Conditional Random Fields for Word Hyphenation

Nikolaos Trogkanis
Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093-0404
tronikos@gmail.com

Charles Elkan
Computer Science and Engineering
University of California, San Diego
La Jolla, California 92093-0404
elkan@cs.ucsd.edu

Abstract

Finding allowable places in words to insert hyphens is an important practical problem. The algorithm that is used most often nowadays has remained essentially unchanged for 25 years. This method is the TEX hyphenation algorithm of Knuth and Liang. We present here a hyphenation method that is clearly more accurate. The new method is an application of conditional random fields. We create new training sets for English and Dutch from the CELEX European lexical resource, and achieve error rates for English of less than 0.1% for correctly allowed hyphens, and less than 0.01% for Dutch. Experiments show that both the Knuth/Liang method and a leading current commercial alternative have error rates several times higher for both languages.

1 Introduction

The task that we investigate is learning to split words into parts that are conventionally agreed to be individual written units. In many languages, it is acceptable to separate these units with hyphens, but it is not acceptable to split words arbitrarily. Another way of stating the task is that we want to learn to predict for each letter in a word whether or not it is permissible for the letter to be followed by a hyphen. This means that we tag each letter with either 1, for hyphen allowed following this letter, or 0, for hyphen not allowed after this letter.

The hyphenation task is also called orthographic syllabification (Bartlett et al., 2008). It is an important issue in real-world text processing, as described further in Section 2 below. It is also useful as a preprocessing step to improve letter-to-phoneme conversion, and more generally for text-to-speech conversion. In the well-known NETtalk system, for example, syllable boundaries are an input to the neural network in addition to letter identities (Sejnowski and Rosenberg, 1988). Of course, orthographic syllabification is not a fundamental scientific problem in linguistics. Nevertheless, it is a difficult engineering task that is worth studying for both practical and intellectual reasons.

The goal in performing hyphenation is to predict a sequence of 0/1 values as a function of a sequence of input characters. This sequential prediction task is significantly different from a standard (non-sequential) supervised learning task. There are at least three important differences that make sequence prediction difficult. First, the set of all possible sequences of labels is an exponentially large set of possible outputs. Second, different inputs have different lengths, so it is not obvious how to represent every input by a vector of the same fixed length, as is almost universal in supervised learning. Third and most important, too much information is lost if we learn a traditional classifier that makes a prediction for each letter separately. Even if the traditional classifier is a function of the whole input sequence, this remains true. In order to achieve high accuracy, correlations between neighboring predicted labels must be taken into account.

Learning to predict a sequence of output labels, given a sequence of input data items, is an instance of a structured learning problem. In general, structured learning means learning to predict outputs that have internal structure. This structure can be modeled; to achieve high predictive accuracy, when there are dependencies between parts of an output, it must be modeled. Research on structured learning has been highly successful, with sequence classification as its most important and successful subfield, and with conditional random fields (CRFs) as the most influential approach to learning sequence classifiers. In the present paper,

Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics, pages 366–374,
Uppsala, Sweden, 11-16 July 2010.

we show that CRFs can achieve extremely good performance on the hyphenation task.

2 History of automated hyphenation

The earliest software for automatic hyphenation was implemented for RCA 301 computers, and used by the *Palm Beach Post-Tribune* and *Los Angeles Times* newspapers in 1962. These were two different systems. The Florida system had a dictionary of 30,000 words; words not in the dictionary were hyphenated after the third, fifth, or seventh letter, because the authors observed that this was correct for many words. The California system (Friedlander, 1968) used a collection of rules based on the rules stated in a version of Webster's dictionary. The earliest hyphenation software for a language other than English may have been a rule-based program for Finnish first used in 1964 (Jarvi, 2009).

The first formal description of an algorithm for hyphenation was in a patent application submitted in 1964 (Damerau, 1964). Other early publications include (Ocker, 1971; Huyser, 1976). The hyphenation algorithm that is by far the most widely used now is due to Liang (Liang, 1983). Although this method is well-known now as the one used in TEX and its derivatives, the first version of TEX used a different, simpler method. Liang's method was used also in `troff` and `groff`, which were the main original competitors of TEX, and is part of many contemporary software products, supposedly including Microsoft Word. Any major improvement over Liang's method is therefore of considerable practical and commercial importance.

Over the years, various machine learning methods have been applied to the hyphenation task. However, none have achieved high accuracy. One paper that presents three different learning methods is (van den Bosch et al., 1995). The lowest per-letter test error rate reported is about 2%. Neural networks have been used, but also without great success. For example, the authors of (Kristensen and Langmyhr, 2001) found that the TEX method is a better choice for hyphenating Norwegian.

