

Learning Word Structure: Two Novel Algorithms

Ksenia Shalnova

Department of Computer Science
University of Bristol

17 January 2008

Outline

- Framework of the current research
- Tree of Aligned Suffix Rules - TASR
(1-st algorithm)
- Unsupervised Approach for Suffix Extraction
(2-nd algorithm)

Framework of the Current Research

The Local Language Speech Technology Initiative (LLSTI) provided *information access by voice for developing countries* (Text-to-Speech and Automatic Speech Recognition systems).

Our partners in Africa and Asia



Automatic Analysis of Word Structure was missing in all voice systems and led to the great lack of quality in these systems.

Framework of the Current Research

What is morphological analysis or morphological grammars?

- Decomposition of a word into its constituents – stem and prefixes/suffixes/infixes
- Represented as finite state automaton (most European languages) or by finite state automaton with memory (some phenomena in Asian/African languages)

ev(*stem - “house”*)+**de**(*in*)+**ki**(*that*)+**ler**(*the ones*)
“*people in the house*” (*Turkish*)

stalk(*stem - “bump”*)+**nu**(*dummy*)+**vsh**(*past tense, active*)+**ij**(*sg, Nom. case, masc.*)+**sa**(*reflexive*)
“*being bumped*” (*Russian*)

dep(*stem - “buy”*)+**pe**(*not*)
“*not buying*” (*Ibibio*)

Framework of the Current Research

Why automatic learning of morphology is so important?

- Building the rule database for morphological analysers by hand (stem/affix lists and rules of their concatenation) is extremely time-consuming.
- Morphology is used in all speech and language technologies: Machine Translation, Text-to-Speech, Text Mining, Spell checkers etc.

Why automatic learning of morphology is not widely used in

development process ? ? ?

Tree of Aligned Suffix Rules (TASR)

Input 1 set of wordpairs (training set):

word_grammatical_form--word_basic_form (*stayed-stay, said-say ...*)

Functionality 1

given any word_grammatical_form **generate** word_basic_form

Input 2 set of wordpairs (training set):

word_basic_form--word_grammatical_form (*stay-stayed, say-said ...*)

Functionality 2

given any word_basic_form **generate** word_grammatical_form

The Concept of TASR

Definition 1 (Aligned suffix rules) Rule $LHSuff \rightarrow RHSuff$ is aligned with rule $LHSuff' \rightarrow RHSuff'$ if there is a wordpair with left-hand word and right-hand word $LW-RW$ for which $LW=s+LHSuff=s'+LHSuff'$ and $RW=s+RHSuff=s'+RHSuff'$ where s and s' are the prefix substrings in LW and RW , and “+” denotes string concatenation.

For example, the wordpair *stay-stayed* generates the following aligned suffix rules:

