"Oh, Westley, I didn't mean any of it! Not one single syllabub."

Westley knew that she meant syllable, since syllabub was a dessert, but he also knew an apology when he heard it, so he said,

"I know you didn't mean it. Not a single syllabub."

—William Goldman, *The Princess Bride*

University of Alberta

A DISCRIMINATIVE APPROACH TO AUTOMATIC SYLLABIFICATION

by

Susan Elizabeth Bartlett ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of **Master of Science**.

Department of Computing Science

Edmonton, Alberta

Fall 2007

Your file Votre référence
ISBN: 978-0-494-33197-2
Our file Notre référence
ISBN: 978-0-494-33197-2

Abstract

Syllabification is the process of dividing a word into its constituent syllables. Syllabification holds considerable theoretical interest and has a number of practical applications. In addition to providing an in-depth survey of existing syllabification systems, this thesis uses discriminative models to automatically syllabify both letters and phonemes. Syllabification is formulated as a labelling task so that a classifier can learn mappings from characters to labels. The final system is a language-independent implementation for both letters and phonemes that betters current state-of-the-art systems in several languages. When the resulting syllabification models are incorporated into a letter-to-phoneme system, accuracy improves on that task. Several rule-based techniques for syllabification of phonemes are also presented.

Acknowledgements

It has been a pleasure to work with the entire NLP research group over the past two years. Especial thanks to Tee Jiampojamarn for all his help on letter-to-phoneme experiments, and to Colin Cherry for his seemingly-endless enthusiasm and helpfulness. And, of course, many thanks to Dr. Greg Kondrak for supervising this work and guiding me through my master's degree.

Thanks to my parents for twenty-six years (and counting) of love and support. Thanks to Sam for, among other things, keeping me well-fed. And thanks to The Bridge Club, because you guys are the best.

I am very grateful for the financial support provided by the Natural Sciences and Engineering Research Council of Canada, the Informatics Circle of Research Excellence, and the University of Alberta.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Problem Description

Although a syllable can consist of just a single phoneme, and many words are monosyllabic, a syllable is generally defined as a linguistic unit that is larger than a phoneme and smaller than a whole word [Crystal, 2003]. Syllabification is the process of dividing a word into its constituent syllables; it is a long-standing topic of debate in the field of linguistics. More recently, syllabification has become prevalent in natural language processing. Because syllables are an important determiner of pronunciation, we are interested in computational approaches to syllabification both for their theoretical implication and their practical applications.

One such possible application is text-to-speech (TTS) systems. While actual implementations vary, TTS systems must have, at minimum, three components [Damper, 2001]. Syllabification can play a role in all three modules:

1. A letter-to-phoneme (L2P) module converts orthographic forms (typically ASCII input) into an abstract phonetic representation.

2. A prosody module determines the variations in pitch, loudness, tempo, and rhythm that should be applied to the phonemes within a word.

3. A synthesis module produces the actual sounds, either generating them directly from the system's internal representation, or by concatenating together sounds from a database.

Most commonly, prosody units include syllabification modules in some way. A number of TTS systems have incorporated syllabification, including the Bell Labs multilingual TTS system [Sproat, 1998] and the Panasonic Speech Technology Lab's implementation [Pearson *et al.*, 2000]. Stress patterns are typically assigned at the syllable level (*e.g.* [Demberg, 2006]). The pronunciation of a given phoneme also tends to vary slightly depending on its location within a syllable. For instance, the two `[b]` sounds in the word `[bob]` are subtly different because one appears syllable-initially and the other syllable-finally [Kahn, 1976].

More recently, researchers have explored incorporating syllabification into L2P modules. Müller [2001b] uses syllabification as part of a German letter-to-phoneme system. Marchand and Damper [2005] find that using syllabified orthographic forms as input to an L2P system improves L2P performance, compared to a baseline without syllable break information. Intuitively, this finding makes sense. If we want to convert the word *shorthand* to its representative phonemes, knowing that there is a syllable break between the *t* and the *h* should prevent us from predicting that the two letters combine to form the single phoneme `[T]`. Moreover, the position of vowels within a syllable often affects the exact phoneme given letters produce. For example, the verb *presents* has a long *e* sound, and is usually syllabified as *pre—sents*, while the noun *presents* has a short *e* sound and is usually syllabified as *pres—ents*.

Finally, there is some scope for incorporating syllables into synthesis modules. To date, most TTS systems have concatenated together diphones — the units of sound that stretches from the middle of one phone to the middle of the next phone. However, concatenating together longer units typically improves the quality of synthesized speech [Deligne *et al.*, 2001]. Currently, it is considered too costly to store long sequences of phones for concatenation, as the quality improvement does not justify the extra memory requirements. However, in theory, concatenative speech synthesis could be done at the level of the syllable.

On the whole, syllables and syllabification are an important part of text-to-speech systems. Unfortunately, due to the productive nature of language, a dictionary look-up process for syllabification is inadequate. No dictionary can ever contain all possible words in a language. For this reason, we must to turn to systems that can automatically syllabify out-of-dictionary words. Automatic syllabification will be the focus of this thesis.

1.2 Approach to the Problem

Historically, data-driven approaches to TTS ands its components have been rare. However, Damper [2001] argues that this is largely derived from a Chomskian world view that sees innate rules as a prerequisite for language learning; in practice there is much to be gained from employing data-driven approaches. An array of recent research has applied various data-driven techniques to automatic syllabification. The two most common approaches are lazy learning methods, which syllabify new words by finding similar examples in a syllabification dictionary, and hidden Markov models, which predict the most probable syllabification pattern based on observed characters.

In this thesis, I pursue a different strategy, taking a discriminative approach to automatic syllabification. Discriminative techniques learn a mapping from explanatory variables, or features, to desired output labels. To use such a framework for my problem, I formulate syllabification as a tagging problem. When viewed as a tagging problem, every letter or phoneme in an input word must be assigned a tag that indicates whether or not the character is at a syllable boundary. I consider a number of different tagging schemes. Some are positional — the tags indicate nothing more than the characters' position in the syllable. Others are linguistically informed, so that the tag says something about the role the character is playing in the syllable. I use SVM-HMM, a type of structured support vector machine (SVM), to learn the correspondences between characters and tags. My system is language-independent; I use it to syllabify words in English, German, and Dutch.

In addition to using discriminative training to syllabify letters and phonemes, I also apply several rule-based approaches to the phoneme domain. I present an implementation of Kahn's [1976] legality principle, and a version of Selkirk's [1984] Sonority Sequencing Generalization. Rule-based methods are advantageous because they require no training data, and can build on well-studied linguistic theories. However, they are language-dependent and cannot easily be applied to the letter domain.

1.3 Contributions of this Research

This thesis makes a number of contributions. First and foremost, I present an automatic syllabification system that beats state-of-the-art performance on both letters and phonemes. In a direct comparison on English letters, my syllabification system reduces the error rate from the previous state-of-the-art by one-third. Although head-to-head comparisons are lacking, my system also achieves world-best word accuracy on English phonemes. In German and Dutch, word accuracy exceeds 99 percent for both letters and phonemes.

To produce these results, I use a discriminative approach that requires both a tagging scheme and feature set suitable for the task. The tag sets and feature sets I develop for syllabification are the second contribution of this thesis.

Third, my syllabification system, when combined with an existing state-of-the-art letter-to-phoneme system, increases L2P accuracy in English, Dutch, and German. Previous systems have improved English L2P accuracy by adding gold-standard dictionary syllabifications, and improved German L2P accuracy by adding learned syllabifications. However, to my knowledge, this is the first example of an automatic orthographic syllabification system producing improvements in English L2P accuracy.

This thesis also presents several rule-based techniques for syllabification of phonemes. In particular, the rule system based on the legality principle achieves very good word accuracy, without benefit of labelled training data.

Additionally, I proffer evidence that the NETtalk dataset, heretofore in common usage for English syllabification, is unsuitable for the task. In the letter domain, NETtalk contains a substantial proportion of completely non-sensical 'gold standard' syllabifications that are likely to confound syllabification systems and any L2P system that depend on them.

Finally, I contribute a thorough survey of previous work on automatic syllabification.

1.4 Outline

The remainder of this thesis is structured as follows. In Chapter 2 I present the linguistic background of syllables and syllabification. In Chapter 3 I explain the theory underpinning SVM-HMM, the discriminative method I use in my system. I discuss previous work on the topic in Chapter 4. Chapter 5 explains how I develop tag and feature sets to apply SVM-HMM to syllabification of letters, and presents experimental results in that domain. Chapter 6 shows the application of my discriminative technique to phonemes and introduces several rule-based systems for phones. Conclusions and final thoughts are presented in Chapter 7.

Chapter 2

Linguistic Background

Syllabification is the process of dividing a word into its constituent syllables. Presently, I define the syllable and outline several principles that can be used to perform syllabification. First, however, I must clarify a common source of confusion in this area. From a linguistic perspective, a syllable only exists in the phonological domain. We can divide strings of phonemes into syllables; we cannot so divide strings of letters.

In opposition to this, most non-linguists find it perfectly reasonable to syllabify the orthographic representation of a word. Some of this confusion perhaps arises from dictionary headwords. Most North American dictionaries will divide the letters of a word up into segments that look, to the non-linguist, like syllables. The primary purpose of these 'divisions in entry words' is to indicate to typesetters the points where a word may be broken at the end of a line [Gove, 1993]. Typically, word breaks are indicated by a hyphen, so this process is often called hyphenation.

The process of deriving divisions in entry words is complex — Webster's Dictionary lists 26 rules for dividing entry words [Gove, 1993]. All are very detailed and most give rise to a number of exceptions. In broad strokes, Webster's tends to create a segment by attaching a consonant to the following vowel; consonants are attached to the preceding vowel only when that vowel is short and stressed. However, these guidelines are frequently overridden by considerations of morphology.

Generally, common inflectional affixes (*e.g.* *-tion*, *-ing*, *non-*) form an indivisible segment; for less common affixes, this does not hold as consistently. Similarly, divisions of compound words usually occur between the two constituent words (*e.g.* *nose—bleed*).

There is definitely a relationship between orthographic divisions and pronunciation [Guenter, 29 November 2006]. Indeed, the main function of the divisions in words is to ensure hyphenation is consistent with pronunciation. Similarly, there is also a relationship, however ill-defined, between syllabification of phonemes and divisions in entry words. In a number of cases the divisions of a word's orthographic and phonetic forms are identical (*e.g.* *rapt—ly*, `[r{pt|lI]`[1]). However, the two processes are distinct.

Dividing letters into segments is mostly an engineering problem: the fact that we can use it to produce performance improvements on a task like converting letters into phonemes is both necessary and sufficient to justify the endeavour. Conversely, syllabification of phonemes is of significant theoretical interest for its own sake, in addition to the role it plays in text-to-speech systems.

In this thesis, I treat syllabification of phonemes and division of letters as two separate but related problems. In this chapter, whenever I mention syllables or syllabification, I am exclusively referring to the phonological entity. Throughout the rest of this thesis, I refer to 'syllabification' of orthographic forms, using it as shorthand for 'reproducing the dictionary's end-of-line divisions.'

2.1 The Syllable

By most accounts, a syllable is composed of two constituents: a rhyme (or rime) and an (optional) onset [Spencer, 1996; Kessler and Treiman, 1997]. The rhyme can be further subdivided into a nucleus (typically a vowel), and an optional coda. This internal structure is not universally accepted, however. A structure composed of a body and a coda (where the body further subdivides into nucleus and onset) has

[1]The DISC encoding for phonemes is used throughout this thesis.

also been postulated [Kessler and Treiman, 1997]. Conversely, Blevins [1995] argues that a syllable is constructed of a single constituent: an indivisible rhyme. On this interpretation, the onset and coda are not constituents in their own right, but are just whatever is left once we remove the nucleus. Still other theories deny the rhyme as a meaningful construction, arguing that it is synonymous with the nucleus, and therefore a useless addition [Spencer, 1996]. Proponents of this perspective advocate a completely flat syllable structure, consisting of only a nucleus, and optional onsets and codas. Debates in the linguistic community notwithstanding, I take a syllable to be a nucleus and its surrounding consonants, and refer to the consonants before and after the nucleus as the onset and coda, respectively.

Not all languages allow the same flexibility as to syllable types — onsets and codas are not always optional. A language's possible syllables are often typified according to whether the onset is obligatory (or not) and whether a coda is permissible (or not). Examples of languages categorized by this typology appear in Table 2.1. Note that there are no languages in which the onset is prohibited and/or the coda is required.

