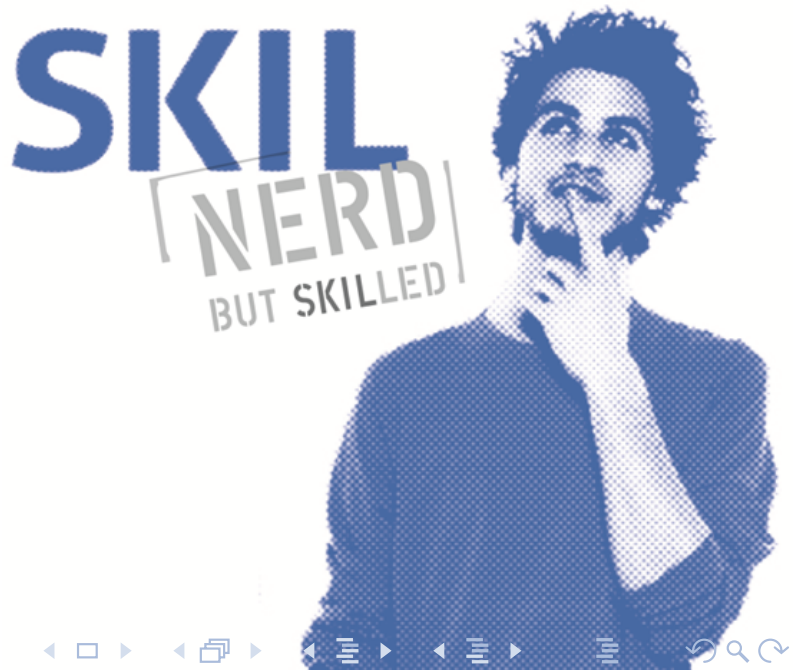
Goal: Given a word, find other words, that belong to the same lexeme.
Example: *zweite* $\mapsto$ {*zweites*, *zweiter*, *zweite*, *zweiten*}
Method: In every node of the trie, find the appropriate paradigm and use all its suffixes to generate word forms (see picture in the next slide).

Extracting lexemes (2)

lex(**"zweite"**):



Evaluation



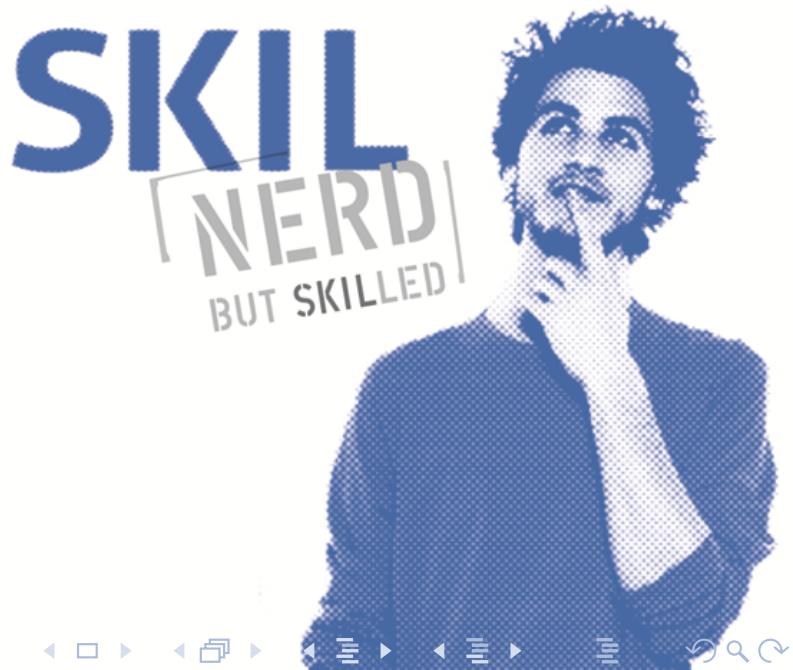
Evaluation

Input: A list of words (alphabetically ordered).
Reference data: A list of lexemes.
Method:

- for each word from the input, generate its lexeme
- match it against this word's lexeme in reference data

Testing set	Words	Lexemes	κ	Precision	Recall	F-measure
Finnish	9k	1k	0.002	**98.6 %**	81.3 %	89.1 %
German	54k	20k	0.01	97.7 %	79.1 %	87.4 %
Polish	980k	61k	0.002	96.1 %	79.0 %	86.7 %
Latin (decl.)	210k	22k	0.05	95.4 %	**92.4 %**	**93.9 %**
Latin (full)	586k	26k	0.015	85.7 %	**41.3 %**	55.8 %
Latin (full)	586k	26k	0.02	67.1 %	81.7 %	73.7 %

Conclusion



Conclusion

Advantages:

- good evaluation results
- customizable, only one parameter with well-defined meaning
- works with different types of morphologies
  - German – small inflected
  - Polish, Latin – large inflected
  - Finnish – agglutinative

Disadvantages:

- depends on the κ parameter
- only suffixes are considered, prefixes and alternations are not
- low performance