Automatic Syllabification with Structured SVMs for Letter-To-Phoneme Conversion

Susan Bartlett[†] **Grzegorz Kondrak**[†] **Colin Cherry**[‡]

[†]Department of Computing Science
University of Alberta
Edmonton, AB, T6G 2E8, Canada
{susan,kondrak}@cs.ualberta.ca

[‡]Microsoft Research
One Microsoft Way
Redmond, WA, 98052
colinc@microsoft.com

Abstract

We present the first English syllabification system to improve the accuracy of letter-to-phoneme conversion. We propose a novel discriminative approach to automatic syllabification based on structured SVMs. In comparison with a state-of-the-art syllabification system, we reduce the syllabification word error rate for English by 33%. Our approach also performs well on other languages, comparing favorably with published results on German and Dutch.

1 Introduction

Pronouncing an unfamiliar word is a task that is often accomplished by breaking the word down into smaller components. Even small children learning to read are taught to pronounce a word by "sounding out" its parts. Thus, it is not surprising that Letter-to-Phoneme (L2P) systems, which convert orthographic forms of words into sequences of phonemes, can benefit from subdividing the input word into smaller parts, such as syllables or morphemes. Marchand and Damper (2007) report that incorporating oracle syllable boundary information improves the accuracy of their L2P system, but they fail to emulate that result with any of their automatic syllabification methods. Demberg et al. (2007), on the other hand, find that morphological segmentation boosts L2P performance in German, but not in English. To our knowledge, no previous English orthographic syllabification system has been able to actually improve performance on the larger L2P problem.

In this paper, we focus on the task of automatic orthographic syllabification, with the explicit goal of improving L2P accuracy. A syllable is a subdivision of a word, typically consisting of a vowel, called the nucleus, and the consonants preceding and following the vowel, called the onset and the coda, respectively. Although in the strict linguistic sense syllables are phonological rather than orthographic entities, our L2P objective constrains the input to orthographic forms. Syllabification of phonemic representation is in fact an easier task, which we plan to address in a separate publication.

Orthographic syllabification is sometimes referred to as *hyphenation*. Many dictionaries provide hyphenation information for orthographic word forms. These hyphenation schemes are related to, and influenced by, phonemic syllabification. They serve two purposes: to indicate where words may be broken for end-of-line divisions, and to assist the dictionary reader with correct pronunciation (Gove, 1993). Although these purposes are not always consistent with our objective, we show that we can improve L2P conversion by taking advantage of the available hyphenation data. In addition, automatic hyphenation is a legitimate task by itself, which could be utilized in word editors or in synthesizing new trade names from several concepts.

We present a discriminative approach to orthographic syllabification. We formulate syllabification as a tagging problem, and learn a discriminative tagger from labeled data using a structured support vector machine (SVM) (Tsochantaridis et al., 2004). With this approach, we reduce the error rate for English by 33%, relative to the best existing system. Moreover, we are also able to improve a state-of-the-art L2P system by incorporating our syllabification models. Our method is not language specific; when applied to German and Dutch, our performance is

Proceedings of ACL-08: HLT, pages 568–576,
Columbus, Ohio, USA, June 2008.

comparable with the best existing systems in those languages, even though our system has been developed and tuned on English only.

The paper is structured as follows. After discussing previous computational approaches to the problem (Section 2), we introduce structured SVMs (Section 3), and outline how we apply them to orthographic syllabification (Section 4). We present our experiments and results for the syllabification task in Section 5. In Section 6, we apply our syllabification models to the L2P task. Section 7 concludes.

2 Related Work

Automatic preprocessing of words is desirable because the productive nature of language ensures that no finite lexicon will contain all words. Marchand et al. (2007) show that rule-based methods are relatively ineffective for orthographic syllabification in English. On the other hand, few data-driven syllabification systems currently exist.

Demberg (2006) uses a fourth-order Hidden Markov Model to tackle orthographic syllabification in German. When added to her L2P system, Demberg's orthographic syllabification model effects a one percent absolute improvement in L2P word accuracy.