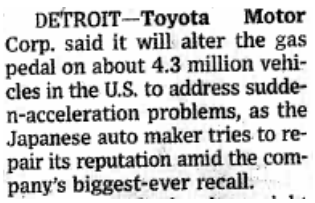
The highest accuracy achieved until now for the hyphenation task is by (Bartlett et al., 2008), who use a large-margin structured learning approach. Our work is similar, but was done fully independently. The accuracy we achieve is slightly higher: word-level accuracy of 96.33% compared to their 95.65% for English. Moreover, (Bartlett et al., 2008) do not address the issue that false positive hyphens are worse mistakes than false negative hyphens, which we address below. Also, they report that training on 14,000 examples requires about an hour, compared to 6.2 minutes for our method on 65,828 words. Perhaps more important for large-scale publishing applications, our system is about six times faster at syllabifying new text. The speed comparison is fair because the computer we use is slightly slower than the one they used.

Methods inspired by nonstatistical natural language processing research have also been proposed for the hyphenation task, in particular (Bouma, 2003; Tsalidis et al., 2004; Woestenburg, 2006; Haralambous, 2006). However, the methods for Dutch presented in (Bouma, 2003) were found to have worse performance than TEX. Moreover, our experimental results below show that the commercial software of (Woestenburg, 2006) allows hyphens incorrectly almost three times more often than TEX.

In general, a dictionary based approach has zero errors for words in the dictionary, but fails to work for words not included in it. A rule-based approach requires an expert to define manually the rules and exceptions for each language, which is laborious work. Furthermore, for languages such as English where hyphenation does not systematically follow general rules, such an approach does not have good results. A pattern-learning approach, like that of TEX, infers patterns from a training list of hyphenated words, and then uses these patterns to hyphenate text. Although useful patterns are learned automatically, both the TEX learning algorithm and the learned patterns must be hand-tuned to perform well (Liang, 1983).

Liang's method is implemented in a program named `PATGEN`, which takes as input a training set of hyphenated words, and outputs a collection of interacting hyphenation patterns. The standard pattern collections are named `hyphen.tex` for American English, `ukhyphen.tex` for British English, and `nehyph96.tex` for Dutch. The precise details of how different versions of TEX and LATEX use these pattern collections to do hyphenation in practice are unclear. At a minimum, current variants of TEX improve hyphenation accuracy by disallowing hyphens in the first and last two or three letters of every word, regardless of what the `PATGEN` patterns recommend.

Despite the success of Liang's method, incorrect hyphenations remain an issue with TEX and its current variants and competitors. For instance, incorrect hyphenations are common in the *Wall Street Journal*, which has the highest circulation of any newspaper in the U.S. An example is the hyphenation of the word "sudden" in this extract:


DETROIT—Toyota Motor Corp. said it will alter the gas pedal on about 4.3 million vehicles in the U.S. to address sudden-acceleration problems, as the Japanese auto maker tries to repair its reputation amid the company's biggest-ever recall.

It is the case that most hyphenation mistakes in the *Wall Street Journal* and other media are for proper nouns such as "Netflix" that do not appear in standard dictionaries, or in compound words such as "sudden-acceleration" above.

3 Conditional random fields

A linear-chain conditional random field (Lafferty et al., 2001) is a way to use a log-linear model for the sequence prediction task. We use the bar notation for sequences, so $\bar{x}$ means a sequence of variable length. Specifically, let $\bar{x}$ be a sequence of n letters and let $\bar{y}$ be a corresponding sequence of n tags. Define the log-linear model

$$p(\bar{y}|\bar{x};w) = \frac{1}{Z(\bar{x},w)} \exp \sum_j w_j F_j(\bar{x},\bar{y}).$$

The index j ranges over a large set of feature-functions. Each such function F_j is a sum along the output sequence for $i = 1$ to $i = n$:

$$F_j(\bar{x},\bar{y}) = \sum_{i=1}^{n} f_j(y_{i-1}, y_i, \bar{x}, i)$$

where each function f_j is a 0/1 indicator function that picks out specific values for neighboring tags y_{i-1} and y_i and a particular substring of $\bar{x}$. The denominator $Z(\bar{x},w)$ is a normalizing constant:

$$Z(\bar{x},w) = \sum_{\bar{y}} \exp \sum_j w_j F_j(\bar{x},\bar{y})$$

where the outer sum is over all possible labelings $\bar{y}$ of the input sequence $\bar{x}$. Training a CRF means finding a weight vector w that gives the best possible predictions

$$\bar{y}^* = \arg\max_{\bar{y}} p(\bar{y}|\bar{x};w)$$

for each training example $\bar{x}$.

