Segmentation from 97% to 100%: Is It Time for Some Linguistics?

Petr Sojka
Masaryk University in Brno, Faculty of Informatics
sojka@fi.muni.cz

December 8th, 2012

Segmentation Task

- In NLP, we often *segment*. Texts into paragraphs, paragraphs into sentences, sentences into phrases, phrases into words, words into syllables or words into stems for indexing in IR, hyphenation,...
- Many NLP tasks could be transformed into segmentation tasks (divide et impera), like PoS tagging: sequence of [ambiguious] tags could be 'segmented' after the right tag.
- It is frequent, often used task, that needs to be solved *quickly*, in *limited space* (memory matters) and linguistic data are typically *big*, full of *exceptions*, and *change* slowly *in time* (decades, even years).

- $\longrightarrow$ *language-aware* machine learning (as oposed to brute-force machine learning).

Machine Learning

- Recognition of *rules/patterns* in NLP (textual) data: catching regularities in empirical data.
- in 1960's Michal Chytil and Hájek started to generate unary hypotheses on finite models using the GUHA method: *p-truth formulas* (Hájek, Havránek).
- Standard way today: data [features] represented as points in high-dimensional space: methods (like SVM) then try to cover/separate different set of points by hyperplanes.
- Noise added to data to get from 85% to 87% to optimize hyperplane (like MALT parsing with SVM). Principally wrong, as ignoring [linguistic] data properties: will never reach 100%.
- Remedy? $\longrightarrow$ For *some* tasks *competing patterns approach/ methodology* [Sojka 2003].

Patterns

- Recognition of rules/patterns in (textual) data: catching regularities (and iregularities) in empirical data.
- NLP examples: SVOMPT. Syllable CV patterns (V, CV, CCV,...).
- Main problem with patterns: they only *cover*, no contextual handling and no negation.

Competing patterns: algebraic definition (words over free monoid $\langle (\Sigma \cup V)^*, \varepsilon, \cdot \rangle$).

- Hyphenation: o1, o2u1, .po3uč

Com-pet-ing Pat-terns in Lan-guage En-gi-neer-ing and Com-puter Type-set-ting

Definition (matrix representation of the data)

Let us have $m \times n$ matrix $W = w_{ij}$ of data that describe m objects with n binary attributes $P_1, P_2, \ldots, P_n$ (unary predicates). Either P_j or $\neg P_j$ holds. *Elementary conjunction* is a conjuction of literals P_j, $1 \leq j \leq n$, where every predicate appears once at most. Similarly, *Elementary disjunction*. We say that the object i fulfills elementary conjunction Φ if the formula exactly describes the attributes in line i of W. We say that Φ holds for W if Φ holds for all objects (lines in W). We say that formula Φ is *p-truth* if Φ holds for at least $100p\%$ of objects, $p \in \mathbb{R}, 0 < p \leq 1$.

p-truth pattern

Definition (*p*-truth pattern *a*)

Let us have *m* hyphenated words represented in matrix *W* as in previous definition. We say that pattern *a* is *p-truth pattern* if it covers at least 100*p*% of applicable word segmentation points.

Competing Patterns and Pattern Levels

```
   h y p h e n a t i o n
l1          1n a
l1              1t i o
l2           n2a t
l2                2i o
l2        h e2n
l3.h y3p h
l4        h e n a4
l5        h e n5a t
  .h0y3p0h0e2n5a4t2i0o0n.
   h y-p h e n-a t i o n
```

In this *example* $\langle A, \leq \rangle$ is $\mathbb{N}$ (natural numbers). There are 5 pattern levels —`l1...l5`. Patterns in *odd* levels are *covering*, in an *even* levels *inhibiting*. Winner pattern is `.h0y3p0h0e2n5a4t2i0o0n.` Pattern `h e n5a t` wins over `n2a t`, thus hyphenation is possible.

Towards Efficient, Correct and *Minimal* Patterns

We want to find a pattern set that is minimal in size and maximal in performance; we have to define these performance measures.