Bouma (2002) explores syllabification in Dutch. He begins with finite state transducers, which essentially implement a general preference for onsets. Subsequently, he uses transformation-based learning to automatically extract rules that improve his system. Bouma's best system, trained on some 250K examples, achieves 98.17% word accuracy. Daelemans and van den Bosch (1992) implement a backpropagation network for Dutch orthography, but find it is outperformed by less complex look-up table approaches.

Marchand and Damper (2007) investigate the impact of syllabification on the L2P problem in English. Their Syllabification by Analogy (SbA) algorithm is a data-driven, lazy learning approach. For each input word, SbA finds the most similar substrings in a lexicon of syllabified words and then applies these dictionary syllabifications to the input word. Marchand and Damper report 78.1% word accuracy on the NETtalk dataset, which is not good enough to improve their L2P system.

Chen (2003) uses an n-gram model and Viterbi decoder as a syllabifier, and then applies it as a preprocessing step in his maximum-entropy-based English L2P system. He finds that the syllabification pre-processing produces no gains over his baseline system.

Marchand et al. (2007) conduct a more systematic study of existing syllabification approaches. They examine syllabification in both the pronunciation and orthographic domains, comparing their own SbA algorithm with several instance-based learning approaches (Daelemans et al., 1997; van den Bosch, 1997) and rule-based implementations. They find that SbA universally outperforms these other approaches by quite a wide margin.

Syllabification of phonemes, rather than letters, has also been investigated (Müller, 2001; Pearson et al., 2000; Schmid et al., 2007). In this paper, our focus is on orthographic forms. However, as with our approach, some previous work in the phonetic domain has formulated syllabification as a tagging problem.

3 Structured SVMs

A structured support vector machine (SVM) is a large-margin training method that can learn to predict structured outputs, such as tag sequences or parse trees, instead of performing binary classification (Tsochantaridis et al., 2004). We employ a structured SVM that predicts tag sequences, called an SVM Hidden Markov Model, or SVM-HMM. This approach can be considered an HMM because the Viterbi algorithm is used to find the highest scoring tag sequence for a given observation sequence. The scoring model employs a Markov assumption: each tag's score is modified only by the tag that came before it. This approach can be considered an SVM because the model parameters are trained discriminatively to separate correct tag sequences from incorrect ones by as large a margin as possible. In contrast to generative HMMs, the learning process requires labeled training data.

There are a number of good reasons to apply the structured SVM formalism to this problem. We get the benefit of discriminative training, not available in a generative HMM. Furthermore, we can use an arbitrary feature representation that does not require

any conditional independence assumptions. Unlike a traditional SVM, the structured SVM considers complete tag sequences during training, instead of breaking each sequence into a number of training instances.

Training a structured SVM can be viewed as a multi-class classification problem. Each training instance x_i is labeled with a correct tag sequence y_i drawn from a set of possible tag sequences Y_i. As is typical of discriminative approaches, we create a feature vector $\Psi(x, y)$ to represent a candidate y and its relationship to the input x. The learner's task is to weight the features using a vector w so that the correct tag sequence receives more weight than the competing, incorrect sequences:

$$\forall_i \forall_{y \in Y_i, y \neq y_i} \left[\Psi(x_i, y_i) \cdot w > \Psi(x_i, y) \cdot w\right] \quad (1)$$

Given a trained weight vector w, the SVM tags new instances x_i according to:

$$\text{argmax}_{y \in Y_i} \left[\Psi(x_i, y) \cdot w\right] \quad (2)$$

A structured SVM finds a w that satisfies Equation 1, and separates the correct taggings by as large a margin as possible. The argmax in Equation 2 is conducted using the Viterbi algorithm.

Equation 1 is a simplification. In practice, a structured distance term is added to the inequality in Equation 1 so that the required margin is larger for tag sequences that diverge further from the correct sequence. Also, slack variables are employed to allow a trade-off between training accuracy and the complexity of w, via a tunable cost parameter.