	Onset Required	Onset Optional
Coda Prohibited	Arabela[a], Hua[b]	Fijian, Hawaiian
Coda Permitted	German, Totonac[c]	English, Italian

[a]an endangered language spoken in Peru and Ecuador
[b]spoken in Botswana
[c]spoken in Eastern Mexico

Table 2.1: Examples of syllable types across languages (adapted from [Spencer, 1996; Blevins, 1995]).

More elaborate typologies further consider whether syllables can have complex nuclei, codas, or onsets. Complex constituents are composed of more than one phoneme, as in the onset of `[str{p]` (*strap*).

Most phonologists argue that the syllable is a valuable "prosodic constituent" (e.g.[Blevins, 1995; Spencer, 1996]). Blevins [1995] lists four reasons for the importance of the syllable:

- many phonological rules and constraints apply within the context of a syllable,

- other phonological rules apply at syllable boundaries,
- many language games and play languages (*e.g.* Ubbi Dubbi) imply the use of syllables,
- native speakers have strong intuitions about the existence of syllables, and in some languages speakers also have a strong sense about where syllable breaks occur.

Historically, there has been some disagreement as to the syllable's usefulness. Kohler [1966], for example, argues that syllables are actually "harmful" because they sometimes prevent us from appreciating a word's underlying phonological structure. However, views like Kohler's are now distinctly in the minority, and on the whole, the idea of a syllable is no longer a particularly controversial one.

2.2 Linguistic Theories of Syllabification

In spite of the broad linguistic consensus on the existence of syllables, syllabification remains somewhat contentious. Indeed, one of Kohler's arguments against the syllable as an entity is the ambiguity inherent in syllabification. Of course, for languages like Hawaiian and Hua, where codas are prohibited, syllabification is trivial. Every syllable has a single vowel, and any consonants must be onsets. However, for the vast majority of languages, codas are permissible, so the process is not so clear. Given a consonant, how do we determine whether it should attach to the previous nucleus as a coda, or to the following nucleus as an onset? What should be done with the `[g]` in `[Eg5]` (*ego*)? Should we syllabify it as `[E|g5]` or `[Eg|5]`? This is the crux of the syllabification task. Several theories attempt to model this decision; I present a number of the most prominent. The following is by no means an exhaustive list.

Maximal Onset Principle

The most basic theory is the maximal onset principle (MOP) [Blevins, 1995]. It states that given a sequence ... *VCV* ..., we always syllabify it *V—CV*, expanding the onset at the expense of the previous syllable's coda. This principle holds very strongly, particularly for English: it is used by Merriam-Webster when syllabifying the phonetic representations of dictionary words [Guenter, 29 November 2006]. However, there is considerable evidence that this principle is not universal. By way of counter example, in the Australian Aboriginal language Kunjen, all syllables begin with vowels, thus violating MOP. More saliently, the principle is frequently violated, even for languages where MOP generally holds. A letter sequence like `[*kf@s]`, for example, is clearly not plausible as an English word, because it would be unpronounceable. Thus, given a word like `[brEkf@st]` (*breakfast*), we would not want to maximize the onset of the second syllable (*i.e.* `[*brE|kf@st]`, preferring instead to syllabify the word as `[brEk|f@st]` and produce two pronounceable syllables.

In other words, MOP often holds, but it is also frequently violated due to other constraints. The obvious question then becomes, how do we know when to ignore MOP?

Sonority Sequencing Generalization

Two different theories provide an answer to this question. One theory is based on the idea of sonority. A sound's sonority is its inherent loudness, holding other factors equal [Crystal, 2003]. In a sonority scale, phonemes are ranked from least sonorant to most sonorant, as in Figure 2.1. Selkirk [1984] introduces the sonority sequencing generalization (SSG), which states "In any syllable, there is a segment constituting a sonority peak that is preceded and/or followed by a sequence of segments with progressively decreasing values." The generalization tends to rule out many impossible syllables. For instance, a hypothetical syllable like `[*Jkav]` would violate SSG because the `[J]` is more sonorant than `[k]`. However, SSG does not rule out the equally implausible syllable `[*bmav]`, because `[m]` is more sonorant than [b].

Selkirk argues that sequences like [bm] can be eliminated by requiring a minimum sonority difference between adjacent consonants in a syllable. In practice, this margin would vary by language. Indeed, Selkirk never claims to be presenting a complete sonority scale with invariant sonority values. Rather, she couches her scale as provisional, where the relative positions of the sounds are more important than their absolute values. The position of certain sounds within the scale, and the exact values that should be on the scale remains somewhat of an open problem.

As Selkirk conceived of it, the SSG should be used in conjunction with a series of templates that describe the kinds of onsets and codas a syllable can have. Together, the SSG and the templates represent a complete theory of syllabification. However, SSG may also be combined with MOP, as Goslin and Frauenfelder [2001] have done.

class	example	sonority value
vowels	[a], [V]	6
glides	[w], [y]	5
liquids	[r], [l]	4
nasals	[m], [n]	3
fricatives/affricates	[z], [J]	2
plosives	[k], [b]	1

Figure 2.1: A sonority scale, adapted from Spencer [Spencer, 1996].

Legality Theory

A second theory that can tell us when to override MOP is the legality principle. As introduced by Kahn [1976], the legality principle states that all possible syllable onsets appear as word-initial consonant clusters, and all possible syllable codas appear as word-final consonant clusters. The notion of legality by itself is insufficient to syllabify many words. For example, in a word like [m#st@R] (*master*), the cluster [st] is found repeatedly both at the beginning and end of words. Therefore, Kahn proposes to combine his legality principle with MOP. Furthermore, Kahn argues that any sequence that cannot be syllabified by the legality principle (*e.g.*

[akpsa] — all of [kps], [kp], and [ps] are illegal as both onsets and codas) must be rejected as a valid word in English.

Kahn also allows for ambisyllabicity of consonants. Ambisyllabicity permits some consonants to simultaneously contribute to the coda of one syllable and the onset of the next. Thus, in the word [m#st@R], we have two syllables: [m#s] and [st@R], with the [s] belonging to both syllables. Ambisyllabicity is a fairly controversial idea. An opposing theory is that of resyllabification. On this theory, syllabification is a two-stage process: we first syllabify a word with the consonant attaching to the onset of the second syllable, in keeping with the Maximal Onset Principle (*i.e.* [m# | st@R]). Subsequently, the word is resyllabified so the consonant attaches unambiguously to the coda of the first syllable to fit better with the word's stress patterns (*i.e.* m#s | t@R). At no point does a sound belong to two syllables simultaneously. In my work I do not pursue the ideas of ambisyllabicity or resyllabification. Largely, this avenue is precluded by a lack of data. I must rely on dictionaries for a gold standard, and dictionaries provide a single, unambiguous syllabification.

Optimality Theory

SSG, MOP, and the Legality Principle are not necessarily mutually exclusive. Hammond [1997a] makes an attempt to combine several of these theories as a test bed for optimality theory (OT). Optimality theory stands in opposition to rule-based approaches, which require all constraints to be satisfied. In OT, constraints are ranked in order of importance, and lower-ranked constraints can be happily violated in order to satisfy higher-ranked constraints. In Hammond's theory, the constraints are:

1. PEAK: syllables have one vowel.
2. LICENSING: all words must be composed of syllables.
3. SONORITY: onsets must increase and codas must decrease in sonority.
4. FAITHFULNESS: the output of the syllabification should be the same as the input.

5. ONSET: syllables begin with a consonant.

6. NOCODA: syllables end with a vowel.

7. COMPLEX: syllables have at most one consonant at an edge.

The NOCODA and ONSET constraints together specify something akin to the maximal onset principle. However, SONORITY is the higher-ranked constraint, so NOCODA can be violated in order to maintain SONORITY.

Empirical Studies

Treiman and Zukowski [1990] conduct an empirical study to determine how well actual human syllabification practice accords with phonological theory. Their findings are twofold. First, people strongly obey the legality principle — when asked to syllabify `[EdmVntVn]` (*Edmonton*), subjects will never produce `[E|dmVn|tVn]` or `[Edm|Vn|tVn]` because `[dm]` is not valid as either a word-initial or a word-final cluster. Second, people seem to have greater deference to SSG than to MOP. A cluster like `[st]`, with a small sonority gap, is more likely to be divided than a cluster like `[dr]`, where the sonority gap is larger. That is, `[Is|t1t]` (*estate*) is preferred to `[I|st1t]`, but `[mV|drId]` (*Madrid*) is preferred to `[mVd|rId]`. If MOP were more important, we would expect people to prefer both `[mV|drId]` and `[I|st1t]`.

Goslin and Frauenfelder [2001] compare several syllabification algorithms (in French) and report how often the different approaches agree. Two of the algorithms they compare are based on SSG, two are rule-based algorithms based on the legality principle, and the fifth approach is MOP. The details of the algorithms are largely specific to the French language. In a comparison of all the intervocalic consonant clusters in the French corpus BDLex, Goslin and Frauenfelder find that agreement between algorithms is high when there are only two consonants in the cluster, but drops off sharply as the number of consonants increases. For example, in the case of the two legality-driven, rule-based algorithms, agreement drops from 99 percent with two-consonant clusters, to only 9 percent for clusters containing more than three consonants. Such high levels of disagreement, even between theoretically very similar approaches, indicates that subtle variations in models can have substantial effects.

2.3 Summary

In this chapter, I defined a syllable as a phonological construct consisting of a vowel-based nucleus, optionally preceded by consonant(s) forming the onset and optionally followed by consonant(s) forming the coda. I have introduced three theories of syllabification: the maximal onset principle, the legality principle, and the sonority sequencing generalization. However, no theory or combination of theories can correctly syllabify all words, and small discrepancies between implementations can have large effects on output. Consequently, any rule-based approach to syllabification will leave some scope for improvement.

This chapter also draws a distinction between syllabification of a word's phonetic form, and reproducing the dictionary end-of-line divisions in the orthographic form. Because the latter is not a well-defined linguistic problem, rule-based approaches are unlikely to succeed. Notwithstanding the differences between the two tasks, I refer to syllabification of both phonemes and letters for the balance of this thesis.

Chapter 3

SVM-HMM: A Primer

The primary contribution of this thesis is applying a discriminative technique to the problem of syllabification. This chapter introduces the specific discriminative technique that my system employs: SVM-HMM. SVM-HMM is a type of structured support vector machine (SVM) that combines attributes of a hidden Markov model (HMM) and a single-instance SVM. In this chapter, I first outline HMMs and single-instance SVMs. Subsequently, I explain structured SVMs in general, and SVM-HMM in particular.

3.1 Hidden Markov Models

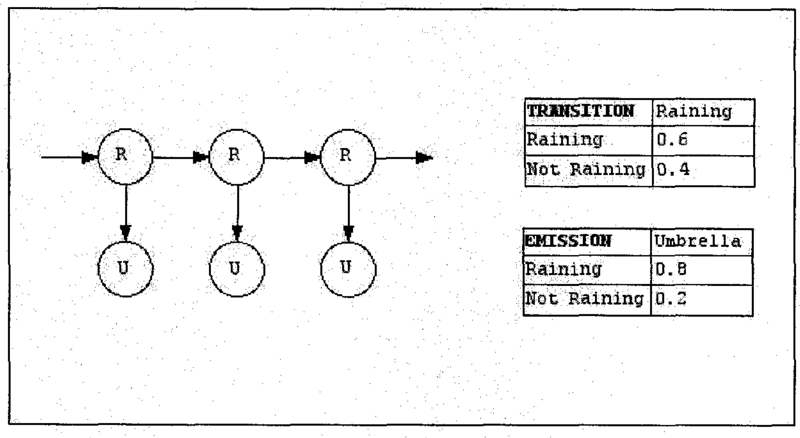
Hidden Markov Models (HMMs) are generative models of the joint probability of an observation and its (hidden) label. Based on the known probabilities for observations and labels, we can predict which label accompanies a given observation. In the canonical example, we find ourselves locked in a windowless basement. We want to know if the weather is rainy, sunny, or cloudy, but all we can observe is whether the guard carries an umbrella. We know that the guard tends to carry an umbrella when it rains. We also know the probability that a rainy day follows a non-rainy day.