The software we use as an implementation of conditional random fields is named CRF++ (Kudo, 2007). This implementation offers fast training since it uses L-BFGS (Nocedal and Wright, 1999), a state-of-the-art quasi-Newton method for large optimization problems. We adopt the default parameter settings of CRF++, so no development set or tuning set is needed in our work.

We define indicator functions f_j that depend on substrings of the input word, and on whether or not a hyphen is legal after the current and/or the previous letter. The substrings are of length 2 to 5, covering up to 4 letters to the left and right of the current letter. From all possible indicator functions we use only those that involve a substring that occurs at least once in the training data.

As an example, consider the word `hy-phen-ate`. For this word $\bar{x} = hyphenate$ and $\bar{y} = 010001000$. Suppose $i = 3$ so p is the current letter. Then exactly two functions f_j that depend on substrings of length 2 have value 1:

$$I(y_{i-1} = 1 \text{ and } y_i = 0 \text{ and } x_2x_3 = yp) = 1,$$
$$I(y_{i-1} = 1 \text{ and } y_i = 0 \text{ and } x_3x_4 = ph) = 1.$$

All other similar functions have value 0:

$$I(y_{i-1} = 1 \text{ and } y_i = 1 \text{ and } x_2x_3 = yp) = 0,$$
$$I(y_{i-1} = 1 \text{ and } y_i = 0 \text{ and } x_2x_3 = yq) = 0,$$

and so on. There are similar indicator functions for substrings up to length 5. In total, 2,916,942 different indicator functions involve a substring that appears at least once in the English dataset.

One finding of our work is that it is preferable to use a large number of low-level features, that is patterns of specific letters, rather than a smaller number of higher-level features such as consonant-vowel patterns. This finding is consistent with an emerging general lesson about many natural language processing tasks: the best performance is achieved with models that are discriminative, that are trained on as large a dataset as possible, and that have a very large number of parameters but are regularized (Halevy et al., 2009).

When evaluating the performance of a hyphenation algorithm, one should not just count how many words are hyphenated in exactly the same way as in a reference dictionary. One should also measure separately how many legal hyphens are actually predicted, versus how many predicted hyphens are in fact not legal. Errors of the second type are false positives. For any hyphenation

method, a false positive hyphen is a more serious mistake than a false negative hyphen, i.e. a hyphen allowed by the lexicon that the method fails to identify. The standard Viterbi algorithm for making predictions from a trained CRF is not tuned to minimize false positives. To address this difficulty, we use the forward-backward algorithm (Sha and Pereira, 2003; Culotta and McCallum, 2004) to estimate separately for each position the probability of a hyphen at that position. Then, we only allow a hyphen if this probability is over a high threshold such as 0.9.

Each hyphenation corresponds to one path through a graph that defines all 2^{k-1} hyphenations that are possible for a word of length k. The overall probability of a hyphen at any given location is the sum of the weights of all paths that do have a hyphen at this position, divided by the sum of the weights of all paths. The forward-backward algorithm uses the sum operator to compute the weight of a set of paths, instead of the max operator to compute the weight of a single highest-weight path. In order to compute the weight of all paths that contain a hyphen at a specific location, weight 0 is assigned to all paths that do not have a hyphen at this location.

4 Dataset creation

We start with the lexicon for English published by the Dutch Centre for Lexical Information at `http://www.mpi.nl/world/celex`. We download all English word forms with legal hyphenation points indicated by hyphens. These include plurals of nouns, conjugated forms of verbs, and compound words such as "off-line". We separate the components of compound words and phrases, leading to 204,466 words, of which 68,744 are unique. In order to eliminate abbreviations and proper names which may not be English, we remove all words that are not fully lower-case. In particular, we exclude words that contain capital letters, apostrophes, and/or periods. This leaves 66,001 words.

Among these words, 86 have two different hyphenations, and one has three hyphenations. For most of the 86 words with alternative hyphenations, these alternatives exist because different meanings of the words have different pronunciations, and the different pronunciations have different boundaries between syllables. This fact implies that no algorithm that operates on words in isolation can be a complete solution for the hyphenation task.[1]

We exclude the few words that have two or more different hyphenations from the dataset. Finally, we obtain 65,828 spellings. These have 550,290 letters and 111,228 hyphens, so the average is 8.36 letters and 1.69 hyphens per word. Informal inspection suggests that the 65,828 spellings contain no mistakes. However, about 1000 words follow British as opposed to American spelling.