For most structured problems, the set of negative sequences in Y_i is exponential in the length of x_i, and the constraints in Equation 1 cannot be explicitly enumerated. The structured SVM solves this problem with an iterative online approach:

1. Collect the most damaging incorrect sequence y according to the current w.
2. Add y to a growing set $\bar{Y}_i$ of incorrect sequences.
3. Find a w that satisfies Equation 1, using the partial $\bar{Y}_i$ sets in place of Y_i.
4. Go to next training example, loop to step 1.

This iterative process is explained in far more detail in (Tsochantaridis et al., 2004).

4 Syllabification with Structured SVMs

In this paper we apply structured SVMs to the syllabification problem. Specifically, we formulate syllabification as a tagging problem and apply the SVM-HMM software package[1] (Altun et al., 2003). We use a linear kernel, and tune the SVM's cost parameter on a development set. The feature representation Ψ consists of emission features, which pair an aspect of x with a single tag from y, and transition features, which count tag pairs occurring in y. With SVM-HMM, the crux of the task is to create a tag scheme and feature set that produce good results. In this section, we discuss several different approaches to tagging for the syllabification task. Subsequently, we outline our emission feature representation. While developing our tagging schemes and feature representation, we used a development set of 5K words held out from our CELEX training data. All results reported in this section are on that set.

4.1 Annotation Methods

We have employed two different approaches to tagging in this research. **Positional tags** capture where a letter occurs within a syllable; **Structural tags** express the role each letter is playing within the syllable.

Positional Tags

The **NB tag** scheme simply labels every letter as either being at a syllable boundary (B), or not (N). Thus, the word *im-mor-al-ly* is tagged ⟨*N B N N B N B N N*⟩, indicating a syllable boundary after each B tag. This binary classification approach to tagging is implicit in several previous implementations (Daelemans and van den Bosch, 1992; Bouma, 2002), and has been done explicitly in both the orthographic (Demberg, 2006) and phoneme domains (van den Bosch, 1997).

A weakness of NB tags is that they encode no knowledge about the length of a syllable. Intuitively, we expect the length of a syllable to be valuable information — most syllables in English contain fewer than four characters. We introduce a tagging scheme that sequentially numbers the *N* tags to impart information about syllable length. Under the **Numbered**

[1] http://svmlight.joachims.org/svm_struct.html

NB tag scheme, *im-mor-al-ly* is annotated as ⟨*N1 B N1 N2 B N1 B N1 N2*⟩. With this tag set, we have effectively introduced a bias in favor of shorter syllables: tags like *N6*, *N7*... are comparatively rare, so the learner will postulate them only when the evidence is particularly compelling.

Structural Tags

Numbered NB tags are more informative than standard NB tags. However, neither annotation system can represent the internal structure of the syllable. This has advantages: tags can be automatically generated from a list of syllabified words without even a passing familiarity with the language. However, a more informative annotation, tied to phonotactics, ought to improve accuracy. Krenn (1997) proposes the **ONC tag** scheme, in which phonemes of a syllable are tagged as an onset, nucleus, or coda. Given these ONC tags, syllable boundaries can easily be generated by applying simple regular expressions.

Unfortunately, it is not as straightforward to generate ONC-tagged training data in the orthographic domain, even with syllabified training data. Silent letters are problematic, and some letters can behave differently depending on their context (in English, consonants such as *m*, *y*, and *l* can act as vowels in certain situations). Thus, it is difficult to generate ONC tags for orthographic forms without at least a cursory knowledge of the language and its principles.

For English, tagging the syllabified training set with ONC tags is performed by the following simple algorithm. In the first stage, all letters from the set $\{a, e, i, o, u\}$ are marked as vowels, while the remaining letters are marked as consonants. Next, we examine all the instances of the letter y. If a *y* is both preceded and followed by a consonant, we mark that instance as a vowel rather than a consonant. In the second stage, the first group of consecutive vowels in each syllable is tagged as nucleus. All letters preceding the nucleus are then tagged as onset, while all letters following the nucleus are tagged as coda.