Figure 3.1 shows a very simple HMM for this situation. We know there is a 60 percent chance that it will rain today if it rained yesterday. More formally, this is the transition probability that the current hidden state R_t will follow the previous hidden state R_{t-1}. We know that there is an 80 percent chance it will rain today if our guard has an umbrella. This is the emission probability of the current hidden state R_t given the current observation U_t. We further know that it was sunny on the day we were locked in the basement ($\bar{R_0}$. If the guard does not carry an umbrella on the first day, we can calculate the probability of rain on day one as:

$$Pr(R_1) = Pr(R_1|\bar{R_0}) \cdot Pr(R_1|\bar{U_1})$$

$$= 0.4 \cdot 0.2$$

$$= 0.08.$$



TRANSITION	Raining
Raining	0.6
Not Raining	0.4

EMISSION	Umbrella
Raining	0.8
Not Raining	0.2

Figure 3.1: An illustration of the HMM set-up for the weather example.

In the general case, with j possible hidden states, we can determine the probability that the ith hidden state, S_i, is equal to x, given the ith observation, O_i by calculating:

$$Pr(S_i = x) = \sum_j [Pr(S_{i-1} = j) \cdot Pr(S_i = x|S_{i-1} = j)] \cdot Pr(S_i = x|O_i).$$

This is a first-order HMM, where state transitions only depend on one previous state; higher-order HMMs are also common. We can determine the optimal sequence of states for a whole sequence of observations using the Viterbi algorithm.

HMMs can be used to predict syllable boundaries (Chapter 4 presents some previous work in that vein). In such a case, the letters or phonemes of the word are the observations, while the labels that indicate syllable boundaries are the hidden states. However, there are a number of reasons why HMMs are suboptimal for the task. The primary objection is that syllabification is not usually done in the equivalent of a locked, windowless basement. We know all the letters in the word, and it makes sense to look at more than one of them in determining the best 'state' to select. However, any features used to represent the observations of an HMM must be conditionally independent, which precludes the inclusion of some very useful features. Another problem is that the arrows in Figure 3.1 point the 'wrong' direction. In the rain example, the hidden state (the weather) was influencing whether or not the guard had an umbrella. In syllabification, the fact that there is a syllable boundary after the ith letter does not affect what that letter is. The letters (observations) determine the syllable boundaries (hidden states), not the other way around.

3.2 Support Vector Machines

In contrast to HMMs, a Support Vector Machine (SVM) is a discriminative supervised learning technique that allows for a rich feature representation of the input space. We learn from training examples, and then generalize what we have learned to new, unseen cases. In its simplest guise, an SVM is a tool for performing binary classification of objects, based on features of those objects.

Before considering the general case and the mathematics that underpin it, I will sketch a simple high-level example of the intuition behind SVMs. There are thirty teams in the National Hockey League (NHL), and each year 16 of them qualify for the playoffs. Suppose that, in January, we want to predict which teams will make the playoffs at the end of the season. Based on our knowledge of the NHL, we might guess that two factors are good predictors of whether or not a team will be in the postseason: the number of wins the team has, and the team's goal differential (goals scored less goals allowed).

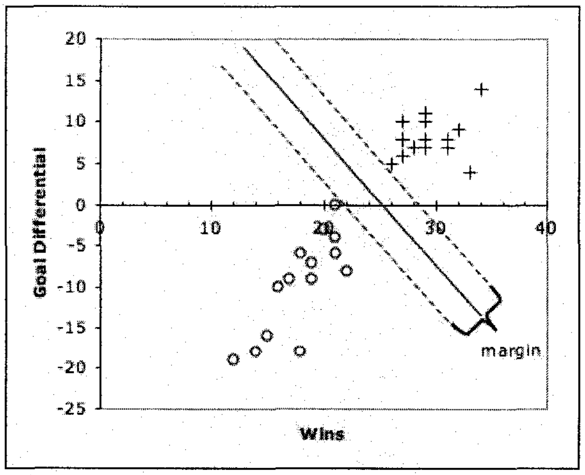


Figure 3.2: An example of a maximum margin separating positive and negative examples.

Further suppose we have access to last year's statistics. We know which teams made the playoffs (our positive examples) and which teams did not (our negative examples). We also know the win totals and goal differentials each team had last January. We will use these two features to describe our input examples. In an ideal world, we can plot our negative and positive examples as in Figure 3.2, and draw a straight line dividing up the space between playoff teams (plus signs) and non-playoff teams (circles). This line serves as our classifier: teams whose features put them above the line will be in the playoffs, while teams that fall below the line will get early tee times. We do not draw our classifier line arbitrarily; rather, we choose the line that will maximize the margin between positive and negative examples. Once we have our line, we can plot next year's teams on the same axes, and predict which teams will make the playoffs.

Mathematically, we formalize this as the search for a weight vector w and offset b that maximizes the margin between positive and negative examples[1]. Each of the N training instances is associated with features, $x_i \in \Re^p$ and a desired output,

[1]all equations in this section are adapted from [Hastie *et al.*, 2001]

$y_i \in \{-1, 1\}$. The width of the margin is defined by $2 \cdot \frac{1}{w}$, so we can maximize the margin by solving the following argmin:

$$\text{argmin}_{w,b}\frac{1}{2}||w||^2 \quad \text{s.t.} \quad y_i(w \cdot x_i - b) \geq 1, i = 1, \ldots, N. \tag{3.1}$$

Finding this argmin is a convex optimization problem; we use a quadratic program to determine the solution.

Once we have found w and b, we can use them to classify new examples using the classification rule:

$$\hat{y} = \text{sign}[w \cdot x + b]. \tag{3.2}$$

Extending the Basic SVM Framework

Practically speaking, there are a number of conflating factors that make SVMs more difficult to employ for real-world prediction problems.

First, real-world classification is almost never so easy. If we could perfectly predict the playoff teams from the wins and goal differentials in January, there would be no need to play the rest of the games. In reality, some teams with a lot of wins and a good goal differential at the halfway mark will not make the playoffs, while other teams with few wins and poor goal differentials will start to play better and get in to the postseason. Figure 3.3 plots the actual wins and goal differentials of NHL teams in January of 2007, with teams that actually made the playoffs represented by plus signs. For this diagram, there is no straight line that can cleanly divide the playoff teams from the non-playoff teams. In other words, the data is not linearly separable. To overcome this problem, SVMs employ slack variables. Conceptually, a slack variable allows us to treat one or more examples as outliers, and ignore them when drawing the line.

Mathematically, the addition of slack variables, s_i, means we have to reformulate equation 3.1 as:

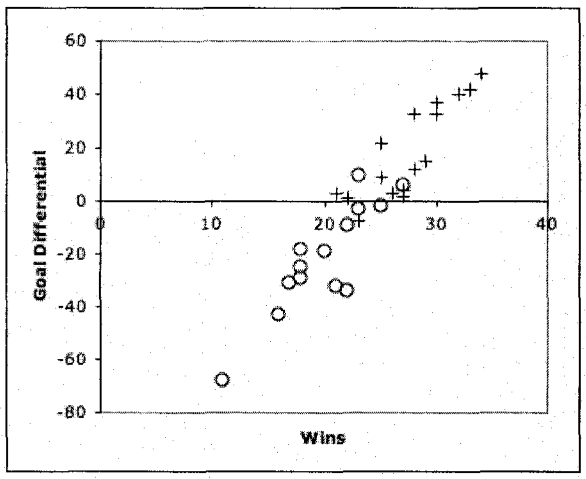


Figure 3.3: An example of data that is not linearly separable.

$$\text{argmin}_{w,b,s}\frac{1}{2}||w||^2 + c \cdot \sum_i s_i \quad \text{s.t.} \quad \begin{cases} y_i(w \cdot x_i - b) \geq 1 - s_i, i = 1, \ldots, N \\ s_i \geq 0, i = 1, \ldots, N. \end{cases} \tag{3.3}$$

Equation 3.2 remains unchanged.

The s_i values in Equation 3.3 represent how much the associated x_i values are misclassified by under the current w. Therefore, the $\sum_i s_i$ term represents the total amount of error across all examples. The amount of error we are willing to accept on the training data is determined by the c term, which is a tunable parameter. In practice, the value of c often makes a significant difference in the accuracy on our test examples.

A second complicating factor is the number of features. In a typical classification task, there will be more than two factors that are important in making our prediction. (In the NHL example, we might consider things like player injuries, for instance.) In some domains we might use thousands, or even millions. Thus, we are typically not drawing a line to divide two-dimensional space, but rather constructing a hyperplane to divide n-dimensional space. Finding this hyperplane is more time-consuming than drawing a line, although it does not change the underlying mathematics.

It is also non-trivial to determine which features we should use to represent our problem. In practice, it usually will not hurt to include extra features (the SVM will learn to ignore features that do not help classification). However, missing a feature that is crucial to the task can greatly impede performance.

Another thing that makes SVMs more complicated is that many classification problems are not binary. Suppose we want to classify the NHL teams as
(1) non-playoff teams,
(2) teams that get eliminated in the first round of the post-season, and
(3) teams that advance to the second round or later.

To perform multi-class classification, we need to alter equation 3.3. The first thing we need to do is modify our notation somewhat. Because we have more than two classes, we can no longer rely on the sign of $w \cdot x$ to perform our classification — the desired output must be represented explicitly. We use the notation $\Psi(x_i, y_i)$ to represent the relationship between the features of an instance x_i and its desired class label y_i. Given a set of possible class labels, l, we want to maximize the difference between the desired label, y_i and all competing, incorrect labels, for each training instance:

$$\text{argmin}_{w,b,s}\frac{1}{2}||w||^2+c\cdot\sum_i s_i \quad \text{s.t.} \quad s_i \geq 1(l_i \neq y_i)-w(\Psi(x_i,y_i)-\Psi(x_i,l_i)), \forall i, l_i. \tag{3.4}$$

The $1(l_i \neq y_i)$ term is an indicator function, which will be equal to zero when $l_i = y_i$. Thus, we still have the $s_i \geq 0$ constraint in the multi-class case. However, we are adding an extra constraint for every other possible class. This greatly increases the number of constraints that must be enforced when solving the quadratic program.

Multi-class problems also change the classification rule slightly:

$$\hat{y} = \text{argmax}_{l_i} w(\Psi(x_i, l_i)). \tag{3.5}$$

We can simply try all possible labels until we find the one that maximizes the product of $w \cdot \Psi$.

Whether an SVM is doing binary or multi-class prediction, it has a significant drawback compared to an HMM: the SVM can only make single predictions in isolation. This is problematic for many problems, including the NHL example. Exactly 16 teams make the playoffs, with a certain number coming from each conference and division. However, the single instance SVM might predict that 18 teams will make the playoffs, or that all the playoff teams are in the same conference. Whether a given NHL team makes the playoffs is somewhat dependent on what other teams do. This sort of problem is very much an obstacle if we want to use SVMs to perform automatic syllabification. Whether a given letter is at a syllable boundary is dependent on where the other syllable boundaries are. We are not performing a series of individual classification problems; we want to make predictions about a number if interrelated entities. There is a structure to the problem we are trying to solve, but the SVM does not allow us to take advantage of it.

3.3 Structured SVMs

The structured support vector machine was designed to deal with this exact shortcoming of single-instance SVMs. In many ways, structured SVMs are a fairly straightforward extension of the multi-class case [Tsochantaridis *et al.*, 2004]. Indeed, the structured SVM derives its weight vector by solving a generalization of Equation 3.4:

$$\text{argmin}_{w,b,s} \frac{1}{2} \|w\|^2 + c \cdot \sum_i s_i \quad \text{s.t.} \quad s_i \geq \text{LOSS}(y, y_i) - w(\Psi(x_i, y_i) - \Psi(x_i, y)), \forall i, y \in Y. \tag{3.6}$$

Classification is done using the following:

$$\hat{y} = \text{argmax}_{y \in Y} w(\Psi(x_i, y)). \tag{3.7}$$

There are two differences between the multi-class and structured cases. First, we substitute the indicator function with a general-purpose loss function that varies depending on the structure being predicted. Second, we no longer have a nice discrete list of possible competing wrong class labels, l_i. Instead, we are trying to maximize the the difference between the correct structure, y_i and all other possible (but incorrect) structures, $y \in Y$. This is where the difficulty arises. In the multi-class case, even if we have 10,000 possible class labels, there are only 9,999 incorrect answers. By contrast, if we are attempting to predict a complex structure, the number of potential answers is exponential, or even infinite. This explodes the number of constraints the quadratic programmer must satisfy, and the number of possible answers the argmax must search over. Consequently, finding a solution becomes prohibitively expensive, if not impossible.

