Finite State Methods for Hyphenation

Gosse Bouma

Alfa-informatica, Rijksuniversiteit Groningen

1 Introduction

We present a two-stage finite state method for accurate hyphenation of Dutch words. The base system is a transducer which inserts hyphens between syllables, using the maximal onset principle. Compound words require that morpheme boundaries are marked as hyphenation points. This requirement often conflicts with the maximal onset principle. We use transformation-based learning to resolve such conflicts and report on some practical experiments where the result of TBL is compiled into a cascade of transducers.

2 Hyphenation rules for Dutch

Hyphenation is the process of inserting markers indicating potential hyphenation points in the orthographic representation of a word. Hyphenation can be done using a word list or using hyphenation rules (possibly in combination with a word list to account for exceptional cases). The advantage of a rule-based approach is that it can deal with unknown words as well.

In Dutch, the *core rule of hyphenation* is that hyphens must be inserted between orthographic strings corresponding to syllables. If a string can be segmented into syllable strings in several ways, the segmentation satisfying the *maximal onset principle* (i.e. where onsets are maximal) provides the correct hyphenation:

(1) alfabet → √al-fa-bet, *al-fab-et, *alf-a-bet, *alf-ab-et

Syllabification as studied in Optimality Theory (Prince and Smolensky, 1995; Karttunen, 1998) is concerned with segmenting strings of phonemes, whereas hyphenation operates on orthographic representations. The distinction becomes relevant where the two diverge. In a word such as *aarde* (*earth*), a long vowel '*a*' is followed by a non-empty coda. In these cases, it is represented by two characters ('*aa*') in Dutch. Second, in words such as *appel* (*apple*) a single consonant '*p*' is preceded by a short vowel and followed by a syllable with an empty onset. In those cases, the orthographic form contains a doubled consonant character and the hyphenation point is between the two consonants.

Hyphenation of compound words follows morpheme structure, i.e. hyphens are placed between morphemes, and the individual morphemes are segmented into syllable strings according to the core rule. The word *aardappel* (*potato*, lit. *earth-apple*) is hyphenated as *aard-ap-pel*, and not as *aar-dap-pel* as the core rule would suggest.

Hyphenation of Dutch words is well-studied (Brandt Corstius, 1978). Daelemans and van den Bosch (1992) and Vosse (1994) report hyphenation accuracies (i.e. the percentage of correctly identifed hyphenation points) of 98.2% and 97.8% using Memory-based learning and a HMM technique, respectively.

3 Finite state hyphenation

Hyphenation of Dutch words can be achieved by the composition of two finite-state transducers, the first identifying nuclei, and the second inserting hyphens before maximal onsets:[1]

```
(2)  macro(hyphenate, mark_nucleus o maximize_onset )
(3)  macro(mark_nucleus, replace([[]:@, identity(nucleus), []:@],[],[]))
(4)  macro(insert_hyphens, replace([]:-,[@,consonant *], [onset ^, @]))
```

The definition of `replace(Target,LeftContext,RightContext)` (Karttunen, 1995; Gerdemann and van Noord, 1999) states that strings in the domain of `Target` occurring between `LeftContext` and `RightContext` are replaced by strings in the range of `Target`. Replace performs left-most, longest match, replacement, i.e. it operates as if moving through the string from left to right, at each point identifying the longest possible replacement target. Longest match implies that `mark_nucleus` will mark `aarde` as `@aa@rd@e@` and not as `@a@@a@rd@e@`, eventhough both `a` and `aa` qualify as nuclei. Left-to-right operation is essential in the definition of `insert_hyphens`, as it ensures that hyphens are inserted in a left-most fashion, thus implicitly ensuring that onsets are maximal. The word `alfabet` is transduced into `@a@lf@a@b@e@t` by `mark_nucleus` and hyphenated as `@a@l-f@a@-b@e@t`, which corresponds to the correct hyphenation `al-fa-bet` after removing the nucleus markers.

Note that `insert_hyphens` does not require the preceding left-context to consist of a well-formed coda. Some loan-words with codas that are exceptional for Dutch orthography (`ck` in a word such as `checklist`) can therefore be hyphenated correctly. At the same time, it does not change the hyphenation of words with regular codas.

On a list of almost 300.000 words extracted from Celex (Baayen, Piepenbrock, and van Rijn, 1993), the `hyphenate`-transducer (in combination with a few minor rules dealing with the fact that some nuclei cannot be followed by an empty coda and with the irregular hyphenation of words containing the character '`x`' or the '`qu`' grapheme) achieves a a word accuracy of 86% and a hyphenation accuracy of 94.4%.

4 Improving accuracy using TBL

Transformation-based learning (TBL) (Brill, 1995; Lager, 1999) can be used to improve the accuracy of the system outlined above. Given a word-list hyphenated by the base system, aligned with the correct hyphenation patterns, TBL will attempt to induce rules which correct the maximal number of errors while introducing a minimum of new errors. Learning proceeds until no rules can be found which correct a sufficient number of errors.