Our development set experiments suggested that numbering ONC tags increases their performance. Under the **Numbered ONC tag** scheme, the single-syllable word *stealth* is labeled ⟨*O1 O2 N1 N2 C1 C2 C3*⟩.

A disadvantage of Numbered ONC tags is that, unlike positional tags, they do not represent syllable breaks explicitly. Within the ONC framework, we need the conjunction of two tags (such as an N1 tag followed by an O1 tag) to represent the division between syllables. This drawback can be overcome by combining ONC tags and NB tags in a hybrid **Break ONC tag** scheme. Using Break ONC tags, the word *lev-i-ty* is annotated as ⟨*O N CB NB O N*⟩. The ⟨*NB*⟩ tag indicates a letter is both part of the nucleus and before a syllable break, while the ⟨*N*⟩ tag represents a letter that is part of a nucleus but in the middle of a syllable. In this way, we get the best of both worlds: tags that encapsulate information about syllable structure, while also representing syllable breaks explicitly with a single tag.

4.2 Emission Features

SVM-HMM predicts a tag for each letter in a word, so emission features use aspects of the input to help predict the correct tag for a specific letter. Consider the tag for the letter *o* in the word *immorally*. With a traditional HMM, we consider only that it is an *o* being emitted, and assess potential tags based on that single letter. The SVM framework is less restrictive: we can include *o* as an emission feature, but we can also include features indicating that the preceding and following letters are *m* and *r* respectively. In fact, there is no reason to confine ourselves to only one character on either side of the focus letter.

After experimenting with the development set, we decided to include in our feature set a window of eleven characters around the focus character, five on either side. Figure 1 shows that performance gains level off at this point. Special beginning- and end-of-word characters are appended to words so that every letter has five characters before and after. We also experimented with asymmetric context windows, representing more characters after the focus letter than before, but we found that symmetric context windows perform better.

Because our learner is effectively a linear classifier, we need to explicitly represent any important conjunctions of features. For example, the bigram *bl* frequently occurs within a single English syllable, while the bigram *lb* generally straddles two syllables. Similarly, a fourgram like *tion* very often

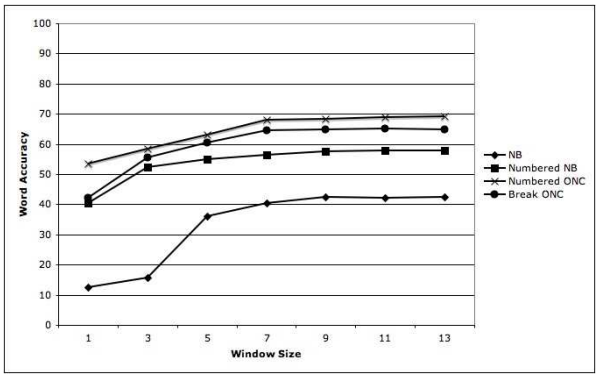

Figure 1: Word accuracy as a function of the window size around the focus character, using unigram features on the development set.

forms a syllable in and of itself. Thus, in addition to the single-letter features outlined above, we also include in our representation any bigrams, trigrams, four-grams, and five-grams that fit inside our context window. As is apparent from Figure 2, we see a substantial improvement by adding bigrams to our feature set. Higher-order n-grams produce increasingly smaller gains.

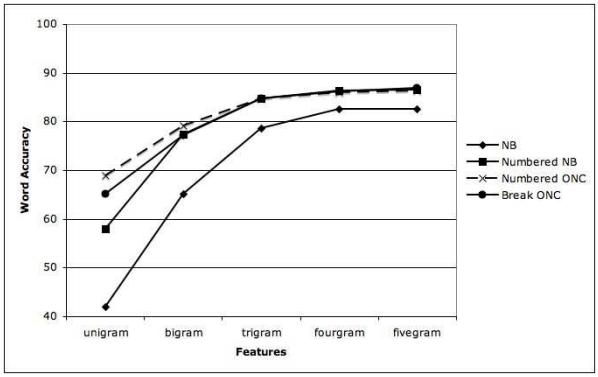

Figure 2: Word accuracy as a function of maximum n-gram size on the development set.