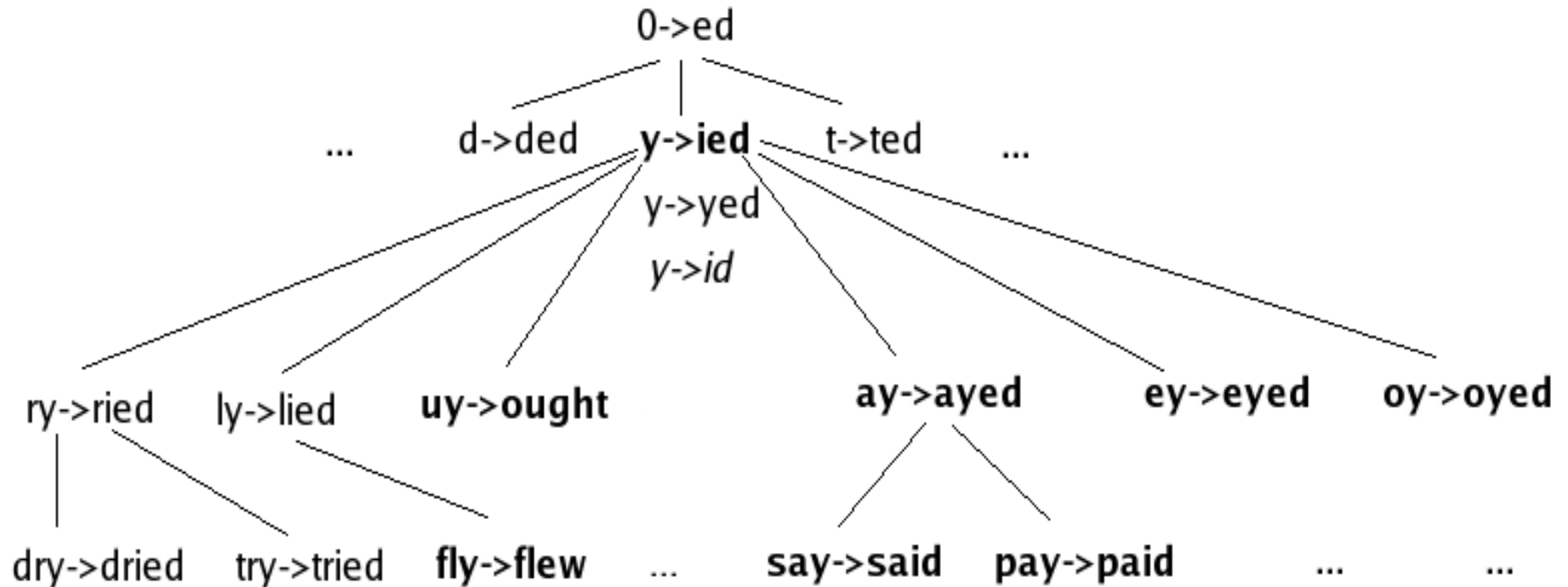
LHSuff	→	RHSuff
<i>stay</i>		<i>stayed</i>
<i>tay</i>		<i>tayed</i>
<i>ay</i>		<i>ayed</i>
<i>y</i>		<i>yed</i>
<i>0</i>		<i>ed</i>

The Concept of TASR

Definition 2 (TASR) Tree of Aligned Suffix Rules (TASR) is a tree with the following properties:

- The root of the tree is the $LHS_{\text{uff}} \rightarrow RHS_{\text{uff}}$ rule with the minimal length of LHS_{uff} .
- Nodes represent aligned suffix rules $LHS_{\text{uff}} \rightarrow RHS_{\text{uff}}$. The winning rule has got the highest frequency (Definition 3) and is not subsumed (Definition 4) by the rule in a parent node.
- LH_{uff}' in child node is represented by lefhand one letter expansion of LHS_{uff} in parent node.

TASR for English Past Tense Verbs Terminating on Letter “y”



The Concept of TASR

Definition 3 (Suffix rule with higher frequency) Rule $LHS_{\text{uff}} \rightarrow RHS_{\text{uff}}$ has got higher frequency than rule $LHS_{\text{uff}'} \rightarrow RHS_{\text{uff}'}$ if the number of wordpairs $LW-RW$ where $LW=s+LH_{\text{suff}}$ and $RW=s+RHS_{\text{suff}}$ is larger than the number of wordpairs where $LW=s+LH_{\text{suff}'}$ and $RW=s+RHS_{\text{suff}'}$.

Definition 4 (Rule subsumption) Child node $LHS_{\text{uff}} \rightarrow RHS_{\text{uff}}$ is subsumed by parent node $LHS_{\text{uff}'} \rightarrow RHS_{\text{uff}'}$ if there exists a character x such that $LHS_{\text{uff}}=x+LH_{\text{suff}'}$ and $RHS_{\text{uff}}=x+RHS_{\text{suff}'}$.

$y \rightarrow ied$ is NOT subsumed by $ay \rightarrow ayed$ and is subsumed by $ry \rightarrow ried$

TASR

Extract List of Aligned Suffix Rules

Input: set of wordpairs: LW-RW

for each word pair_i: <LW> - <RW>

s = first letter in word LW;

proc extract aligned suffix rules (s)

1. Extract the rule $LHSuffix \rightarrow RHSuffix$ if it fits the following match: $LW = s + LHSuffix$ and $RW = s + RHSuffix$ where s is the coinciding string;
2. $s' = s$ where substring s' is expanded by next letter after letter s from LW.
3. extract aligned suffix rules (s').