As the base system implements the maximal onset principle, but has no way of recognizing morpheme boundaries, errors typically occur where a morpheme boundary conflicts with the maximal onset principle. This suggests that TBL should be able to learn rules which shift a hyphen one position to the right. Other potential, but less frequent, error corrections involve shifting a hyphen two positions rightward, one position leftward, or removing or inserting a hyphen. As recognition of nuclei is fairly accurate, the latter two situations are rare. Hyphenation patterns were aligned as follows:

word	a	a	r	d	a	p	p	e	l
system				1			1		
correct				2			1		

Each character preceded by a hyphen in the system output is marked with a '1'. The corresponding positions in the correct output are marked with a '1' if there is a hyphen in the same position,

[1]We use the FSA Utilities (van Noord, 1997) notation for regular expressions. Concatenation is expressed using Prolog list notation, the empty list [ ] represents the empty string, ':' is the pair operator, '*' is the kleene star operator, ' ^ ' indicates optionality, and 'o' represents composition.

	Initial	Intermediate	Final
Initial Hyphenation Accuracy	92.2	95.3	97.3
Final Hyphenation Accuracy	98.6	99.3	99.6
Number of Induced Rules	748	446	185
Number of Hyphens in Test Set	28,720	34,552	23,002

Table 1: Learning hyphenation of the first, last, and intermediate hyphens

Initial	Intermediate	Final
i-s → is-	i-st → is-t	i-st → is-t
ve-r → ver-	ing-s → ings-	a-st → as-t
-th → t-h	a-st → as-t	e-ste → es-te
a-f → af-	ee-? → ee?-	u-st → us-t
o-n → on-	u-st → us-t	y-stee → ys-tee

Table 2: First five rules for initial, intermediate and final hyphen

with a '2' or '3' if there is a hyphen one or two positions further to the right, with '-1' if there is a hyphen one position further to the left, and with a '0' if there is no corresponding hyphen in the correct output. Other conceivable corrections are those where a hyphen has to be shifted two positions leftward, or where a missing hyphen has to be inserted. These cases are rare (occurring in approximately 0.1% of the data), however, and we did not include them in the training or test data.[2] The task of TBL is to learn in which contexts a '1' has to be corrected into a '2' (thus implicitly registering an instruction to shift a hyphen one position to the right), etc.

Rule templates for TBL were provided which allow a window of maximally 5 characters to be used. The training data consisted of the Celex word-list, which was hyphenated using the base system, and for which the correct hyphenation is given by Celex. Using 90% of the data for training and 10% for testing leads to an amount of data that is unmanageable for TBL. Training on smaller subsets of the data did not lead to satisfactory improvements in accuracy. Therefore, we decided to split the material in three parts. Belz (2000) observes that the distribution of syllables is not equal for all positions within a word. That is, initial syllables will relatively often consist of a derivational prefix, while final syllables will relatively often be inflection endings or derivational suffixes. This suggests that learning can profit from a setup which considers separately the first and second syllable string (and thus only the first hyphen), the penultimate and last syllable (and thus only the last hyphen), and the intermediate syllables and hyphens.

The results of TBL for the three datasets is given in table 1. Given the high frequency of inflectional suffixes following or surrounding the final hyphen, it is not surprising that this hyphenation point can be predicted with high accuracy, and that the system needs relatively few rules to achieve this accuracy. The fact that the initial hyphen is harder to predict than the intermediate hyphens is unexpected. At the moment, we have no explanation for this fact.

Table 2 shows that for the initial position the system learns rules, among others, which refer specifically to the prefixes `ver-`, `af-`, `on-`. Note also the TBL proposes to shift a hyphen following `ee` over the next character ('?' represents an arbitrary character). This suggests that eventhough Dutch spelling in principle allows `ee` to be followed by an empty coda, in practice this situation is exceptional. The rule `y-stee` → `ys-tee` learns the correct hyphenation of the morpheme *systeem*.

The weighted average of the accuracies for initial, intermediate, and final hyphens is 99.1%. If we take this as an indication of what the accuracy of a system combining the three rule sets could be, this suggests a reduction in error rate with respect to the previously published best results

[2]In particular, the inclusion of information that a hyphen has to be inserted, would require all chacters not preceded by a hyphen to be associated with a '0' in the system and correct row of the data representation. This leads to such an increase in the number of facts to be represented that we would no longer be able to use 90% of the available data for training.

of 50%. It is only an estimate of the actual accuracy, because some of the available data (0.1%) could not be aligned using the alignment method described above, and because training of initial and final hyphens assumes that the second and penultimate hyphen have a correct coda and onset, respectively. A proper test requires all rules to be applied in sequence on a test set hyphenated initially by the core system. Attempts to do this are described below.

5 Compilation of TBL rules to FS transducers

TBL rules can be interpreted as finite state transducers (Roche and Schabes, 1997). A finite state transducer implementing the base system and the result of TBL can be conceptualized as follows:

```
macro(hyphenate_tbl,  hyphenate
                    o mark_initial o apply_tbl(initial)
                    o mark_final o apply_tbl(final)
                    o mark_medial o apply_tbl(medial) ).
```

Here, `hyphenate` provides the base hyphenation, `mark_initial` identifies the first hyphen, `apply_tbl(initial)` applies the initial hyphenation rules learned by TBL, etc. In practice, such a setup is not feasible, however. One problem is the large number of rules that has to be included. Another problem is the fact that `mark_final` and `mark_medial` must be able to identify the *last* hyphen in a string. While it is possible to do this deterministically, in practice this gives rise to large automata. We therefore experimented with a set-up where the various components are connected by means of a UNIX pipe.