The Dutch dataset of 293,681 words is created following the same procedure as for the English dataset, except that all entries from CELEX that are compound words containing dashes are discarded instead of being split into parts, since many of these are not in fact Dutch words.[2]

5 Experimental design

We use ten-fold cross validation for the experiments. In order to measure accuracy, we compute the confusion matrix for each method, and from this we compute error rates. We report both word-level and letter-level error rates. The word-level error rate is the fraction of words on which a method makes at least one mistake. The letter-level error rate is the fraction of letters for which the method predicts incorrectly whether or not a hyphen is legal after this letter. Table 1 explains the terminology that we use in presenting our results. Precision, recall, and F1 can be computed easily from the reported confusion matrices.

As an implementation of Liang's method we use TEX Hyphenator in Java software available at `http://texhyphj.sourceforge.net`. We evaluate this algorithm on our entire English and Dutch datasets using the appropriate language pattern files, and not allowing a hyphen to be placed between the first `lefthyphenmin` and last `righthyphenmin` letters of each word. For

[1] The single word with more than two alternative hyphenations is "invalid" whose three hyphenations are `in-va-lid in-val-id` and `in-valid`. Interestingly, the Merriam–Webster online dictionary also gives three hyphenations for this word, but not the same ones: `in-va-lid in-val-id invalid`. The American Heritage dictionary agrees with Merriam-Webster. The disagreement illustrates that there is a certain irreducible ambiguity or subjectivity concerning the correctness of hyphenations.

[2] Our English and Dutch datasets are available for other researchers and practitioners to use at `http://www.cs.ucsd.edu/users/elkan/hyphenation`. Previously a similar but smaller CELEX-based English dataset was created by (van den Bosch et al., 1995), but that dataset is not available online currently.

Abbr	Name	Description
TP	true positives	#hyphens predicted correctly
FP	false positives	#hyphens predicted incorrectly
TN	true negatives	#hyphens correctly not predicted
FN	false negatives	#hyphens failed to be predicted
owe	overall word-level errors	#words with at least one FP or FN
swe	serious word-level errors	#words with at least one FP
ower	overall word-level error rate	owe / (total #words)
swer	serious word-level error rate	swe / (total #words)
oler	overall letter-level error rate	(FP+FN) / (TP+TN+FP+FN)
sler	serious letter-level error rate	FP / (TP+TN+FP+FN)

Table 1: Alternative measures of accuracy. TP, TN, FP, and FN are computed by summing over the test sets of each fold of cross-validation.

English the default values are 2 and 3 respectively. For Dutch the default values are both 2.

The hyphenation patterns used by TeXHyphenator, which are those currently used by essentially all variants of TEX, may not be optimal for our new English and Dutch datasets. Therefore, we also do experiments with the PATGEN tool (Liang and Breitenlohner, 2008). These are learning experiments so we also use ten-fold cross validation in the same way as with CRF++. Specifically, we create a pattern file from 90% of the dataset using PATGEN, and then hyphenate the remaining 10% of the dataset using Liang's algorithm and the learned pattern file.

The PATGEN tool has many user-settable parameters. As is the case with many machine learning methods, no strong guidance is available for choosing values for these parameters. For English we use the parameters reported in (Liang, 1983). For Dutch we use the parameters reported in (Tutelaers, 1999). Preliminary informal experiments found that these parameters work better than alternatives. We also disallow hyphens in the first two letters of every word, and the last three letters for English, or last two for Dutch.

We also evaluate the TALO commercial software (Woestenburg, 2006). We know of one other commercial hyphenation application, which is named Dashes.[3] Unfortunately we do not have access to it for evaluation. We also cannot do a precise comparison with the method of (Bartlett et al., 2008). We do know that their training set was also derived from CELEX, and their maximum reported accuracy is slightly lower. Specifically, for English our word-level accuracy ("ower") is 96.33% while their best ("WA") is 95.65%.

[3] http://www.circlenoetics.com/dashes.aspx

6 Experimental results

In Table 2 and Table 3 we report the performance of the different methods on the English and Dutch datasets respectively. Figure 1 shows how the error rate is affected by increasing the CRF probability threshold for each language.