To overcome this obstacle, Tsochantaridis *et al.* [2004] propose to solve the optimization in an online fashion, adding constraints on an as-needed basis. The general approach is as follows:

1. Using Equation 3.7, find the (incorrect) output structure $\bar{y}$ that is imposing the largest error cost, according to the current weight vector.

2. Add constraints associated with $\bar{y}$ to the optimization problem as set out in Equation 3.6.

3. Derive a new weight vector, using the new set of constraints.

4. Move to the next training example and return to step 1.

This process continues to iterate until no new constraints are added to the optimization problem. In practice, in order for new constraints to be added, the error imposed by the structure $\bar{y}$ must exceed the slack value by more than some small constant ϵ.

Structured SVMs have been shown to find a solution that is within ϵ of being optimal. Moreover, as long as Equation 3.7 can be solved in polynomial time, the optimization itself is polynomial.

SVM-HMM

To apply the formalism to automatic syllabification, I use a specific instance of structured SVMs, called SVM-HMM. Introduced by Altun, Tsochantaridis, and Hofmann, SVM-HMM predicts a sequence of labels for a sequence of observations [Altun *et al.*, 2003]. To do this, it learns a classifier using Equation 3.6. The loss function is Hamming distance; on this metric, the more labels a candidate sequence y has in common with the correct sequence y_i, the lower its loss will be. The output space Y is infinite, consisting of all possible sequences that can be constructed from the label set Σ. However, for any given training sequence x_i, the number of possible label sequences is fixed, because there must be exactly one label for each observation in the input sequence. To classify sequences, a Viterbi decoder is used to solve the argmax in Equation 3.7.

The name SVM-HMM derives from the two types of features that are incorporated in Ψ. First, there are features that capture the relationship between the observations and labels, as in a typical, non-structured SVM. These 'emission' features require no conditional independence assumptions, as would be required in an HMM framework. Second, there are features representing the transition probabilities between labels in an output sequence, as in an HMM.

Applying SVM-HMM to automatic syllabification allows me to have the benefit of discriminative training and an expressive feature set, while predicting all of a word's syllable boundaries at once. Moreover, SVM-HMM is a freely available software package[2]. The software package requires users to define their own emission features; transition features are supplied by the package, using a first-order Markov assumption. I discuss the emission features for the syllabification task in Chapter 5.

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

3.4 Roads Not Taken

SVM-HMM is by no means the only way to combine an arbitrary feature set with HMM-style transition probabilities. McCallum, Freitag, and Pereira introduce Maximum Entropy Markov Models (MEMMs), in which a single probability function is used to estimate the probability of a state, given the observation and the previous state [McCallum *et al.*, 2000]. Maximum Entropy Markov Models allow non-independent features, and maximize conditional probability (rather than joint) to model the situation, which is more in line with the problems we typically use HMMs to solve. A drawback of MEMMs is that they model the transition probabilities separately for every state. This creates a bias in favour of states with only one outgoing transition, which will necessarily capture more probability mass. Lafferty, McCallum, and Pereira address this problem with Conditional Random Fields (CRFs) [Lafferty *et al.*, 2001]. CRFs are essentially a modification of MEMMs that model the probability of the entire sequence of labels given the a full sequence of observations.

Another possible approach is Collins's [2002] averaged perceptron, which also takes advantage of complex, interrelated features. Rather than estimating the maximum likelihood weights for each weight, the Collins method uses a perceptron update rule: at each iteration, every training example is assigned the highest scoring label sequence under the current weights. If that sequence is incorrect, the weights are updated. After all iterations are completed, each parameter is assigned to be the average of the values it took on during the training process.

There are also several other variations on the SVM-HMM theme. Taskar, Guestrin, and Koller present Max-Margin Markov (M^3) Networks [Taskar *et al.*, 2003]. M^3 Networks still predict a full sequence of labels, but they maximize the margin for each label in the sequence, rather than the sequence as a whole. This makes the number of constraints polynomial in the length of the sequence. Consequently, M^3 Networks can specify all constraints from the outset, and do not need to iteratively solve quadratic programs. Conversely, McDonald, Crammer, and Pereira use structured multilabel classification, or MIRA, which trains online like SVM-HMM [Mc-

Donald *et al.*, 2005]. However, MIRA only maximizes the margin over the k best incorrect sequences. This exponentially reduces the number of constraints, but at the cost of increasing the complexity of inference — the top k sequences must be found, rather than only one.

CRFs, averaged perceptron, M^3 networks, MIRA, and SVM-HMM have all been shown to achieve good performance on NLP tasks like part-of-speech tagging, text classification, and/or text segmentation. Syllabification is a small enough problem that any of the above methods would be tractable. However, only CRFs, M^3 networks, and SVM-HMM solve a well-defined objective, making those techniques preferable to the averaged perceptron and MIRA. CRFs are less desirable because they are computationally expensive, and do not allow a loss function. M^3 networks and SVM-HMM are nearly identical in their objectives and expressiveness; the availability of a free toolkit for SVM-HMM makes it the obvious choice.

3.5 Summary

In this chapter I presented SVM-HMM, a type of structured support vector machine. SVM-HMM is an excellent formalism for the syllabification task because it is a discriminative method that allows for a complex feature representation. When syllabification is formulated as a tagging task, SVM-HMM can make a unified prediction of all the syllable breaks in a word. SVM-HMM is preferable to techniques like averaged perception and MIRA because it maximizes a well-defined objective; it is more desirable than CRFs because it is less expensive and allows the use of Hamming distance as a loss function.

Chapter 4

Previous Work

Although there is a fair amount of previous work on the topic of syllabification, the literature is not particularly unified. Comparison with previous results is relatively rare, as syllabification is more often used as a test bed for a new technique, or a means to some text-to-speech ends, rather than a task in its own right. It is also very difficult to compare accuracy scores achieved on different data sets as syllabification tends to vary somewhat across dictionaries. Systems have also been created for several different languages (English, Dutch, and German most prominently), which further confounds comparative analysis. Moreover, it is sometimes unclear whether a particular approach has been applied to strings of letters or strings of phonemes. While the same techniques will often work in both domains, the accuracy level achieved varies substantially. Syllabification of phonemes is also a well-known linguistic/phonological process, where syllabification of letters has no such theoretical underpinnings. This allows syllabification of phonemes to explore rule-based or parsing techniques that are not viable in the letter domain.

The upshot is that there is no obvious progression through the literature. I dedicate Section 4.1 to a detailed examination of Marchand and Damper's [2005] Syllabification by Analogy (SbA), which represents the current state-of-the-art for English syllabification. Like my approach, it is a data-driven technique that can be applied to both letters and phonemes. In practice, SbA is language-independent, although I know of no results applying the system to either German or Dutch. Furthermore, unlike other techniques, I can make direct comparisons between SbA and my method. Both SVM-HMM and SbA have been trained and tested using the same data for English letters, which allows me to make direct comparisons between them. Additionally, uniquely in the literature, Marchand and Damper have already directly compared SbA with existing English syllabification techniques. Finally, Marchand and Damper were the first to establish that syllabification has the potential to improve English letter-to-phoneme (L2P) performance. In many ways, their work is the impetus for my research.

In Section 4.2, I discuss other previous work that is closely related to my own, either in the approaches undertaken or the datasets used for testing. Systems in this section are broadly comparable to my discriminative implementation. In Section 4.3, I sketch some other previous work on syllabification that is more tangential to the work of this thesis. Although this chapter combines work on both letters and phonemes of several different languages, I clearly state both the domain and language of application for each technique.

4.1 Syllabification by Analogy

Syllabification by Analogy (SbA) is a modification of the Pronunciation by Analogy algorithm for the syllabification task [Marchand and Damper, 2005]. In a nutshell, SbA finds analogies between an unsyllabified query word and syllabified words appearing in the system's lexicon. Substrings of the query word are compared against all dictionary entries, and the most analogous substrings are concatenated together to form a syllabification of the whole query word. The definition of the 'most analogous' substrings is somewhat intricate; in practice, a directed graph (called a syllabification lattice) is constructed to determine which substrings in the dictionary should be used to syllabify the query word.

Figure 4.1 illustrates a simplified example of SbA's execution. The juncture between every letter is viewed as a potential syllable boundary. Boundaries are indicated by |; non-boundaries are indicated by $*$. For ease of exposition, I use only a small dictionary, and show only a portion of the syllabification lattice. In reality, the graph would contain a node for every letter and junction in the query word.

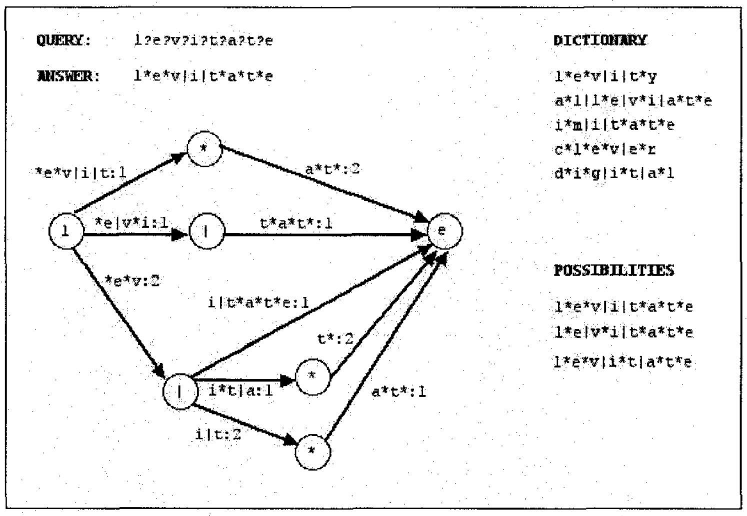


Figure 4.1: An example of a lattice SbA might use to syllabify the input word *lev—i—tate*.

To construct the lattice, SbA compares every letter and junction of the query word with every letter in the dictionary. Any time there is a match between a dictionary substring and a query substring, that substring is added to the lattice. The ? symbols in the query word can match either the | or * markers. Hence, the query `l?e?v?i?t?a?t?e` matches the dictionary entry `l*e*v|i|t*y` up until the letter `y`. This pattern match is captured in the lattice by the nodes labelled `l` and `*`, connected by the arc labelled `*e*v|i|t:1`. The notation `:1` is a frequency count, indicating we have seen the pattern once. Substrings can match anywhere in the word. Thus, the count for the substring `l*e*v|` is two, because we see it both in ***lev—i—ty*** and ***clev—er***.

Once the syllabification lattice has been constructed, every path through the graph represents a plausible syllabification. Many of these paths will produce the same syllabification, as in the example. In cases where there are multiple syllabifications within the lattice, SbA selects the shortest path through the lattice as its best syllabification. Thus, in Figure 4.1, only paths of length two will be considered, eliminating *lev—it—ate* as a possible syllabification. When choosing between shortest path syllabifications, SbA uses the product of the frequencies indicated on the arcs and selects the path with the highest value. In the example, *lev—i—tate*, the correct syllabification, has a frequency product of 2, and so beats out *le—vi—tate*, which only has a frequency product of 1. If there are several syllabifications with the shortest path length and identical frequency products, SbA chooses the whole-word syllabification that appears most frequently. Thus, if the example dictionary contained the word *le—vi—a—than*, making the frequency product on the *le—vi—tate* path also equal to two, SbA would still choose *lev—i—tate* because there are two paths of length two that produce that syllabification.

Marchand and Damper first introduced SbA in the context of their Pronunciation by Analogy (PbA) implementation. Using the NETtalk dataset as their gold standard, they found that applying dictionary syllabification as a pre-processing step in PbA improved L2P performance from 65.35 percent word accuracy to 71.74 percent word accuracy [Marchand and Damper, 2005]. This is an 18 percent relative reduction in the error rate, and highly statistically significant. Given these results, the challenge is to replicate the improvement using an automatic syllabification technique. SbA achieves reasonably good word accuracy on the syllabification task. Experimenting on the NETtalk dataset and using leave-one-out testing, SbA correctly syllabifies 78.1 percent of words. Unfortunately, this level of accuracy is not sufficient to improve accuracy in Marchand and Damper's L2P application. In fact, adding SbA-generated syllabification as pre-processing reduces L2P word accuracy from 65.4 percent to 64.3 percent.

Subsequent to their initial work on the topic, Marchand, Damper, and Adsett perform a detailed comparison of existing syllabification techniques for English [Marchand *et al.*, to appear]. In this comparison they look at syllabification of both letter and phoneme strings. They use NETtalk as one of their gold standards, and also use 18K words taken from the Wordsmyth English Dictionary-Thesaurus. All Wordsmyth words also appear in the NETtalk dictionary, although not necessarily with the same syllabification. They construct a third dataset, Overlap, consisting of the words appearing in both NETtalk and Wordsmyth, with the same syllabifications in each.