In addition to these primary n-gram features, we experimented with linguistically-derived features. Intuitively, basic linguistic knowledge, such as whether a letter is a consonant or a vowel, should be helpful in determining syllabification. However, our experiments suggested that including features like these has no significant effect on performance. We believe that this is caused by the ability of the SVM to learn such generalizations from the n-gram features alone.

5 Syllabification Experiments

In this section, we will discuss the results of our best emission feature set (five-gram features with a context window of eleven letters) on held-out unseen test sets. We explore several different languages and datasets, and perform a brief error analysis.

5.1 Datasets

Datasets are especially important in syllabification tasks. Dictionaries sometimes disagree on the syllabification of certain words, which makes a gold standard difficult to obtain. Thus, any reported accuracy is only with respect to a given set of data.

In this paper, we report the results of experiments on two datasets: CELEX and NETtalk. We focus mainly on CELEX, which has been developed over a period of years by linguists in the Netherlands. CELEX contains English, German, and Dutch words, and their orthographic syllabifications. We removed all duplicates and multiple-word entries for our experiments. The NETtalk dictionary was originally developed with the L2P task in mind. The syllabification data in NETtalk was created manually in the phoneme domain, and then mapped directly to the letter domain.

NETtalk and CELEX do not provide the same syllabification for every word. There are numerous instances where the two datasets differ in a perfectly reasonable manner (*e.g. for-ging* in NETtalk vs. *forg-ing* in CELEX). However, we argue that NETtalk is a vastly inferior dataset. On a sample of 50 words, NETtalk agrees with Merriam-Webster's syllabifications in only 54% of instances, while CELEX agrees in 94% of cases. Moreover, NETtalk is riddled with truly bizarre syllabifications, such as *be-aver*, *dis-hcloth* and *som-ething*. These syllabifications make generalization very hard, and are likely to complicate the L2P task we ultimately want to accomplish. Because previous work in English primarily used NETtalk, we report our results on both datasets. Nevertheless, we believe NETtalk is unsuitable for building a syllabification model, and that results on CELEX are much more indicative of the efficacy of our (or any other) approach.

At 20K words, NETtalk is much smaller than CELEX. For NETtalk, we randomly divide the data into 13K training examples and 7K test words. We

randomly select a comparably-sized training set for our CELEX experiments (14K), but test on a much larger, 25K set. Recall that 5K training examples were held out as a development set.

5.2 Results

We report the results using two metrics. Word accuracy (WA) measures how many words match the gold standard. Syllable break error rate (SBER) captures the incorrect tags that cause an error in syllabification. Word accuracy is the more demanding metric. We compare our system to Syllabification by Analogy (SbA), the best existing system for English (Marchand and Damper, 2007). For both CELEX and NETtalk, SbA was trained and tested with the same data as our structured SVM approach.

Data Set	Method	WA	SBER
CELEX	NB tags	86.66	2.69
	Numbered NB	89.45	2.51
	Numbered ONC	89.86	2.50
	Break ONC	89.99	2.42
	SbA	84.97	3.96
NETtalk	Numbered NB	81.75	5.01
	SbA	75.56	7.73

Table 1: Syllabification performance in terms of word accuracy and syllable break error percentage.

Table 1 presents the word accuracy and syllable break error rate achieved by each of our tag sets on both the CELEX and NETtalk datasets. Of our four tag sets, NB tags perform noticeably worse. This is an important result because it demonstrates that it is not sufficient to simply model a syllable's boundaries; we must also model a syllable's length or structure to achieve the best results. Given the similarity in word accuracy scores, it is difficult to draw definitive conclusions about the remaining three tags sets, but it does appear that there is an advantage to modeling syllable structure, as both ONC tag sets score better than the best NB set.