Figure 1 shows confidence intervals for the error rates. These are computed as follows. For a single Bernoulli trial the mean is p and the variance is $p(1-p)$. If N such trials are taken, then the observed success rate $f = S/N$ is a random variable with mean p and variance $p(1-p)/N$. For large N, the distribution of the random variable f approaches the normal distribution. Hence we can derive a confidence interval for p using the formula

$$Pr[-z \leq \frac{f-p}{\sqrt{p(1-p)/N}} \leq z] = c$$

where for a 95% confidence interval, i.e. for $c = 0.95$, we set $z = 1.96$. All differences between rows in Table 2 are significant, with one exception: the serious error rates for PATGEN and TALO are not statistically significantly different. A similar conclusion applies to Table 3.

For the English language, the CRF using the Viterbi path has overall error rate of 0.84%, compared to 6.81% for the TEX algorithm using American English patterns, which is eight times worse. However, the serious error rate for the CRF is less good: 0.41% compared to 0.24%. This weakness is remedied by predicting that a hyphen is allowable only if it has high probability. Figure 1 shows that the CRF can use a probability threshold up to 0.99, and still have lower overall error rate than the TEX algorithm. Fixing the probability threshold at 0.99, the CRF serious error rate is 0.04% (224 false positives) compared to 0.24% (1343 false positives) for the TEX algorithm.

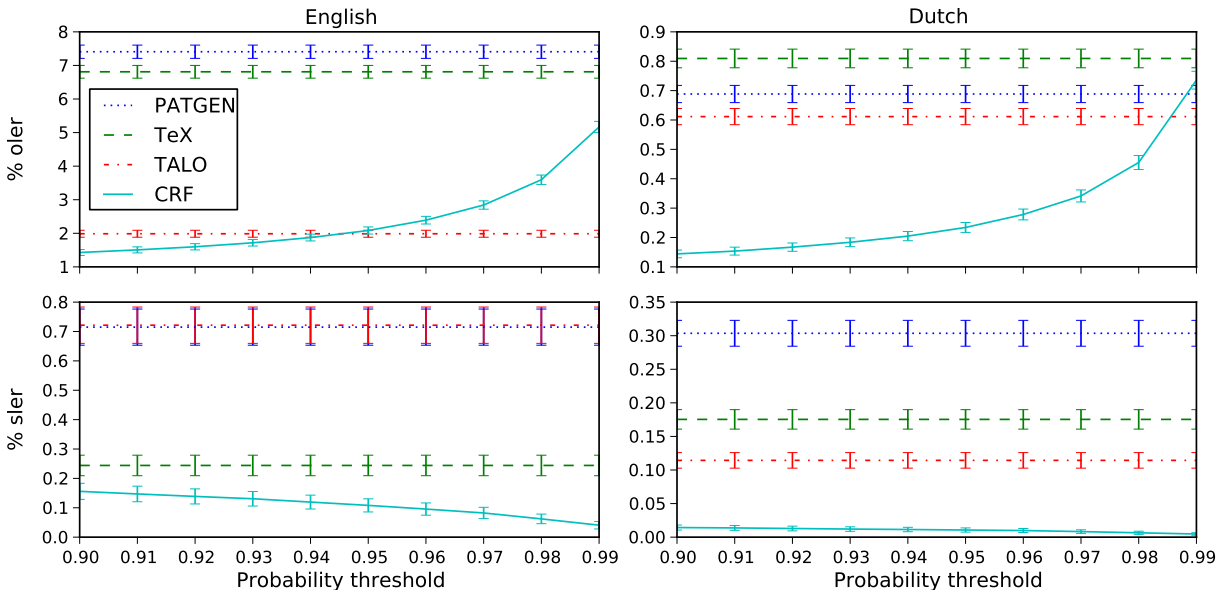


Figure 1: Total letter-level error rate and serious letter-level error rate for different values of threshold for the CRF. The left subfigures are for the English dataset, while the right ones are for the Dutch dataset. The TALO and PATGEN lines are almost identical in the bottom left subfigure.

Method	TP	FP	TN	FN	owe	swe	% ower	% swer	% oler	% sler
Place no hyphen	0	0	439062	111228	57541	0	87.41	0.00	20.21	0.00
TEX (hyphen.tex)	75093	1343	437719	36135	30337	1311	46.09	1.99	6.81	0.24
TEX (ukhyphen.tex)	70307	13872	425190	40921	31337	11794	47.60	17.92	9.96	2.52
TALO	104266	3970	435092	6962	7213	3766	10.96	5.72	1.99	0.72
PATGEN	74397	3934	435128	36831	32348	3803	49.14	5.78	7.41	0.71
CRF	**108859**	2253	436809	**2369**	**2413**	2080	**3.67**	3.16	**0.84**	0.41
CRF (threshold = 0.99)	83021	**224**	**438838**	28207	22992	**221**	34.93	**0.34**	5.17	**0.04**

Table 2: Performance on the English dataset.