Rules learned by TBL should be applied to the data in the order in which they are learned. A cascade of TBL rules therefore corresponds to the composition of the automata corresponding with the individual rules. A single rule corresponds to a transducer which typically interchanges characters and hyphens. However, as we code mismatches between system and correct hyphens using numbers, TBL actually learns rules for changing numbers. Two approaches therefore suggest themselves for translating rules in regular expressions. Given a rule `i-s` $\rightarrow$ `is-`, the regular expression in (5) captures its meaning directly. Alternatively, we may assume that hyphens are actually represented as '1's initially in the input, apply the transducer corresponding to the regular expression in (6), and only perform the correspondig shifting of the hyphen in a final processing step (after all rules have applied that is), defined by regular expressions such as (7). The latter approach turns out to be computationally more efficient.

(5) `replace([-,s]:[s,-], [i], [])`

(6) `replace(1:2, [i], [s])`

(7) `replace([ 2:[], id(a..z), []:- ],[],[])`

We experimented with including a fixed number of rules from each of the three rule sets in the corresponding transducer. The effect on hyphenation accuracy of each individual rule set, as well as for the pipe-line of transducers, is given in table 3. As the rule sets apply to different parts of the input string, the reduction in error rate for the accuracy of the combined system in general is the sum of the gains achieved by the separate components. The best accuracy of the combined system achieves a reduction in error rate of 60% with respect to the base system. Yet, its accuracy is lower than the weighted average of the accuracies computed during TBL. We suspect that this is due to the way the data was aligned for TBL (assuming correct hyphenation of the surrounding strings), not because we have failed to include all rules. In fact, using more than half of the rules in each set seems to have little effect on accuracy.

References

Baayen, R. H., R. Piepenbrock, and H. van Rijn. 1993. *The CELEX Lexical Database (CD-ROM).* University of Pennsylvania, Philadelphia, PA: Linguistic Data Consortium.

Initial		Medial		Final		Combined Acc	
N	acc	N	acc	N	acc	*base*	94.48
100	96.07	100	95.20	100	95.24		97.47
200	96.29	200	95.30	185	95.29		97.84
400	96.53	400	95.45	185	95.29		98.24

Table 3: Hyphenation accuracy using N TBL rules for correcting the first, last, and medial hyphens on full words.

Belz, Anja. 2000. Multi-syllable phonotactic modelling. In Lauri Karttunen Jason Eisner and Alain Thériault, editors, *Proceedings of SIGPHON 2000: Finite-State Phonology.* Luxembourg.

Brandt Corstius, H. 1978. *Computer-taalkunde.* Muiderberg: Coutinho.

Brill, Eric. 1995. Transformation-based error-driven learning and natural language processing: A case study in part-of-speech tagging. *Computational Linguistics*, 21:543–566.

Daelemans, Walter and Antal van den Bosch. 1992. Generalization performance of backpropagation learning on a syllabification task. In *Connectionism and Natural Language Processing. Proceedings Third Twente Workshop on Language Technology*, pages 27–38.

Gerdemann, Dale and Gertjan van Noord. 1999. Transducers from rewrite rules with backreferences. In *Proceedings of the Ninth Conference of the European Chapter of the Association for Computational Linguistics*, pages 126–133, Bergen.

Karttunen, Lauri. 1995. The replace operator. In *33th Annual Meeting of the Association for Computational Linguistics*, pages 16–23, Boston, Massachusetts.

Karttunen, Lauri. 1998. The proper treatment of optimality in computational phonology. In Lauri Karttunen, editor, *FSMNLP'98: International Workshop on Finite State Methods in Natural Language Processing.* Association for Computational Linguistics, Somerset, New Jersey, pages 1–12.

Lager, Torbjörn. 1999. The μ-TBL System: Logic programming tools for transformation-based learning. In *Proceedings of the Third International Workshop on Computational Natural Language Learning (CoNLL'99)*, pages 33–42, Bergen.

Prince, A. and P. Smolensky. 1995. *Optimality Theory: Constraint Interaction in Generative Grammar.* Cambridge, MA: MIT Press.

Roche, Emmanuel and Yves Schabes. 1997. Deterministic part-of-speech tagging with finite-state transducers. In Emmanuel Roche and Yves Schabes, editors, *Finite state language processing.* MIT Press, Cambridge, Mass., pages 205–239.

van Noord, Gertjan. 1997. FSA Utilities: A toolbox to manipulate finite-state automata. In Darrell Raymond, Derick Wood, and Sheng Yu, editors, *Automata Implementation.* Springer Verlag. Lecture Notes in Computer Science 1260.

Vosse, Theo. 1994. *The Word Connection.* Ph.D. thesis, Rijksuniversiteit Leiden.