Definition (precision, recall, F-score)

Let $W = (\Sigma \cup \{0,1\})^*$, and P a set of patterns over $\Sigma' \cup$. Let *good*(w,P) is the number of word divisions where *classify*(w,P) covers w, *good*$(W,P) = \sum_{w \in W}$ *good*(w,P). *bad*(w,P) is the number of word divisions where *classify*(w,P) classifies word division that is not in w, *bad*$(W,P) = \sum_{w \in W}$ *bad*(w,P). *missed*(w,P) is the number of word divisions where *classify*(w,P) fails to classify word division that is in w, *missed*$(W,P) = \sum_{w \in W}$ *missed*(w,P).

Measures

The definition of the measures is then as follows:

Definition (Precision, Recall)

$$precision(W, P) = \frac{good(W, P)}{good(W, P) + bad(W, P)} \quad (1)$$

$$recall(W, P) = \frac{good(W, P)}{good(W, P) + missed(W, P)} \quad (2)$$

The precision and recall scores can be combined into a single measure, known as the *F-score* [Manning 1999]:

Definition (F-score)

$$F(W, P) = \frac{2 \times precision(W, P) \times recall(W, P)}{precision(W, P) + recall(W, P)} \quad (3)$$

Full Coverage ⟶ Compression, and Competing Patterns as Data Structure

An F-score reaches its maximum when both precision and recall is maximal; in the case $F(W, P) = 1$ all information about word division is compressed into the pattern base P.

Definition (lossless compression, cross validation)

If $F(W, P) = 1$ we say that we *losslessly compressed* W into P. We can test performance of P on an unseen word list W' to measure the generalization properties of pattern set P—in the machine learning community, the term *cross validation* is used.

Towards Minimalistic Patterns

There are many pattern sets P that losslessly compress (cover) W; one straightforward solution is having just one pattern for every word $w \in W$ by putting dot symbol around the word with division points marked by 1. Such a pattern set P is a *feasible solution*. But we want to obtain minimal pattern set. Minimality can be measured by the number of patterns, by the number of characters in patterns, or by the space the patterns occupy when stored in some data structure. Even if we take the simplest measure by counting the patterns, and try to find a *minimal* set of patterns that cover W, we will show how hard the task is. To formulate it more precisely, we need to define:

Definition (minimum set cover problem)

An instance of set cover problem is finite set X and a family $\mathcal{F}$ of subsets of X, such that $X = \bigcup_{S \in \mathcal{F}} S$. The problem is to find a set $C \subseteq \mathcal{F}$ of *minimal size* which covers X, i.e. $X = \bigcup_{S \in C} S$.

Minimum Set Cover Problem

The minimum set cover problem (MSCP) is known to be in the class of NPO problems (optimization problems analogical to NP decision problems), [Ausiello 1999]. A variant of MSCP, in which the subsets have positive weights and the objective is to minimize the sum of the weights in a set cover, is also NPO. Weighted version of minimum set cover problem is approximable within $1 + ln|X|$ as shown by Chvatal in 1979.

Main Result

Theorem (pattern minimization problems)

Let W be a set of words with one division only. Problem of finding minimal number of patterns P that losslessly compress W is equivalent to the (weighted) minimum set cover problem.

Proof.

For every subset $C \in W$ there exists at least one feasible solution P_C such that P_C covers C and does not cover any word in $W - C$, e.g., pattern set $\{.c.| \ c \in C\}$. Between all such feasible solutions we choose a canonical representative P'_C —a set which is smallest by some measure (e.g., number of patterns, or number of characters in the pattern set). We now have a one to one correspondence between all pattern sets that cover exactly C represented by P'_C and C. Thus we showed that a pattern coverage minimization problem is equivalent to the weighted minimum set cover [Chvatal 1979] in NPO class. □

Competing Patterns

We have shown that even a pattern covering problem without competition is already NPO. When trying to cover W by competing patterns, complicated interactions may arise—we need some approximation of the optimal solution.