end each

TASR

Build a Tree on the Basis of Aligned Suffix Rules

Input: set of unique aligned suffix rules with their frequencies

Current node of the tree=Root;

LHSuff="0" (or empty string);

proc Build Tree (LHSuff)

1. For LHSuff find the RHSuff with largest frequency (see Definition 3);
2. Current node of the tree = LHSuff -> RHSuff with largest frequency;
3. Check subsumption criterion

if (rule LHSuff -> RHSuff is not SUBSUMED by LHSuff -> RHSuff (see Definition 4))

mark the rule LHSuff -> RHSuff as the winning one.

end if

For all lefthand 1-letter extensions LHSuff' of LHSuff call Build Tree (LHSuff') and add the resulting sub-tree as child of Current node.

TASR (Summary)

1. The winning rule has got the following conjunction of criteria:
 - it has higher frequency than other rules with the same LHSuff;
 - it is not subsumed by the rule in the parent node
2. The tree is searched bottom-up
3. The complexity of TASR is $O(n * m \log n * m)$.

TASR (Summary)

Functionality of TASR

- The algorithm provides the function learning (given basic word form – generate grammatical form of this basic form or vice versa).
- The algorithm does not generate overlapping rules.
- The algorithm does not require explicit negative examples.

TASR in comparison to CLOG and FOIDL differ in the following way

- Does not depend on the order of the training examples
- Provides rules in the form of a tree
- Better performance
- Less amount of the induced rules
- Much faster

Comparison of CLOG and TASR Performance

DATA SET	CLOG (MEAN OVER 10 FOLDS)	TASR (MEAN OVER 10 FOLDS)	T-TEST BETTER?
ENGLISH PAST TENSE	86.4±2.8	88.3±3.6	✓
GENITIVE CASE (RUSSIAN)	90.1±3.7	92.1±3.2	✓
DATIVE CASE (RUSSIAN)	90.4±3.3	92.7±3.4	✓
INSTRUMENTAL CASE (RUSSIAN)	89.8±3.7	90.6±4.1	-
PREPOSITIONAL CASE (RUSSIAN)	90.8±3.3	91.7±3.3	-
ACCUSATIVE CASE (RUSSIAN)	87.2±6.0	84.1±5.9	x

Comparison of CLOG and TASR

DATA SET	AVERAGE NUMBER OF GENERATED RULES		DATA SET	CLOG (AVERAGE TIME)	TASR (AVERAGE TIME)
	CLOG	TASR			
ENGLISH PAST TENSE	183	159	ENGLISH PAST TENSE (1242 EXAMPLES)	3 min	0.3 min
GENITIVE CASE (RUSSIAN)	99	79			
DATIVE CASE (RUSSIAN)	95	65			
INSTRUMENTAL CASE (RUSSIAN)	103	86			
PREPOSITIONAL CASE (RUSSIAN)	95	75			
ACCUSATIVE CASE (RUSSIAN)	139	114			

Comparison of FOIDL and TASR on English Data Set

ALGORITHM M	PERFORMAN CE	AVERAGE TIME	AVERAGE NUMBER OF RULES
TASR	88.3	0.3 min	159
FOIDL	82.2	90 min	175

TASR Performance with Differentiating Grammatical Features

DATA SET	FEATURE_VALUE	PERFORMANCE (%)
GENITIVE CASE	F	99.3
	M	96.3
	N	99.6
	Total	98.4
DATIVE CASE	F	99.3
	M	96.7
	N	99.6
	Total	98.5
INSTRUMENTAL CASE	F	99.1
	M	94.8
	N	99.3
	Total	97.7
PREPOSITIONAL CASE	F	99.3
	M	96.3
	N	99.2
	Total	98.3
ACCUSATIVE CASE	Animated	97.5
	Not_Animated	97.8
	Total	97.7

Unsupervised Approach for Suffix Extraction

Input – list of `suffix_seq` obtained after applying TASR.

stalk(stem - “bump”) + `suffix_seq1` = *nuvshijsa*
... + `suffix_seq2` = *ovaemogosa*
...