Method	TP	FP	TN	FN	owe	swe	% ower	% swer	% oler	% sler
Place no hyphen	0	0	2438913	742965	287484	0	97.89	0.00	23.35	0.00
TEX (nehyph96.tex)	722789	5580	2433333	20176	20730	5476	7.06	1.86	0.81	0.18
TALO	727145	3638	2435275	15820	16346	3596	5.57	1.22	0.61	0.11
PATGEN	730720	9660	2429253	12245	20318	9609	6.92	3.27	0.69	0.30
CRF	**741796**	1230	2437683	**1169**	**1443**	1207	**0.49**	0.41	**0.08**	0.04
CRF (threshold = 0.99)	719710	**149**	**2438764**	23255	22067	**146**	7.51	**0.05**	0.74	**0.00**

Table 3: Performance on the Dutch dataset.

Method	TP	FP	TN	FN	owe	swe	% ower	% swer	% oler	% sler
PATGEN	70357	6763	432299	40871	35013	6389	53.19	9.71	8.66	1.23
CRF	**104487**	6518	432544	**6741**	**6527**	5842	**9.92**	8.87	**2.41**	1.18
CRF (threshold = 0.99)	75651	**654**	**438408**	35577	27620	**625**	41.96	**0.95**	6.58	**0.12**

Table 4: Performance on the English dataset (10-fold cross validation dividing by stem).

Method	TP	FP	TN	FN	owe	swe	% ower	% swer	% oler	% sler
PATGEN	727306	13204	2425709	15659	25363	13030	8.64	4.44	0.91	0.41
CRF	**740331**	2670	2436243	**2634**	**3066**	2630	**1.04**	0.90	**0.17**	0.08
CRF (threshold = 0.99)	716596	**383**	**2438530**	26369	24934	**373**	8.49	**0.13**	0.84	**0.01**

Table 5: Performance on the Dutch dataset (10-fold cross validation dividing by stem).

Method	TP	FP	TN	FN	owe	swe	% ower	% swer	% oler	% sler
TEX	2711	43	21433	1420	1325	43	33.13	1.08	5.71	0.17
PATGEN	2590	113	21363	1541	1466	113	36.65	2.83	6.46	0.44
CRF	**4129**	2	21474	**2**	**2**	2	**0.05**	0.05	**0.02**	0.01
CRF (threshold = 0.9)	4065	**0**	**21476**	66	63	**0**	1.58	**0.00**	0.26	**0.00**

Table 6: Performance on the 4000 most frequent English words.

For the English language, TALO yields overall error rate 1.99% with serious error rate 0.72%, so the standard CRF using the Viterbi path is better on both measures. The dominance of the CRF method can be increased further by using a probability threshold. Figure 1 shows that the CRF can use a probability threshold up to 0.94, and still have lower overall error rate than TALO. Using this threshold, the CRF serious error rate is 0.12% (657 false positives) compared to 0.72% (3970 false positives) for TALO.

For the Dutch language, the standard CRF using the Viterbi path has overall error rate 0.08%, compared to 0.81% for the TEX algorithm. The serious error rate for the CRF is 0.04% while for TEX it is 0.18%. Figure 1 shows that any probability threshold for the CRF of 0.99 or below yields lower error rates than the TEX algorithm. Using the threshold 0.99, the CRF has serious error rate only 0.005%.

For the Dutch language, the TALO method has overall error rate 0.61%. The serious error rate for TALO is 0.11%. The CRF dominance can again be increased via a high probability threshold. Figure 1 shows that this threshold can range up to 0.98, and still give lower overall error rate than TALO. Using the 0.98 threshold, the CRF has serious error rate 0.006% (206 false positives); in comparison the serious error rate of TALO is 0.11% (3638 false positives).

For both languages, PATGEN has higher serious letter-level and word-level error rates than TEX using the existing pattern files. This is expected since the pattern collections included in TEX distributions have been tuned over the years to minimize objectionable errors. The difference is especially pronounced for American English, for which the standard pattern collection has been manually improved over more than two decades by many people (Beeton, 2002). Initially, Liang optimized this pattern collection extensively by upweighting the most common words and by iteratively adding exception words found by testing the algorithm against a large dictionary from an unknown publisher (Liang, 1983).