① Competing patterns extend the power of finite state transducer (FST) somewhat like adding the "not" operator to regular expressions.

② Instead of storing full FST, we make patterns that embody the same information in an even more compact manner (compression).

③ Collecting patterns matching a given word can be done in *linear* time, using a *trie* data structure for pattern storage.

Standard Czech Hyphenation Generation

level	length	param	% correct	% wrong	# patterns	size
1	1–3	1 5 1	95.43	6.84	+2,261	
2	1–3	1 5 1	95.84	1.17	+1,051	
3	2–5	1 3 1	99.69	1.24	+3,255	
4	2–5	1 3 1	99.63	0.09	+1,672	40 kB

Methodology for Competing Patterns Development

The methodology consists of several parts:

stratification – for repetitive pattern generation, it is practical to have a stratified word list with 'information bearing' samples only;

bootstrapping – input data (word list with marked hyphenation points) preparation;

goal-driven threshold setting heuristics – the quality of generated patterns depends on many parameters that have to be set in advance;

data filtering by threshold setting heuristics – we can filter out 'dangerous' data—data that are hard to learn for manual inspection.

Stratified Sampling

A large body of information can be comprehended reasonably well
by studying more or less random portions of the data.
The technical term for this approach is *stratified sampling*.
Donald Knuth, 1991

As word lists from which patterns are generated are rather big (5,000,000 for Czech morphology or hyphenation, even more for other tasks such as POS tagging), they may be stratified. Stratification means that from 'equivalent' words only one or a small number of representatives are chosen for the pattern generation process.

Bootstrapping

The road to wisdom? Well it's plain and simple to express:
Err and err and err again but less and less and less. Piet Hein, 1966

Developing patterns is usually an iterative process. One starts with hand-written patterns, uses them on input word list, sees the results, makes the correction, generates new patterns, etc. This technique succeeded in accelerating the pattern development process by the order of magnitude. We usually do not start from scratch, but use some previously collected data (e.g., word list).

Pattern Generation

An important feature of a learning machine is that its teacher will often be very largely ignorant of quite what is going on inside, although he may still be able to some extent to predict his pupil's behavior.
Alan Turing, 1950

A generation process can be parametrised by several parameters (thresholds) for every level. Parameters could be tuned (heuristics) so that virtually all hyphenation points are covered, with no misses.

Input wordlist can be *loslessly* compressed into patterns.

Size optimal pattern generation is NPO problem even without competition.

Summary of Results

① [Formal definition of competing patterns.] We have described and developed a new approach to language engineering based on the theory of covering and inhibiting patterns.

② [New approaches to competing pattern generation.] We have verified plausibility and usefulness of bootstrapping and stratification techniques for machine learning techniques of pattern generation process. We have related our new techniques to those used so far—with the new approach, the results improve significantly.

③ [Properties of pattern generation process.] We have shown that reaching size-optimality of pattern generation process is an NPO problem; however, it is possible to achieve full data recall and precision on the given data with the heuristics presented.

Summary of Results (cont.)

④ [New approach to Thai text segmentation problem.] We have shown that an algorithm using competing patterns learnt from segmented Thai text returns better results than current methods for this task.

⑤ [Thai segmentation patterns.] New patterns for Thai segmentation problem were generated from data in the ORCHID corpus.

⑥ [New Czech and Slovak hyphenation patterns.] The new hyphenation patterns for Czech and Slovak give much better performace than the previous ones, and are in practical use in distributions of text processing systems ranging from TEX, SCRIBUS, OPENOFFICE.ORG to Microsoft Word.

Summary of Results (cont.)

⑦ [New patterns for specific tasks.] Patterns for specific tasks demanded in the areas of computer typesetting and NLP were developed—phonetic hyphenation, universal syllabic hyphenation, and the possibility of using context-sensitive patterns for disambiguation tasks were shown.