Output– list of suffixes and rules of their concatenation.

nu(dummy)+*vsh(past tense, active)*+*ij(sg, Nom. case, masc.)*+*sa(reflexive)*

□ **Unsupervised Approach for Suffix Extraction**

- Generate Initial Suffix List
(using global pairwise alignment and a heuristics)
- High-level suffix representation
- Generate a set of patterns or regular expressions
(using multiple sequence alignment based on high-level suffix representation)

□ **Unsupervised Approach for Suffix Extraction (Generate Initial Suffix List)**

*In the language, there are only differences, without positive terms ...
When you come to the terms themselves, resulting from relations
between signifying and signified elements you can speak of oppositions
(Ferdinand de Saussure).*

A substring is a good **suffix candidate** if a strong evidence about oppositions for this substring can be proved.

Two types of substring context required for finding an opposition:

- Letter context (letter to the right and to the left from a particular substring);
- Relative position of a particular substring in the sequence of other substrings within one `suffix_seq`;

Unsupervised Approach for Suffix Extraction (Generate Initial Suffix List)

INPUT: set of suffix_seq (*lcdf, adcdl, acdlt, akmltuv ...*)

A substring is considered to be a suffix if two oppositions for this substring can be found in set of suffix_seq:

1. There is a pair of suffix_seq where substring occurs in a different letter context. For example, substring *cd* occurs in two suffix_seq surrounded by different letters: *lcdf* and *adcdl*.
2. There is a pair of suffix_seq where substring occurs in the same letter context as substring1. For example, for substring=*cd* and substring1=*km* there exists a pair of suffix_seq: *acdlt* and *akmltuv*.

OUTPUT: set of suffixes (*cd, ...*)

```
Global FTFTALILLAVAV  
F--TAL-LLA-AV
```

```
Local FTFTALILL-AVAV  
--FTAL-LLAAV--
```

For each pair of suffix_seq global pairwise alignment was used.

□ **Unsupervised Approach for Suffix Extraction
(Higher-Level Suffix Representation)**

INPUT: set of `suffix_seq` (*xyzegolgrijab, ...*)
 set of `suffixes` (*ego,ij, ...*).

Higher-level suffix_seq	* <i>xyz</i>	A	* <i>lgr</i>	B	* <i>ab</i>
Original suffix_seq	<i>xyz</i>	<i>ego</i>	<i>lgr</i>	<i>ij</i>	<i>ab</i>
Segment match with suffixes	dummy suffix	match	dummy suffix	match	dummy suffix
Position number	5	4	3	2	1

Representing `suffix_seq` *xyzegolgrijab* as ***A*B***

OUTPUT: set of `suffix_seq` represented as a higher-level suffix sequence

□ Unsupervised Approach for Suffix Extraction (Pattern Table)

INPUT: set of `suffix_seq` represented as a higher-level suffix sequence

1. Apply multiple sequence alignment for set of `suffix_seq`;

5	4	3	2	1
A	B	* (<i>e</i>)	F	W
O	G	* (<i>e</i>)	F	
* (<i>e</i>)	G	* (<i>e</i>)	F	
...
		*	F	

2. Update dummy_suff * depending on the , for example, ***F-->K:**

AB*FW-->ABKW, OG*F-->OGK...

3. Apply multiple sequence alignment for updated set of `suffix_seq`;

OUTPUT: final `pattern_table` or regular expressions

Final Pattern Table and Underlying Assumptions

4 (derivational suffixes)	3 (tense)	2 (case, gender, number)	1 (reflexive)
<i>A=nu</i>	<i>B=vsh</i>	<i>K=ij</i>	<i>W=sa</i>
<i>O=ova</i>	<i>G=nn</i>	<i>K=ij</i>	
...

- One particular suffix can occur only once in a `suffix_seq` that allows to avoid recursion (e.g., cases like *laegohaego* where *ego* is a suffix in both positions are not possible).
- One suffix can occur only in one particular occurrence position (or column) in `suffix_seq`.

Results and Future Work

Unsupervised algorithm gave 100% performance on training set.

To Do:

1. Combine the current approach with stochastic mechanisms.
2. Apply similar approach for other language types with other underlying assumptions (including recursion).