One can tune PATGEN to yield either better overall error rate, or better serious error rate, but not both simultaneously, compared to the TEX algorithm using the existing pattern files for both languages. For the English dataset, if we use Liang's parameters for PATGEN as reported in (Sojka and Sevecek, 1995), we obtain overall error rate of 6.05% and serious error rate of 0.85%. It is possible that the specific patterns used in TEX implementations today have been tuned by hand to be better than anything the PATGEN software is capable of.

7 Additional experiments

This section presents empirical results following two experimental designs that are less standard, but that may be more appropriate for the hyphenation task.

First, the experimental design used above has an issue shared by many CELEX-based tagging or transduction evaluations: words are randomly divided into training and test sets without being grouped by stem. This means that a method can get credit for hyphenating "accents" correctly, when "accent" appears in the training data. Therefore, we do further experiments where the folds for evaluation are divided by stem, and not by word; that is, all versions of a base form of a word appear in the same fold. Stemming uses the English and Dutch versions of the Porter stemmer (Porter, 1980).[4] The 65,828 English words in our dictionary produce 27,100 unique stems, while the 293,681 Dutch words produce 169,693 unique stems. The results of these experiments are shown in Tables 4 and 5.

The main evaluation in the previous section is based on a list of unique words, which means that in the results each word is equally weighted. Because cross validation is applied, errors are always measured on testing subsets that are disjoint from the corresponding training subsets. Hence, the accuracy achieved can be interpreted as the performance expected when hyphenating unknown words, i.e. rare future words.

However, in real documents common words appear repeatedly. Therefore, the second less-standard experimental design for which we report results restricts attention to the most common English words. Specifically, we consider the top 4000 words that make up about three quarters of all word appearances in the American National Corpus, which consists of 18,300,430 words from written texts of all genres.[5] From the 4,471 most

[4] Available at http://snowball.tartarus.org/. A preferable alternative might be to use the information about the lemmas of words available directly in CELEX.

[5] Available at americannationalcorpus.org/SecondRelease/data/ANC-written-count.txt

frequent words in this list, if we omit the words not in our dataset of 89,019 hyphenated English words from CELEX, we get 4,000 words. The words that are omitted are proper names, contractions, incomplete words containing apostrophes, and abbreviations such as DNA. These 4,000 most frequent words make up 74.93% of the whole corpus.

We evaluate the following methods on the 4000 words: Liang's method using the American patterns file `hyphen.tex`, Liang's method using the patterns derived from `PATGEN` when trained on the whole English dataset, our CRF trained on the whole English dataset, and the same CRF with a probability threshold of 0.9. Results are shown in Table 6. In summary, TEX and `PATGEN` make serious errors on 43 and 113 of the 4000 words, respectively. With a threshold of 0.9, the CRF approach makes zero serious errors on these words.

8 Timings

Table 7 shows the speed of the alternative methods for the English dataset. The column "Features/Patterns" in the table reports the number of feature-functions used for the CRF, or the number of patterns used for the TEX algorithm. Overall, the CRF approach is about ten times slower than the TEX algorithm, but its performance is still acceptable on a standard personal computer. All experiments use a machine having a Pentium 4 CPU at 3.20GHz and 2GB memory. Moreover, informal experiments show that CRF training would be about eight times faster if we used CRFSGD rather than CRF++ (Bottou, 2008).

From a theoretical perspective, both methods have almost-constant time complexity per word if they are implemented using appropriate data structures. In TEX, hyphenation patterns are stored in a data structure that is a variant of a trie. The CRF software uses other data structures and optimizations that allow a word to be hyphenated in time that is almost independent of the number of feature-functions used.

9 Conclusions

Finding allowable places in words to insert hyphens is a real-world problem that is still not fully solved in practice. The main contribution of this paper is a hyphenation method that is clearly more accurate than the currently used Knuth/Liang method. The new method is an application of CRFs, which are a major advance of recent years in machine learning. We hope that the method proposed here is adopted in practice, since the number of serious errors that it makes is about a sixfold improvement over what is currently in use. A second contribution of this paper is to provide training sets for hyphenation in English and Dutch, so other researchers can, we hope, soon invent even more accurate methods. A third contribution of our work is a demonstration that current CRF methods can be used straightforwardly for an important application and outperform state-of-the-art commercial and open-source software; we hope that this demonstration accelerates the widespread use of CRFs.