All variations of our system outperform SbA on both datasets. Overall, our best tag set lowers the error rate by one-third, relative to SbA's performance. Note that we employ only numbered NB tags for the NETtalk test; we could not apply structural tag schemes to the NETtalk training data because of its bizarre syllabification choices.

Our higher level of accuracy is also achieved more efficiently. Once a model is learned, our system can syllabify 25K words in about a minute, while SbA requires several hours (Marchand, 2007). SVM training times vary depending on the tag set and dataset used, and the number of training examples. On 14K CELEX examples with the ONC tag set, our model trained in about an hour, on a single-processor P4 3.4GHz processor. Training time is, of course, a one-time cost. This makes our approach much more attractive for inclusion in an actual L2P system.

Figure 3 shows our method's learning curve. Even small amounts of data produce adequate performance — with only 2K training examples, word accuracy is already over 75%. Using a 60K training set and testing on a held-out 5K set, we see word accuracies climb to 95.65%.

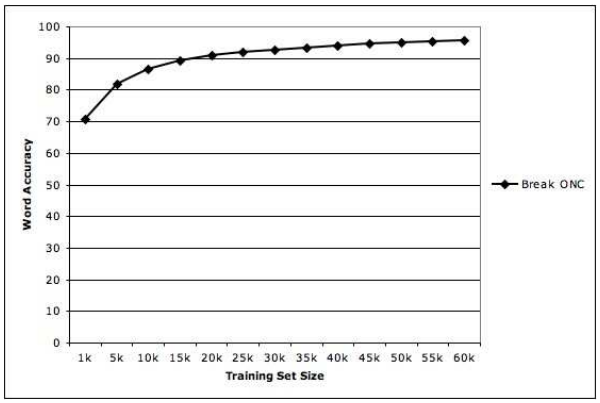

Figure 3: Word accuracy as function of the size of the training data.

5.3 Error Analysis

We believe that the reason for the relatively low performance of unnumbered NB tags is the weakness of the signal coming from NB emission features. With the exception of *q* and *x*, every letter can take on either an N tag or a B tag with almost equal probability. This is not the case with Numbered NB tags. Vowels are much more likely to have N2 or N3 tags (because they so often appear in the middle of a syllable), while consonants take on N1 labels with greater probability.

The numbered NB and ONC systems make many of the same errors, on words that we might expect to

cause difficulty. In particular, both suffer from being unaware of compound nouns and morphological phenomena. All three systems, for example, incorrectly syllabify *hold-o-ver* as *hol-dov-er*. This kind of error is caused by a lack of knowledge of the component words. The three systems also display trouble handling consecutive vowels, as when *co-ad-jutors* is syllabified incorrectly as *coad-ju-tors*. Vowel pairs such as *oa* are not handled consistently in English, and the SVM has trouble predicting the exceptions.

5.4 Other Languages

We take advantage of the language-independence of Numbered NB tags to apply our method to other languages. Without even a cursory knowledge of German or Dutch, we have applied our approach to these two languages.

# Data Points	Dutch	German
$\sim$ 50K	98.20	98.81
$\sim$ 250K	99.45	99.78

Table 2: Syllabification performance in terms of word accuracy percentage.

We have randomly selected two training sets from the German and Dutch portions of CELEX. Our smaller model is trained on $\sim$ 50K words, while our larger model is trained on $\sim$ 250K. Table 2 shows our performance on a 30K test set held out from both training sets. Results from both the small and large models are very good indeed.

Our performance on these language sets is clearly better than our best score for English (compare at 95% with a comparable amount of training data). Syllabification is a more regular process in German and Dutch than it is in English, which allows our system to score higher on those languages.

Our method's word accuracy compares favorably with other methods. Bouma's finite state approach for Dutch achieves 96.49% word accuracy using 50K training points, while we achieve 98.20%. With a larger model, trained on about 250K words, Bouma achieves 98.17% word accuracy, against our 99.45%. Demberg (2006) reports that her HMM approach for German scores 97.87% word accuracy, using a 90/10 training/test split on the CELEX dataset. On the same set, Demberg et al. (2007) obtain 99.28% word accuracy by applying the system of Schmid et al. (2007). Our score using a similar split is 99.78%.