⑧ [Foundation for new pattern generation algorithms.] Redesign of a program for pattern generation in of OPATGEN in an object oriented manner allows easy experiment with new pattern generation heuristics.

⑨ [Usage of the methodology for partial morphological disambiguation.]
We have shown that the methodology can be used for partial disambiguation tasks. Experiments showed performance for the partial morphological disambiguation of Czech.

From 97% to 100%: Some Linguistics Needed

"What is best for the final application, i.e. breaking paragraphs into lines: near zero misses or occasional errors for seldomly used words."

Czech *narlval* 'narwhal' and *nalrval* 'gathered by tearing, plucked'; *podlrobit* 'subjugate, to bring under one's domination' and *poldrobit* 'to crumble'; *olblít* 'to vomit up' and *obllít* 'to pour around'

Danish *trælkvinden* 'the wood lady' and *træklvinden* 'the draught'; *kulplet* 'verse' and *kupllet* 'domed'

From 97% to 100%: Some Linguistics needed (cont.)

Dutch *kwartlslagen* 'quarter turns' and *kwartsllagen* 'quartz layers'; *golspel* 'the game of Go' and *goslpel* 'certain type of music'; *rotsltempel* 'rock temple' and *rotlstempel* 'damned stamp'; *dijlkramp* 'cramp in the thighs' and *dijklramp* 'dike catastrophe'; *verlste* 'farthest' and *verslte* 'most fresh'.

English *reclord* (noun) *relcord* (verb) or even *record* (adjective).

German *Staublecken* 'dusty eck' and *Staulbecken* 'traffic jam in the valley'; *Wachlstube* 'guard room' and *Wachsltube* 'wax tube'; *Betltuch* and *Bettltuch*.

Fortunately, number of these homonyms is far below 1% and for such a small number of possible hyphenation points it is not worth to do full semantical analysis.

Practical applications of the developed methods?

① Hyphenation modules in various text and language processing tools: DTP systems (TEX, InDesign, Scribus), text processors (OpenOffice, Word), FOP processors (XEP,...).

② Segmentation of texts (for machine translation, for further processing (Thai), SMS).

③ Various types of disambiguation (context dependent).

④ Contextual dependent ligatures [šéflékař], Fraktur s, Arabic hamza (e.g. in OpenType).

⑤ Compound word hyphenation (Dutch, German).

Application of the competing patterns method for the hyphenation data compression?

Information about segmentation of 5.000.000 Czech words (of average length 9 characters) compressed into 10.000 patterns (stored in *packed trie* in some 50 kB of RAM). In addition, one gets the information in constant time (linear with respect to the word length).

Other data can be encoded in this way.

Mission to find solution that is *time and space efficient* and *precise* is *completed*.

Questions?

THE END

Journals


Petr Sojka and Pavel Ševeček. 1995.
Hyphenation in TEX—Quo Vadis?
TUGboat, 16(3):280–289.


Petr Sojka. 1995.
Notes on Compound Word Hyphenation in TEX.
TUGboat, 16(3):290–297.


Pavel Smrž and Petr Sojka. 1997.
Word Hy-phen-a-tion by Neural Networks.
Neural Network World, 7:687–695.

Journals (cont.)

Petr Sojka. 1999.
Hyphenation on Demand.
TUGboat, 20(3):241–247.


Petr Sojka. 2000.
Competing Patterns for Language Engineering.
In Sojka et al. (Eds.): Proceedings of TSD 2000, Springer-Verlag, LNCS 1902, pages 157–162.


David Antoš and Petr Sojka. 2001.
Pattern Generation Revisited.
In Simon Pepping, editor, Proceedings of the 16th European TEX Conference, Kerkrade, 2001, pages 7–17, Kerkrade, The Netherlands, September. NTG.


Petr Sojka and David Antoš. 2003.
Context Sensitive Pattern Based Segmentation: A Thai Challenge.
pages 65–72, EACL 2003, Budapest, April.