Method	Letters	Phonemes
Optimality Theory	36.88	—
Legality Principle	—	74.42
Simple EBG	73.53	83.66
Learned EBG	74.36	83.12
SbA	85.43	91.08

Table 4.1: A comparison of word accuracy scores for several existing English methods, adapted from [Marchand *et al.*, to appear].

For each of the three datasets, Marchand *et al.* compare their SbA implementation with existing methods for English. The two other data-driven approaches compared are variations of exemplar-based generalization (EBG), sometimes called instance-based learning, or IB1-IG [Daelemans and van den Bosch, 1992; Daelemans *et al.*, 1997; van den Bosch, 1997]. EBG generally performs a simple database look-up to syllabify a test pattern, choosing the most common syllabification. In cases where the test pattern is not found in the database, the most similar pattern is used to syllabify the test pattern. Similarity is determined by comparing a focus letter and its surrounding context, where each context letter is weighted according to its importance. Simple EBG systematically applies 15 different sets of integer weights to determine which one produces the best performance. Learned EBG uses information entropy (log probabilities) to determine the weights for a given context letter at a given position. In the letter domain, Marchand *et al.* also compare against an implementation of optimality theory [Hammond, 1997b]; in the phoneme domain, they compare against an implementation of the legality principle [Fisher, 1996]. It is unclear why they do not attempt to apply Hammond's optimality theory in the phoneme domain, where it would be more suited.

Across the board, performance is best on the Overlap dataset. Regardless of dataset, SbA has the highest accuracy scores by a very wide margin. A summary of the best results for each method on the Overlap set is reproduced in Table 4.1.

4.2 Related Work

Daelemans and van den Bosch's Data-Driven Methods

One of the earliest works on automatic syllabification is Daelemans and van den Bosch's [1992] neural network-based implementation for Dutch letters. Neural networks are the precursors of SVMs, and the two technologies share a number of similarities. Daelemans and van den Bosch train two backpropagation networks that accept an input letter and its surrounding context and output whether that letter is at a syllable boundary or not. They design one of the networks to be aggressive, postulating lots of syllable boundaries; the other is designed to be conservative. The output of these two networks is then fed into a third network which trades off the responses of the conservative and aggressive networks. Daelemans and van den Bosch compare this connectionist approach to Learned EBG (described above). Results are reported in terms of letters correctly classified (either at a syllable boundary or not). They find that EBG out-performs the neural network, achieving 96.6 percent letter accuracy against only 94.2 percent for the back-propagation network.

Daelemans, van den Bosch, and Weijters also explore using IGTrees for English syllabification [Daelemans *et al.*, 1997]. IGTrees effectively store the same information as Learned EBG, but compress it into a tree form. On a set of 20K CELEX English orthographic forms, IGTree scores 94.53 percent, slightly lower than Learned EBG on the same data set (95.21%). In subsequent work, the same authors incorporate a Learned EBG module for syllabification of English phonemes into a larger IGTree L2P system [van den Bosch *et al.*, 1998; van den Bosch, 1997]. Their syllabification system achieves 97.78 percent word accuracy on CELEX English phonemes, but they find that the system performs better without the syllabification module.

HMM-based Methods

Krenn [1997] introduces the idea of treating syllabification as a tagging task. Working from a list of syllabified German phoneme strings, she automatically generates tags for each phone. A second-order Hidden Markov Model is then used to predict sequences of tags; syllable boundaries can be trivially recovered from the tags. The HMM is intended to be incorporated as module in a larger TTS system.

Krenn experiments with two different tag schemes. Her Onset-Nucleus-Coda model assigns all vowels and diphthongs to be nuclei. All consonants preceding the nucleus in the syllable are onsets; all consonants following the nucleus are codas. Krenn's Positional Model assigns the first phoneme in every syllable to be a *B* (beginning), the last phoneme to be an *E* (end), and all intervening phonemes to be *M*s (middle). In Chapters 5 and 6, I use Krenn's BME tag scheme for my discriminative approach. Krenn's results are a reference point for assessing my system's performance.

Krenn uses the CELEX dataset for her experiments, evaluating using 50-fold cross-validation. She reports results at the phoneme level: how many individual phones received the correct tag. Krenn's ONC model outperforms her positional model, scoring 98.34 percent phoneme accuracy, against only 93.56 percent. Krenn argues that the onset-nucleus-coda model beats out her positional model because it has less inherent ambiguity. With the BME tags, almost every phone can be assigned all three tags; with ONC tags, a large number of phonemes can assume only one possible tag.

Demberg [2006] also applies HMMs to the syllabification task, as a component of a larger German text-to-speech system. Although her TTS system only uses syllabification of letters, Demberg presents experiments on both German letters and phonemes. Demberg uses simple binary tags, requiring every letter to be labelled either N (Not at a syllable boundary) or B (Boundary). She applies a fourth-order HMM to the problem and additionally enforces the constraint that every syllable must contain a nucleus. To smooth the counts obtained from her training data, she applies Modified Kneser-Ney smoothing.

Demberg's results are very good; using 10-fold cross-validation on German CELEX she achieves 97.87 percent word accuracy on the letter domain. In the phoneme domain, she achieves 98.47 percent word accuracy[1]. Additionally, Dem-

[1]Schmid, Möbius, and Weidenkaff [to appear 2007] improve on Demberg's phoneme results by

berg is able to improve L2P performance in German from 73.4 percent to 74.4 percent word accuracy, a statistically significant improvement. More importantly, syllabification allows her to automatically generate stress patterns (syllable position is an important predictor of German word stress). This further increases her L2P word accuracy to 78.85 percent. I use Demberg's results as a point of comparison in my experiments.

Müller's PCFG-based Approaches

Müller [2001a] applies probabilistic context free grammars (PCFGs) to the syllabification of German phonemes. She first manually constructs a context-free grammar (CFG) of possible syllables. Then, she uses counts from a labelled training set to assign probabilities to the rules in the CFG. Armed with this probabilistic context-free grammar, Müller can then find all possible syllable parse trees for an input word. These parse trees entail the syllable boundaries for the word. The most likely parse tree, according to the grammar, is selected as the correct syllabification of the word. Müller also experiments with a number of linguistic annotations for her CFG rules in an attempt to improve accuracy. She tries using extra productions that capture whether a phone is a vowel or a consonant, or whether it is part of an onset, a nucleus, or coda. Her best-performing grammar contains productions for the syllable's rhyme (*e.g.* Syl $\rightarrow$ Onset Rhyme; Rhyme $\rightarrow$ Nucleus Coda), and captures the syllable's position within the word (*e.g.* Word $\rightarrow$ Syl.ini Syl; Syl $\rightarrow$ Syl.fin).

To evaluate her approach, Müller uses a large newswire corpus of 3 million words, looking up their pronunciations in CELEX. Ninety percent of the corpus is used as training data, while 10 percent is used for testing. The two sets are not disjoint: more than 90 percent of the words from the test set also appear in the training data. Under this set-up, her system's scores word accuracy of 96.49 percent.

Subsequently, Müller [2002] further augments her grammar by adding produc-

applying a fifth-order HMM and tuning their smoothing algorithm specifically for syllabification of German phonemes. Word accuracy increases to 99.85 percent with the higher-order Markov assumption.

tions for onsets and codas of different lengths. This produces an incremental improvement, scoring 96.88 percent word accuracy using the same experimental setup. Müller [2006] reprises the earlier work with a few variations. Rather than hand-constructing the CFGs, she exhaustively generates all possible syllables by systematically concatenating together all phonemes that occur in the language. This allows the technique to be easily applied to other languages. Many of the syllables in these generated grammars will never occur because they violate phonotactic constraints. Müller enforces the constraints by introducing grammars with more complex productions. For example, in her best-performing grammar, different production rules are applied to the onset and coda depending on the actual phoneme in the nucleus. That is, different rules will be used to build the tree for the word *string* than will be used to build the tree for the word *strong*.

There is not much to choose between the various instantiations of Müller's PCFGs. However, the 2006 work is most relevant to this thesis because of the evaluation methodology. She applies the method to both German and English phonemes, uses CELEX, rather than newswire corpora for her tests, and evaluates using 10-fold cross validation. Thus, her test words do not appear in the training data. Her best word accuracy score in English is 92.64 percent; in German it is 90.45 percent. I will use these results as a point of reference in my later work.

Bouma's WFST-based approach

Bouma [2002] uses finite state transducers (FSTs) to implement a hyphenation system for Dutch. He begins with a deterministic FST implementing a straightforward regular expression that essentially imposes maximal onset as its only guiding principle. On the Dutch orthography portion of the CELEX dataset, Bouma achieves 86.1 percent word accuracy with his deterministic implementation. In error analysis, he finds that most errors occur where morphology overrides maximal onset.

To overcome these morphologically-induced mistakes, Bouma uses transformation-based learning (TBL) to learn rules for correcting these mistakes without introducing new errors. These new rules are then composed into FSTs. By using 90 percent of the CELEX orthography word list (~261,000 training points), Bouma is able to extract 1,409 rules and improve his word accuracy to 98.17 percent. Even with only 10 percent of the CELEX dataset (deriving 264 rules), Bouma sees a very substantial increase in word accuracy — 95.34 percent.

Bouma's excellent results can be achieved very efficiently using transducers. Better still, TBL produces actual rules from which we can glean linguistic meaning. For example, a rule re-writing all instances of *i—st* as *is—t* will remove nearly ten percent of the errors in Bouma's test set. Meaningful rules are not available from my system, due to the large number of features I employ. However, it is not clear whether this FST-based approach would fare as well on English, a much less regular language than Dutch. English has so many exceptions that TBL may effectively devolve into creating a rule for almost every word in the language. Nonetheless, I will use Bouma's results as a benchmark for my Dutch results.

4.3 Other Previous Work on Syllabification

Zhang and Hamilton [1997] explore syllabification of English letters as a component of their L2P system. Their system, LE-SR, learns syllabification rules based on pattern matching. Rules are ranked based on the frequency they apply. Their final system comprises 1089 rules, although using just 22 of the most common rules produces better than 90 percent accuracy. The vast majority of their rules (77%) deal specifically with single-case exceptions. Overall, their performance is good, achieving 95.42 percent word accuracy in ten-fold cross-validation experiments on a NETtalk-derived dataset.

These results must be interpreted with caution. Although Zhang and Hamilton couch their technique as syllabification of letters, and although they incorporate LE-SR as a pre-processing step in L2P conversion, their techniques does not actually work on letters. The input to their system has already been 'chunked'. That is, letters that combine to make a single sound have already been merged, and silent letters have been removed. Moreover, letters have already been categorized as syllabic (vowels and syllabic consonants) or non-syllabic (other consonants). Thus, rather than syllabify the string *ro—guish*, LE-SR syllabifies the string *C S C S C*, where *S* represents syllabic entities and *C* represents non-syllabic entities. The letters *sh* and *ui* have been combined into single chunks. This is clearly an easier task than syllabifying raw letters, and is more comparable to syllabifying phonemes.

Kiraz and Möbius [1998] implement a weighted finite-state transducer (WFST) solution to the syllabification problem for phonemes in both English and German. The FST they describe is a component in Bell Labs' TTS system. The weights for the German WFST are learned from the German portion of CELEX; the weights for the English WFST are learned from English CELEX and Bell Labs' in-house pronunciation dictionary. A separate WFST is built for each of onsets, nuclei, and codas. These three transducers are then composed together into a deterministic transducer accepting syllables according to a regular expression that defines legal syllables in a given language. A word is then syllabified by a transducer that accepts one or more syllables, separated by syllable-boundary markers. Unfortunately, Kiraz and Möbius only perform a qualitative analysis of their system, rather than providing a quantitative measure of their performance.