Note that none of these scores are directly comparable, because we did not use the same train-test splits as our competitors, just similar amounts of training and test data. Furthermore, when assembling random train-test splits, it is quite possible that words sharing the same lemma will appear in both the training and test sets. This makes the problem much easier with large training sets, where the chance of this sort of overlap becomes high. Therefore, any large data results may be slightly inflated as a prediction of actual out-of-dictionary performance.

6 L2P Performance

As we stated from the outset, one of our primary motivations for exploring orthographic syllabification is the improvements it can produce in L2P systems. To explore this, we tested our model in conjunction with a recent L2P system that has been shown to predict phonemes with state-of-the-art word accuracy (Jiampojamarn et al., 2007). Using a model derived from training data, this L2P system first divides a word into letter chunks, each containing one or two letters. A local classifier then predicts a number of likely phonemes for each chunk, with confidence values. A phoneme-sequence Markov model is then used to select the most likely sequence from the phonemes proposed by the local classifier.

Syllabification	English	Dutch	German
None	84.67	91.56	90.18
Numbered NB	85.55	92.60	90.59
Break ONC	85.59	N/A	N/A
Dictionary	86.29	93.03	90.57

Table 3: Word accuracy percentage on the letter-to-phoneme task with and without the syllabification information.

To measure the improvement syllabification can effect on the L2P task, the L2P system was trained with syllabified, rather than unsyllabified words. Otherwise, the execution of the L2P system remains unchanged. Data for this experiment is again drawn

from the CELEX dictionary. In Table 3, we report the average word accuracy achieved by the L2P system using 10-fold cross-validation. We report L2P performance without any syllabification information, with perfect dictionary syllabification, and with our small learned models of syllabification. L2P performance with dictionary syllabification represents an approximate upper bound on the contributions of our system.

Our syllabification model improves L2P performance. In English, perfect syllabification produces a relative error reduction of 10.6%, and our model captures over half of the possible improvement, reducing the error rate by 6.0%. To our knowledge, this is the first time a syllabification model has improved L2P performance in English. Previous work includes Marchand and Damper (2007)'s experiments with SbA and the L2P problem on NETtalk. Although perfect syllabification reduces their L2P relative error rate by 18%, they find that their learned model actually increases the error rate. Chen (2003) achieved word accuracy of 91.7% for his L2P system, testing on a different dictionary (Pronlex) with a much larger training set. He does not report word accuracy for his syllabification model. However, his baseline L2P system is not improved by adding a syllabification model.

For Dutch, perfect syllabification reduces the relative L2P error rate by 17.5%; we realize over 70% of the available improvement with our syllabification model, reducing the relative error rate by 12.4%.

In German, perfect syllabification produces only a small reduction of 3.9% in the relative error rate. Experiments show that our learned model actually produces a slightly higher reduction in the relative error rate. This anomaly may be due to errors or inconsistencies in the dictionary syllabifications that are not replicated in the model output. Previously, Demberg (2006) generated statistically significant L2P improvements in German by adding syllabification pre-processing. However, our improvements are coming at a much higher baseline level of word accuracy – 90% versus only 75%.

Our results also provide some evidence that syllabification preprocessing may be more beneficial to L2P than morphological preprocessing. Demberg et al. (2007) report that oracle morphological annotation produces a relative error rate reduction of 3.6%. We achieve a larger decrease at a higher level of accuracy, using an automatic pre-processing technique. This may be because orthographic syllabifications already capture important facts about a word's morphology.

7 Conclusion

We have applied structured SVMs to the syllabification problem, clearly outperforming existing systems. In English, we have demonstrated a 33% relative reduction in error rate with respect to the state of the art. We used this improved syllabification to increase the letter-to-phoneme accuracy of an existing L2P system, producing a system with 85.5% word accuracy, and recovering more than half of the potential improvement available from perfect syllabification. This is the first time automatic syllabification has been shown to improve English L2P. Furthermore, we have demonstrated the language-independence of our system by producing competitive orthographic syllabification solutions for both Dutch and German, achieving word syllabification accuracies of 98% and 99% respectively. These learned syllabification models also improve accuracy for German and Dutch letter-to-phoneme conversion.