Method	Features/ Patterns	Training time (s)	Testing time (s)	Speed (ms/word)
CRF	2916942	372.67	25.386	0.386
TEX (us)	4447	-	2.749	0.042
`PATGEN`	4488	33.402	2.889	0.044
TALO	-	-	8.400	0.128

Table 7: Timings for the English dataset (training and testing on the whole dataset that consists of 65,828 words).

References

Susan Bartlett, Grzegorz Kondrak, and Colin Cherry. 2008. Automatic syllabification with structured SVMs for letter-to-phoneme conversion. *Proceedings of ACL-08: HLT*, pages 568–576.

Barbara Beeton. 2002. Hyphenation exception log. *TUGboat*, 23(3).

Léon Bottou. 2008. Stochastic gradient CRF software CRFSGD. Available at `http://leon.bottou.org/projects/sgd`.

Gosse Bouma. 2003. Finite state methods for hyphenation. *Natural Language Engineering*, 9(1):5–20, March.

Aron Culotta and Andrew McCallum. 2004. Confidence Estimation for Information Extraction. In Susan Dumais, Daniel Marcu, and Salim Roukos, editors, *HLT-NAACL 2004: Short Papers*, pages 109–112, Boston, Massachusetts, USA, May. Association for Computational Linguistics.

Fred J. Damerau. 1964. Automatic Hyphenation Scheme. U.S. patent 3537076 filed June 17, 1964, issued October 1970.

Gordon D. Friedlander. 1968. Automation comes to the printing and publishing industry. *IEEE Spectrum*, 5:48–62, April.

Alon Halevy, Peter Norvig, and Fernando Pereira. 2009. The Unreasonable Effectiveness of Data. *IEEE Intelligent Systems*, 24(2):8–12.

Yannis Haralambous. 2006. New hyphenation techniques in Ω_2. *TUGboat*, 27:98–103.

Steven L. Huyser. 1976. AUTO-MA-TIC WORD DIVI-SION. *SIGDOC Asterisk Journal of Computer Documentation*, 3(5):9–10.

Timo Jarvi. 2009. Computerized Typesetting and Other New Applications in a Publishing House. In *History of Nordic Computing 2*, pages 230–237. Springer.

Terje Kristensen and Dag Langmyhr. 2001. Two regimes of computer hyphenation–a comparison. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, volume 2, pages 1532–1535.

Taku Kudo, 2007. *CRF++: Yet Another CRF Toolkit*. Version 0.5 available at `http://crfpp.sourceforge.net/`.

John Lafferty, Andrew McCallum, and Fernando Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18th International Conference on Machine Learning (ICML)*, pages 282–289.

Franklin M. Liang and Peter Breitenlohner, 2008. *PATtern GENeration Program for the TEX82 Hyphenator*. Electronic documentation of PATGEN program version 2.3 from web2c distribution on CTAN, retrieved 2008.

Franklin M. Liang. 1983. *Word Hy-phen-a-tion by Com-put-er*. Ph.D. thesis, Stanford University.

Jorge Nocedal and Stephen J. Wright. 1999. Limited memory BFGS. In *Numerical Optimization*, pages 222–247. Springer.

Wolfgang A. Ocker. 1971. A program to hyphenate English words. *IEEE Transactions on Engineering, Writing and Speech*, 14(2):53–59, June.

Martin Porter. 1980. An algorithm for suffix stripping. *Program*, 14(3):130–137.

Terrence J. Sejnowski and Charles R. Rosenberg, 1988. *NETtalk: A parallel network that learns to read aloud*, pages 661–672. MIT Press, Cambridge, MA, USA.

Fei Sha and Fernando Pereira. 2003. Shallow parsing with conditional random fields. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 134–141.

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

Christos Tsalidis, Giorgos Orphanos, Anna Iordanidou, and Aristides Vagelatos. 2004. Proofing Tools Technology at Neurosoft S.A. *ArXiv Computer Science e-prints*, (cs/0408059), August.

P.T.H. Tutelaers, 1999. *Afbreken in TEX, hoe werkt dat nou?* Available at `ftp://ftp.tue.nl/pub/tex/afbreken/`.

Antal van den Bosch, Ton Weijters, Jaap Van Den Herik, and Walter Daelemans. 1995. The profit of learning exceptions. In *Proceedings of the 5th Belgian-Dutch Conference on Machine Learning (BENELEARN)*, pages 118–126.

Jaap C. Woestenburg, 2006. **TALO's Language Technology*, November. Available at `http://www.talo.nl/talo/download/documents/Language_Book.pdf`.