Pearson, Kuhn, Fincke, and Kibre explore syllabification (and stress assignment) of English phoneme strings [Pearson *et al.*, 2000]. They compare two rule-based systems: the syllable module from the MITtalk TTS system, and Panasonic's in-house syllabification rule set. The actual rules applied in these two methods are not described. Pearson *et al.* compare these rule-based methods with a CART decision tree and Panasonic's in-house 'global statistics' algorithm. In the phoneme domain, the challenge is to determine where a consonant cluster should be divided into separate syllables. The Panasonic method categorizes every consonant cluster by its context: the cluster will be preceded and followed by some combination of a short vowel, long vowel, beginning of word, and end of word. Consonant clusters in test words are syllabified based on the most probable boundary location found in training data. The four methods are assessed using the 19K Cybertalk dictionary; half the words are used for training and the other half for testing. The MIT-talk rule system scored 61.6 percent word accuracy, while the Panasonic rules achieved 83.2

percent. Both data-driven methods scored better, with the decision tree syllabifying 92.2 percent of words correctly and the global statistics method getting 96.2 percent of words right. Pearson *et al.* also find that they can improve their syllabification results by a small amount using phoneme strings with stress patterns as input. Their results are, unfortunately, not comparable with mine, because they are based on a proprietary gold standard.

MacKinney-Romero and Goddard [2006] apply decision trees to Spanish letters. Spanish is a very transparent language. The pronunciation of the orthographic form of a Spanish word is deterministic, which means syllabification of Spanish letters and phonemes should be of a similar level of difficulty. MacKinney-Romero and Goddard extract words from literary and newswire corpora and manually assign letters to be either onsets, nuclei, or codas. Using a window of three letters around a focus character as input, they automatically construct decision trees from the training data. These resulting trees are not very complex — binary trees with a maximum depth of 4 — but they perform very well in Spanish. On 1,000 randomly selected words not appearing the training corpus, the trees score 96.8 percent word accuracy. They further attempt to apply their method to English and Italian. However, their gold standard syllabifications are the output from Hammond's OT-based parser, which was less than 40 percent accurate in Marchand and Damper's experiments [Marchand *et al.*, to appear]. With such a flawed gold standard, the reported English letter accuracy score of 97.3 percent is meaningless. The Italian gold standard is also derived from an existing automatic technique, which makes the makes the letter-accuracy score for the Italian tests (99.3%) similarly difficult to interpret.

Unsupervised Techniques

Müller, Möbius, and Prescher explore the use of Expectation-Maximization (EM) as the basis for a clustering algorithm [Müller *et al.*, 2000]. They experiment with both a three-dimensional and five-dimensional clustering model, applying it to both English and German phonemes. The three-dimensional model uses syllable onset, nucleus, and coda to determine the cluster, while the five-dimensional model adds syllable emphasis and the syllable's position in the word. They experiment over a range of clusters, from one single class for all syllables, up to 200 classes of syllables. The German models are further incorporated into a letter-to-phoneme system. After all possible phone sequences are generated from letters and parsed by a context-free-grammar, the highest-probability sequence of syllables is selected as the L2P output. Compared to a baseline of probabilities counted directly from the training data, the learned EM clusters improve L2P accuracy. However, these improvements are in the context of rather low initial accuracy — their best reported L2P accuracy is 75 percent.

This EM-based method is problematic as a syllabification technique. Generally speaking, EM and clustering are used for unsupervised approaches without training data. However, Müller *et al.*'s technique relies on being able to extract syllables from words, and therefore requires labelled training data (they use CELEX to look up words appearing in newswire text). If we have access to labelled training data, it is better to use some sort of supervised approach. Thus, this work is better viewed as a test-bed for multivariate EM clustering rather than syllabification *qua* syllabification.

In subsequent work, Müller [2001b] does explore a true unsupervised approach to syllabifying German phonemes. She uses the same PCFG-based approach described in Section 4.2, but instead of using counts derived from labelled training data, she estimates the probabilities for her grammar using EM. Müller's best grammar achieves syllable accuracy of around 90 percent on pronunciations extracted from newswire text. This is far behind the supervised approach, which scored better than 96 percent word accuracy, a far more difficult metric. Müller then attempts to use only the syllable grammar to perform the L2P task. To do this, she expands her PCFG to include productions that map phonemes to letters. The results are not impressive: her best model achieves only 42.5 percent word accuracy.

Goldwater and Johnson [2005] also explore using EM to learn the structure of English and German phonemes in an unsupervised setting, following Müller in modelling syllable structure with PCFGs. They use two different models: the positional model from Müller's work, and a new bigram model. The bigram model's productions make every syllable expansion depend on the previous syllable.

Goldwater and Johnson attempt to estimate the parameters for their maximum likelihood approach using EM. They initialize their parameters using a deterministic parser implementing sonority and maximum likelihood. On its own, this categorical parser performs fairly well: 92.7 percent of German phone strings and 94.9 percent of English phone strings are correctly syllabified. However, these results are somewhat inflated because a large proportion of the test set is monosyllabic and the training and test sets are not disjoint. Running EM to convergence on the bigram model, after initializing the parameters based on the output of the categorical parser, produces an increase of several percentage points, to 95.9 percent in German and 97.1 percent in English. The same procedure on the positional model actually decreased the word accuracy. Goldwater and Johnson speculate that this is because the positional model does not allow EM to generalize onsets and codas found word-initially and word-finally to consonant clusters found word-medially. In effect, the positional model precludes EM from learning the legality principle. They conclude that models that produce improvement in a supervised setting are not necessarily optimal in an unsupervised one.

4.4 Summary

Many different authors have tackled automatic syllabification, using a variety of different techniques. Nonetheless, the problem remains unsolved. Particularly in English, where syllabification patterns are less regular than in German or Dutch, there is considerable scope for improving performance. Discriminative approaches are notably underrepresented in previous work. The most successful approaches use some version of lazy learning (as in SbA or EBG), or HMMs. By contrast, my system uses supervised discriminative training to achieve state-of-the-art performance.

Chapter 5

Syllabification of Letters

This chapter shows how discriminative training can be applied to automatic syllabification of letters. In particular, I formulate syllabification as a tagging problem and apply the SVM-HMM learner to the task. Given the SVM-HMM formalism, the crux of the task is creating a tagging scheme that produces good results, and deriving a set of features that capture the relationship between input words and their desired labellings. This chapter describes that process, and presents experimental results on the final system.

Section 5.2 outlines a number of different tag sets, and selects two tag sets for further experimentation. Section 5.3 describes my system's emission features and how they are chosen. There is necessarily some overlap here: we cannot train an SVM without selecting both features and a tag set. Furthermore, the performance of the tag sets depends on the features selected, and vice versa. In practice, I tested all combinations of feature sets and tag schemes before settling on the configuration used to obtain the results presented in Section 5.4.

However, both tag sets and features selected (and indeed, the use of SVM-HMM itself) depend on the data available. Hence, I will first turn my attention to the datasets used for training and testing.

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.

At the outset of this research, I primarily used the NETtalk dataset for development. The NETtalk corpus is so-called because it was developed by Sejnowski and Rosenberg [1988] for their NETtalk text-to-speech system. The corpus is a list of 20,008 English words and their phonetic transcriptions, as found in Webster's Pocket Dictionary. In addition to the letters and their associated phones, NETtalk also lists stress and syllable structure for every word. The dictionary maintains a one-to-one alignment between letters and their phonemes, which allows this syllable and stress information to be applied to both the orthographic and phonemic representation of a word. My early work on this subject used this dataset largely because it was the one used by Marchand and Damper for their work on syllabification and letter-to-phoneme systems ([Marchand and Damper, 2005; Marchand *et al.*, to appear]). However, it soon became apparent that there are significant problems with NETtalk as a dataset for syllabification.

NETtalk is riddled with some truly bizarre syllabifications, such as *be—aver*, *dis—hcloth*, and *som—ething*. These 'gold standard' syllabifications are obviously problematic. *hcl* is an invalid onset in English because it is unpronounceable. The syllables *aver* and *ething* each have two vowels separated by consonants, and look as though they should be subdivided into two syllables (*i.e. a—ver* and *e—thing*). Indeed, the word *aver* appears in the NETtalk dataset, syllabified as *a—ver*. I consulted five other dictionaries[1], and all five list *bea—ver*, *dish—cloth*, and *some—thing* as correct.

The examples I have listed are not spurious, or attributable to human error in compiling the dataset. Rather, they are indicative of a systematic approach to syllabification in NETtalk. Because orthographic syllabifications are mapped directly

[1]Merriam-Wester Online, Dictionary.com, Encarta Dictionary, CELEX, OS X Dictionary.

from the phonetic domain, whenever a letter is silent (like the *e* in *something*), or two letters combine to create single phoneme (like the *sh* in *dishcloth*, NETtalk inserts a null symbol in the phoneme string. The phonemic gold standard syllabification almost always places the syllable boundary before the null. For strings of phonemes, this is not a problem; the null has no sound by definition, so it can equally go on either side of the boundary. However, when those same syllabifications are mapped directly to the letter domain, we get counter-intuitive syllabifications like the ones listed above.

Obviously, these bizarre syllable breaks make it difficult for a data-driven method to be successful. Seeing both *a—ver* and *be—aver* in the training data sends the SVM in conflicting directions. Scores will also be artificially lowered if a syllabification like *some—thing* is scored as incorrect. Most worryingly, the NETtalk gold standard syllabification is likely to introduce new errors into a letter-to-phoneme system. An L2P system is likely to correctly predict that the letters *sh* in *dishcloth* map to a single phoneme because it is a common occurrence and the alternatives are unpronounceable. However, if that same L2P system is told that there is a syllable break between the *s* and *h*, correct prediction is less likely: a single phoneme cannot have a syllable break in the middle of it.

Notwithstanding these shortcomings, I still perform some experiments on the NETtalk database, largely because Marchand, Damper, and Adsett use it for their comparative work in English [Marchand *et al.*, to appear]. However, I also argue that NETtalk is entirely unsuitable for the syllabification task. Results on other datasets are a much better measure of the efficacy of a syllabification task, and a better indicator of whether syllabification can improve letter-to-phoneme systems.

Because of NETtalk's deficiencies, the main dataset for this work is the CELEX dictionary [Baayen *et al.*, 1995]. CELEX is a product of the Max Planck Institute for Psycholinguistics in the Netherlands. The dictionary consists of three languages (English, Dutch, and German), and specifies both orthographic and phonetic forms for words. The English section of CELEX is compiled from the Oxford Advanced Learner's Dictionary (1974) and the Longman Dictionary of Contemporary English (1978). It contains 160,595 English words of British English. The German portion of CELEX is based on the Bonnlex and Molex computer dictionaries, as well as a German spelling lexicon. It contains 365,530 words. The Dutch CELEX is derived from Van Dale's Comprehensive Dictionary of Contemporary Dutch (1984) and the Groene Boekje Word List of the Dutch Language (1954), and most of the dictionary entries from the Institute for Dutch Lexicology's 42.4 million-token corpus. It contains 381,292 entries.

I perform some pre-processing of the CELEX dataset for my work here. Only words containing at least two letters are retained. All homographs are removed. There are 23K homographs in English CELEX. Most of the time, the syllabification is the same for all instances of the homograph, such as the seven different CELEX entries for the orthographic form *cross*. However, in a small minority of cases (about 0.3 percent of homographs, or 0.1 percent of the final word list), homographs have different syllabifications, such as *learned*, the past-participle, and *learn—ed*, the adjective. In these cases, one of the possible syllabifications is selected at random as the gold standard; only that syllabification is considered correct during experiments.

I also remove all multiple-word and hyphenated entries, as they are superfluous. Given that we have an entry for *zebra* and another for *crossing*, it is unnecessary to keep *zebra-crossing* as well. On the other hand, the compound word *crosswalk* is retained. I also remove words containing apostrophes (the vast majority of which are possessives), and any diacritics are converted into single-character codes to facilitate processing. Finally, in English I remove any unpronounceable abbreviations, like *lbw* or *bbc*. The final English dataset contains 66K words. 52K of the removed words are homographs; an additional 41K are multi-word or hyphenated entries. Thus, almost 99 percent of the removed words are duplicates of some form. When similar pre-processing is performed for Dutch and German, the datasets have 298K and 311K words, respectively.

I develop my approach using a small subset of English CELEX. In both Sections 5.2 and 5.3, I report results on a randomly selected development set of 5K words. During development, SVM-HMM is trained using another 10K randomly selected words not appearing in the 5K set.