In future work on this task, we plan to explore adding morphological features to the SVM, in an effort to overcome errors in compound words and inflectional forms. We would like to experiment with performing L2P and syllabification jointly, rather than using syllabification as a pre-processing step for L2P. We are also working on applying our method to phonetic syllabification.

Acknowledgements

Many thanks to Sittichai Jiampojamarn for his help with the L2P experiments, and to Yannick Marchand for providing the SbA results.

This research was supported by the Natural Sciences and Engineering Research Council of Canada and the Alberta Informatics Circle of Research Excellence.

References

Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. 2003. Hidden Markov support vector ma-

chines. *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 3–10.

Susan Bartlett. 2007. Discriminative approach to automatic syllabification. Master's thesis, Department of Computing Science, University of Alberta.

Gosse Bouma. 2002. Finite state methods for hyphenation. *Natural Language Engineering*, 1:1–16.

Stanley Chen. 2003. Conditional and joint models for grapheme-to-phoneme conversion. *Proceedings of the 8th European Conference on Speech Communication and Technology (Eurospeech)*.

Walter Daelemans and Antal van den Bosch. 1992. Generalization performance of backpropagation learning on a syllabification task. *Proceedings of the 3rd Twente Workshop on Language Technology*, pages 27–38.

Walter Daelemans, Antal van den Bosch, and Ton Weijters. 1997. IGTree: Using trees for compression and classification in lazy learning algorithms. *Artificial Intelligence Review*, pages 407–423.

Vera Demberg, Helmust Schmid, and Gregor Möhler. 2007. Phonological constraints and morphological preprocessing for grapheme-to-phoneme conversion. *Proceedings of the 45th Annual Meeting of the Association of Computational Linguistics (ACL)*.

Vera Demberg. 2006. Letter-to-phoneme conversion for a German text-to-speech system. Master's thesis, University of Stuttgart.

Philip Babcock Gove, editor. 1993. *Webster's Third New International Dictionary of the English Language, Unabridged*. Merriam-Webster Inc.

Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. 2007. Applying many-to-many alignments and hidden Markov models to letter-to-phoneme conversion. *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics HLT-NAACL*, pages 372–379.

Brigitte Krenn. 1997. Tagging syllables. *Proceedings of Eurospeech*, pages 991–994.

Yannick Marchand and Robert Damper. 2007. Can syllabification improve pronunciation by analogy of English? *Natural Language Engineering*, 13(1):1–24.

Yannick Marchand, Connie Adsett, and Robert Damper. 2007. Evaluation of automatic syllabification algorithms for English. In *Proceedings of the 6th International Speech Communication Association (ISCA) Workshop on Speech Synthesis*.

Yannick Marchand. 2007. Personal correspondence.

Karin Müller. 2001. Automatic detection of syllable boundaries combining the advantages of treebank and bracketed corpora training. *Proceedings on the 39th Meeting of the Association for Computational Linguistics (ACL)*, pages 410–417.

Steve Pearson, Roland Kuhn, Steven Fincke, and Nick Kibre. 2000. Automatic methods for lexical stress assignment and syllabification. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, pages 423–426.

Helmut Schmid, Bernd Möbius, and Julia Weidenkaff. 2007. Tagging syllable boundaries with joint N-gram models. *Proceedings of Interspeech*.

Ioannis Tsochantaridis, Thomas Hofmann, Thorsten Joachims, and Yasemin Altun. 2004. Support vector machine learning for interdependent and structured output spaces. *Proceedings of the 21st International Conference on Machine Learning (ICML)*, pages 823–830.

Antal van den Bosch. 1997. *Learning to pronounce written words: a study in inductive language learning*. Ph.D. thesis, Universiteit Maastricht.