5.2 Choosing a Tag Scheme

The most obvious tagging scheme for automatic syllabification is a binary assignment: a letter is either before a syllable boundary, or it is not. I call this tagging scheme **NB Tags** because every letter is either at a syllable boundary (⟨B⟩), or not (⟨N⟩). Using NB Tags, the gold standard labelling for the word *syl—lab—i—fy* is ⟨N N B N N B B N N⟩, indicating that a syllable boundary should be inserted after the third, sixth, and seventh letters. During development I also experimented with labelling all word-final letters with ⟨B⟩. The legality principle tells us that all possible syllable-final clusters are also found word-finally. The hope was that explicitly marking boundaries at the end of words would help the SVM generalize word-medial boundaries. However, experiments on the development set found that not explicitly indicating word-final boundaries produced better performance. Binary tagging schemes for automatic syllabification are not new. Marchand and Damper employ the same basic idiom for Syllabification by Analogy. The tagging scheme is implicit in a number of previous implementations ([Daelemans and van den Bosch, 1992], [Bouma, 2002]), and has been done explicitly in both the orthographic [Demberg, 2006] and phonetic [van den Bosch *et al.*, 1998] domains.

I also experiment with a very similar tag scheme, **IMF Tags**. Introduced by Krenn as BME tags, IMF tags differentiate between the first letter after a boundary, and the second and subsequent letters, by explicitly marking the initial (⟨I⟩) and final (⟨F⟩) letters in the syllable. All intervening tags are marked as being in the middle (⟨M⟩) of the syllable. Under this tag scheme, *syl—lab—i—fy* is tagged as ⟨I M F I M F F I M⟩.

A clear shortcoming of NB and IMF tags is that they encode no knowledge about the length of a syllable. Intuitively, this seems like important information. Syllables tend to be short — most English syllables contain four or fewer letters. Because the SVM-HMM encodes a first-order Markov assumption, NB and IMF Tags do not even allow the classifier to realize that it is predicting something unusual. Thus, when using NB tags, we get output like **heroines* (instead of *her—o—ines*) and **wellsprings* (*well—springs*). The problem is less pronounced for IMF tags, but we still see **makeshift* (for *make—shift*) and **achieved* (for *a—chieved*).

To combat errors like these, I introduce **Numbered NB Tags**[2], which simply numbers the ⟨N⟩ tags. Thus, the gold standard labelling for *syl—lab—i—fy* becomes ⟨N1 N2 B N1 N2 B B N1 N2⟩. Because higher-numbered ⟨N⟩ tags will be comparatively much rarer than ⟨N1⟩ or ⟨N2⟩, we have effectively introduced a bias in favour of shorter syllables. Tags like ⟨N5⟩ and ⟨N6⟩ will be postulated only when the evidence is particularly compelling. When using Numbered NB tags, we get *make—shift* and *a—chieved*; *well—springs* and **her—oines*. The last example is still incorrect, but is at least linguistically plausible.

Numbered NB Tags capture a key feature of a syllable: its length. However, like IMF and NB tags, it is a purely positional scheme. The tags say nothing about the function a given letter is performing within a syllable. To whit, these positional tags cannot identify which letter(s) in a syllable are acting as a nucleus. The implication of this shortcoming is that Numbered NB tags, NB tags, and IMF tags do not force syllables to contain a plausible nucleus. Using positional tag sets, the SVM-HMM will see no contradiction in predicting a syllable that contains only a single consonant. Consequently, we see output like **lim—b—er* (instead of *lim—ber*) and **par—t—ner—ships* (*part—ner—ships*). .

To prevent errors of this nature, I implement **ONC tags**. ONC tags label every letter of a word as being part of either an onset, nucleus, or coda. Under this tagging scheme, the gold standard of *syl—lab—i—fy* is ⟨O N C O N C N O N⟩. Given this sequence of tags, a regular expression can insert the syllable boundaries in the correct locations[3]. Such a tagging scheme has previously been attempted in the phoneme domain [Krenn, 1997], but to my knowledge has not been previously attempted in the letter domain. This is perhaps not without reason. Training data labelled with these tags is not available, and generating the tags deterministically

[2]Numbering NB tags is exactly equivalent to numbering IMF tags, so there is no need to explore both options.

[3]I abstract. In practice, we must always number the nuclei tags; otherwise, we will be unable to determine whether tag sequences like ⟨O N N C⟩ represent a single syllable with two vowels (as in *reap*), or two syllables (as in *re—up*).

from syllabified word lists is non-trivial.

Consider the word *syl—la—ble*, for which we generate ONC tags ⟨O N C O N O O N⟩. An obvious difficulty for English is the letter *y*, which sometimes acts as vowel and sometimes as a consonant. In actual practice, this is an easy issue to solve. All *y*s that are both preceded and followed by a consonant (as in the first syllable of the example) must be vowels by default. Otherwise, the letter *y* is treated as a consonant.

A more difficult problem can be found in the last syllable of the example. Our script for generating ONC tags chooses the labels ⟨O O N⟩, effectively setting this to be the gold standard. However, these labels are not, strictly speaking, correct. The letter *e* is actually silent here, and the *l* is acting as a syllabic consonant; that is, the *l* represents the nucleus of the syllable. The letters *l*, *m*, *n*, and *r* can all act as syllabic consonants in English. To deal with this, these four letters are allowed to be labelled as nuclei if there are no vowels in the syllable. The vagaries of English spelling mean that syllabic consonants are often accompanied by silent letters. Thus, there is a certain degree of noise in the gold standard labelling. I should stress that this noise does not affect the accuracy of the syllabifications derived from the labellings; the tags are generated specifically to be consistent with the syllable breaks as provided in the dictionary.

These problems with ONC tags are a drawback of the tag set. Across languages, different letters can act as vowels; without knowing which letters can act as vowels in a particular language, ONC tags cannot be generated. Accordingly, I do not experiment with ONC tags for Dutch or German. Nonetheless, I pursue the tag scheme for English because it does capture the internal structure of the syllable, which is a positive attribute. Using ONC tags, SVM-HMM no longer produces syllables consisting of a single consonant. Moreover, I believe ONC tags can easily be adapted for other languages by anyone with enough familiarity with those languages to know their idiosyncrasies.

There is also considerable scope for variation within ONC tags. I explore numbering the ⟨O⟩ and ⟨C⟩ tags: **Numbered ONC Tags** will label the word *strengths* as ⟨O1 O2 O3 N1 C1 C2 C3 C4 C5⟩. **Reverse ONC Tags** also number the tags, but the coda tags are numbered from right-to-left instead of left-to-right. Hence, *strengths* is labelled ⟨O1 O2 O3 N1 C5 C4 C3 C2 C1⟩. Since syllable codas can vary in length, Reverse Numbered ONC Tags ensure that the same tag always appears as the last label in the coda. Another option is to provide a special label for silent letters. Thus, in a word like *blade*, where the final *e* is silent, the gold standard labelling is ⟨O O N C E ⟩. These **ONCE Tags** can, of course, be numbered or reverse-numbered.

With all of these ONC tags, a syllable break is not represented by a single tag. It takes a combination of two tags (*e.g.* adjacent C and O tags) to denote a syllable boundary. I also explore an ONC tag set that models syllable breaks explicitly using a single tag. Under the labelling scheme **Break ONC Tags**, the word *lev—i—ty* is tagged as ⟨O N CB NB O N⟩. The tag ⟨NB⟩ indicates a letter is both part of the nucleus and before a syllable break, while the the tag ⟨N⟩ represents a letter that is part of the nucleus, but in the middle of a syllable.

5.2.1 Tag Set Comparisons

To determine the best tag set, I ran a series of experiments on the development set using my best feature set (see Section 5.3.)

Primarily, I measure my performance using word accuracy, which reports how many words are given the same syllabification as the gold standard. Note that this is not the same as words where all the letters are assigned the correct tag, as the syllable boundaries may be correct even if the tags are not.

I also present a metric I call syllable break accuracy. Syllable break accuracy measures the proportion of tags that produce a correct syllable boundary decision. It is related to simple tag accuracy, but allows for better comparison across tag sets. Consider the word *side—show*, with the gold standard Reverse ONC tagging ⟨O1 N1 C2 C1 O1 O2 N1 C1⟩. If it has been mislabelled ⟨O1 N1 C3 C2 C1 O1 N1 C1⟩ (**sides—how*), the tag accuracy will be 4/8 = 50 percent. If the same mistake is made using the ONC tag scheme, where the gold standard is ⟨O N C C O O N C⟩, the incorrect tag sequence ⟨O N C C C O N C⟩ gives a tag accuracy of 7/8 =

87.5 percent. By contrast, syllable break accuracy only penalizes incorrect tags that cause an error in the syllabification, so it remains constant across tag sets. Under both Reverse ONC and ONC Tags, syllable break accuracy is 87.5 percent.

Positional Tags

Table 5.1 presents the results for all non-ONC tagsets. We see that NB tags lag considerably behind IMF and Numbered NB tags in terms of word accuracy. The poor performance of NB tags is most likely due to the weakness of the signal coming from the emission features. In English, a syllable break can be drawn after almost every letter, depending on the context. Hence, every letter can take on either tag with almost equal probability. This is not the case with either IMF or Numbered NB tags. Vowels often appear in the middle of a syllable, and consequently are much more likely to have N2 or N3 tags, or M tags. Consonants take on ⟨N1⟩ or ⟨I⟩ labels with greater probability.

Tag Set	Word Accuracy	Syllable Break Accuracy
NB Tags	82.56	96.58
IMF Tags	85.55	96.70
Numbered NB Tags	86.61	96.87

Table 5.1: Development set results for positional tag sets.

To illustrate this phenomenon, I have graphed the weights derived from training SVM-HMM with a very restricted feature set: only the central focus character. Figure 5.1 plots the absolute weight value vector learned for a representative sampling of letters under NB Tags. We see that for NB tags, the weight vector assigns almost equal weight for all letters, indicating that there is nothing to differentiate the letters. Meanwhile, the ⟨N⟩ and ⟨B⟩ tags have the same magnitude of weight, but different signs. Because the learner cannot distinguish between the features (in this case a single letter), it will simply postulate the more frequent ⟨N⟩ tags. This is exactly what we do not want to see in a machine learning problem: the signal is the same for all cases, so it is difficult (if not impossible) to learn anything. Indeed, the word accuracy for NB tags with this restricted feature set is only 12 percent — not coincidentally, the same as the proportion of monosyllabic words in the test set.

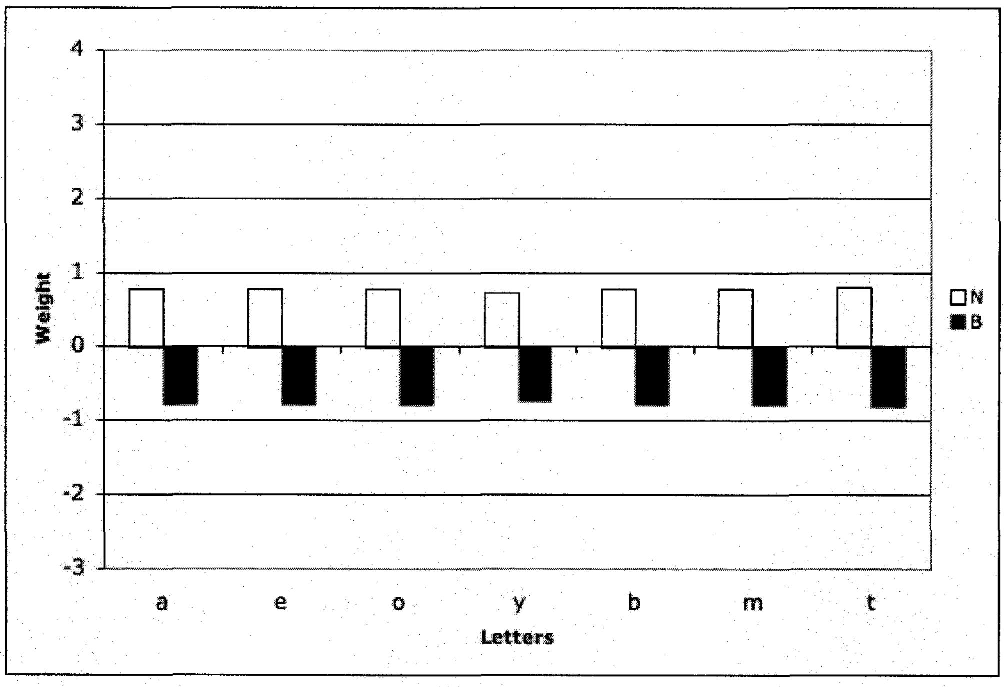


Figure 5.1: Weights learned by SVM-HMM using NB tags.

Compare this to the equivalent graphs for Numbered NB tags, shown in Figure 5.2. Here, we get a much better story. There is significant variation across letters, and the magnitude of the weight varies widely across potential tags. Vowels tend to have a strong, positive signal for ⟨N2⟩ and ⟨N3⟩ tags, indicating that these letters are most often found in the middle of a syllable. Meanwhile, consonants have strong, positive weights for ⟨N1⟩ tags. Word accuracy increases accordingly: using the same feature set with the numbered NB tags increases word accuracy to 40.5 percent.

ONC Tags

Table 5.2 presents the results for the various versions of ONC tags. On the whole, there is not much to choose between any of the variations, although Break ONC tags are more than half a percentage point better than the others. More surprisingly, ONC tags fail to out-perform the best positional tags. At 86.61 percent word accuracy, Numbered NB tags are slightly ahead of all of the ONC tags except Break ONC —

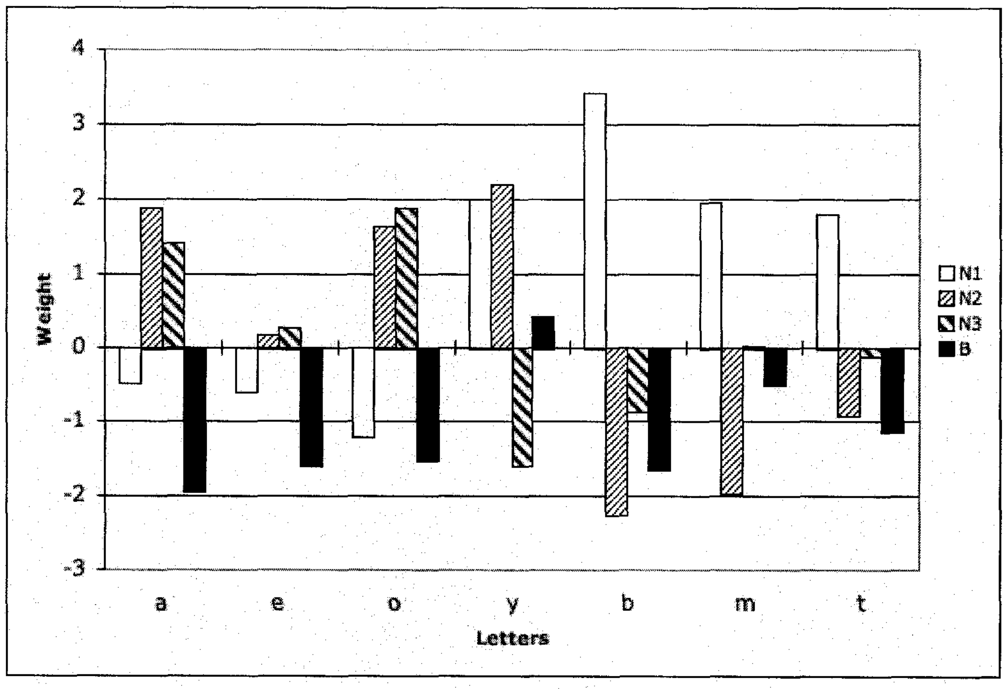


Figure 5.2: Weights learned by SVM-HMM using Numbered NB tags.

and Break ONC tags are a hybrid of ONC tags and NB tags.

Occam's razor tends to favour Numbered NB tags. There is a fair amount of overhead required to generate ONC tags in the letter domain, and they cannot be directly applied to new languages without modification. Numbered NB tags achieve comparable results, and they are much simpler to generate and work with.

Tag Set	Word Accuracy	Syllable Break Accuracy
ONC tags	86.21	96.65
ONCE tags	86.21	96.63
Numbered ONC tags	86.27	96.67
Reverse ONC tags	86.09	96.61
Break ONC tags	86.94	96.89

Table 5.2: Development set results for ONC-based tag sets. ONCE versions of Numbered and Reverse ONC tags were also tested, but did not measurably change the results.

However, an examination of the errors made by both tag sets reveals that Numbered NB tags tend to produce more damaging errors than Break ONC tags. For example, syllabifications like **an—k—let* and **am—ps* are clearly incorrect because they contain syllables with no vowels. Break ONC tags produce 5 such errors in the development set. Conversely, Numbered NB tags produce 32 of these errors — more than six times as many. ONC tags are more resistant to this type of error because they implicitly require every syllable to contain a nucleus. Given that Break ONC tags also achieve slightly higher accuracy than Numbered NB tags, there is a case to be made for preferring Break ONC tags in spite of their greater complexity.

Moreover, it turns out that the various tag sets achieve similar word accuracy scores by correctly syllabifying different subsets of the test set. I do not want to overstate the differences — the various tag sets produce the same syllabifications in about 90 percent of cases. Nonetheless, I consider the variation in the tag sets to be worth exploring. For instance, if we consider the output from ONC tags, Break ONC tags, and Numbered NB tags, we find that at least one of those tag sets is correct 91 percent of the time. Thus, given an oracle to tell us which output to choose, we could decrease the error rate by a third. I have no oracle, and simple solutions like a majority vote are ineffective: when two tag sets agree, they are about as likely to agree on the wrong answer as the right one, leaving accuracy unaffected. However, combining multiple tag sets is a potential avenue for further improvements.

It is difficult to choose between Numbered NB and Break ONC tags on their merits. In addition, it might be advantageous to combine the two tag sets in some fashion. Accordingly, I will report results for both Numbered NB and Break ONC tags in experiments on English CELEX. For other languages, (and for NETtalk data, whose bizarre syllable breaks make ONC tags difficult to apply), I will only apply Numbered NB tags to the problem.

5.3 Feature Engineering

In the SVM-HMM framework, emission features are generated on individual letters within words. I use aspects of the input to help predict the correct tag for a given letter. These emission features are distinct from transition features, which capture the probability of one tag following another. Transition probabilities are calculated from counts of tag pairs in the training data; they are automatically generated by the SVM-HMM package's infrastructure.

Using a traditional HMM, our features are limited. Given a the letter *a* in a word like *syllable*, we generally consider only that it is an *a* being emitted, and assess potential tags based on that single letter. The SVM framework is less restrictive: we can include *a* as an emission feature, but we can also include features indicating that the previous letter is an *l* and the following letter is a *b*. In fact, there is no reason to confine the feature set to one character on either side of the focus letter.

The window size is the number of letters on either side of the focus character that are included in the feature representation. To determine the optimal window size, I experiment with increasingly larger window sizes on the development set, using only unigram features. Special beginning- and end-of-word characters are appended to words to ensure that every letter has sufficient characters before and after. I find that word accuracy is best using a window size of 11: five characters before and after the focus character. Expanding the window size beyond that causes overfitting. Figure 5.3 shows how the window size affects word accuracy. For clarity, only the Break ONC tags are plotted; the results for other variants of ONC tags follow a similar learning curve.

Window Size	Context	Word Accuracy
3	$l_{i-1}l_il_{i+1}$	58.95
4	$l_{i-1}l_il_{i+1}l_{i+2}$	61.56
4	$l_{i-2}l_{i-1}l_il_{i+1}$	60.42
5	$l_{i-2}l_{i-1}l_il_{i+1}l_{i+2}$	63.08

Table 5.3: Symmetric vs. Asymmetric Windows

In the final tested system, I use a symmetric window, with five characters on either side of the focus character. However, during development I also experimented with asymmetric windows, including more characters after the focus character than before, or vice versa. Empirically, the letters after the focus character seem to be more important for prediction accuracy. In Table 5.3, I present the word accuracy values for the Numbered ONC tags for single-letter features at small window sizes. Compared to a baseline of a symmetric window of size three, adding a sec-

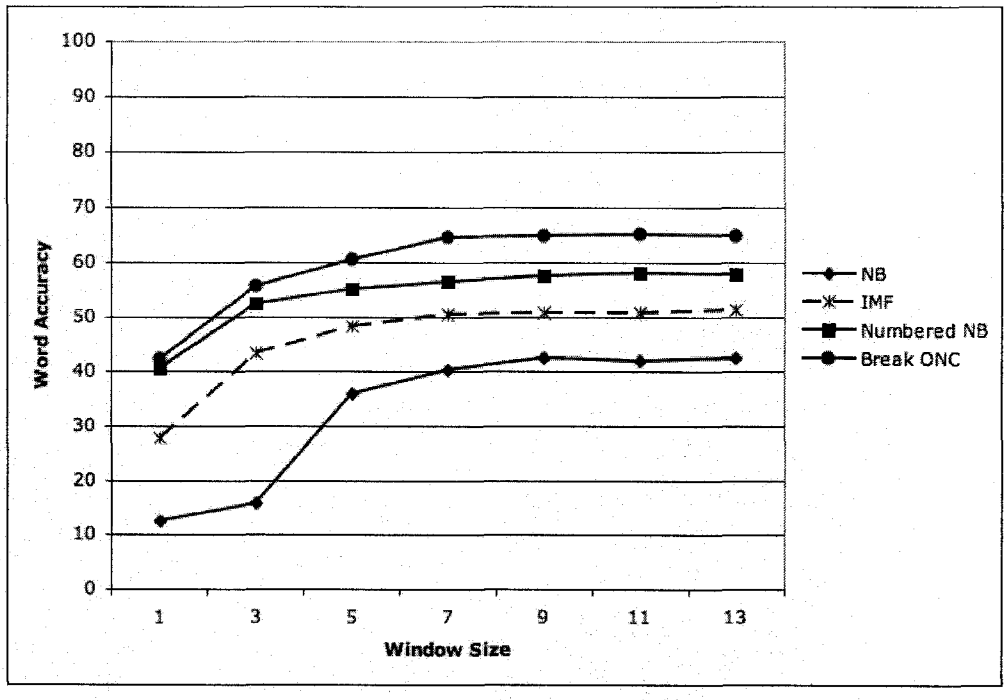


Figure 5.3: Word accuracy as a function of the window size around the focus character.

ond character after the focus letter (line 2) creates a fairly substantial performance boost, increasing by two and a half percentage points. Conversely, adding a second character before the focus letter (line 3) improves performance by only 1.5 points. However, adding a character both before and after the focus letter is by far the best, showing an increase of over four percentage points above the three-character window. This general pattern is typical of all tag sets and window sizes, although the differences are less pronounced as the learning curve starts to level off.

SVM-HMM uses a linear kernel, and so is effectively a linear classifier. Consequently, any important conjunctions of features must be represented explicitly. Inherently, language is non-uniform and letters generally appear in predictable patterns across words. For instance, the bigram *bl* frequently occurs within a single English syllable (*e.g. blow*, *bliss*, *blah*, *etc.*). Conversely, the bigram *lb* generally straddles two syllables (*e.g. al—bum*, *sail—boat*, *el—bow*, *etc.*). Similarly, a four-gram like *tion* or *ates* often forms a syllable in and of itself. Hence, in addition to the single-letter features outlined above, I also include higher order N-grams in the feature representation.

Using the development set, I tested feature representations that include all the bigrams, trigrams, fourgrams, and fivegrams within the 11-character window. Figure 5.4 shows that word accuracy improves by more than 20 percentage points with the addition of N-grams. These are features that would be unavailable using the simple HMM framework, because they are not conditionally independent. Note that when using only unigram features, ONC tags are clearly better than Numbered NB tags, which are in turn better than IMF tags. As we add higher-order features, the differences between the tag sets begin to disappear.

The optimal feature configuration is not the same for all tag sets — for instance, Reverse ONC tags start to overfit if the feature set is extended beyond fourgrams. However, for both Numbered NB tags and Break ONC tags, the best feature set includes all unigrams, bigrams, trigrams, fourgrams, and fivegrams. It is also not obvious that the optimal window size I determined for unigrams will hold for the fivegram case. However, experiments on the development set confirm that including all N-grams that occur in an eleven-character window produces better accuracy than other window sizes. Thus, all N-grams up to five occurring in an eleven-character window make up the optimal feature set, both for the experiments in the following section, and the development set experiments presented in Section 5.2.

Over the course of this work, I tried two different ways of representing N-gram features. Initially, the feature space included an exhaustive list of all possible N-grams. This makes for an extremely large feature space, as there are 531,441 possible four-grams, and eight four-grams within the set window size. This makes for 4.2 million features for fourgrams alone. This is clearly inefficient, as most of the 531,441 possible fourgrams will never appear in English (*qxzp*, for example). Moreover, it precludes experimenting with fivegrams for technical reasons —∼72 million features is computationally intractable. Consequently, in the final tested system, I gather all the N-grams actually appearing in the training data, and generate a feature for each seen N-gram. During testing, all N-grams that were not seen during testing map to a single 'unseen N-gram' feature. The drawback of this approach is that models learned on different training data are not interchangeable, as they will have different feature representations. However, the reduction in the

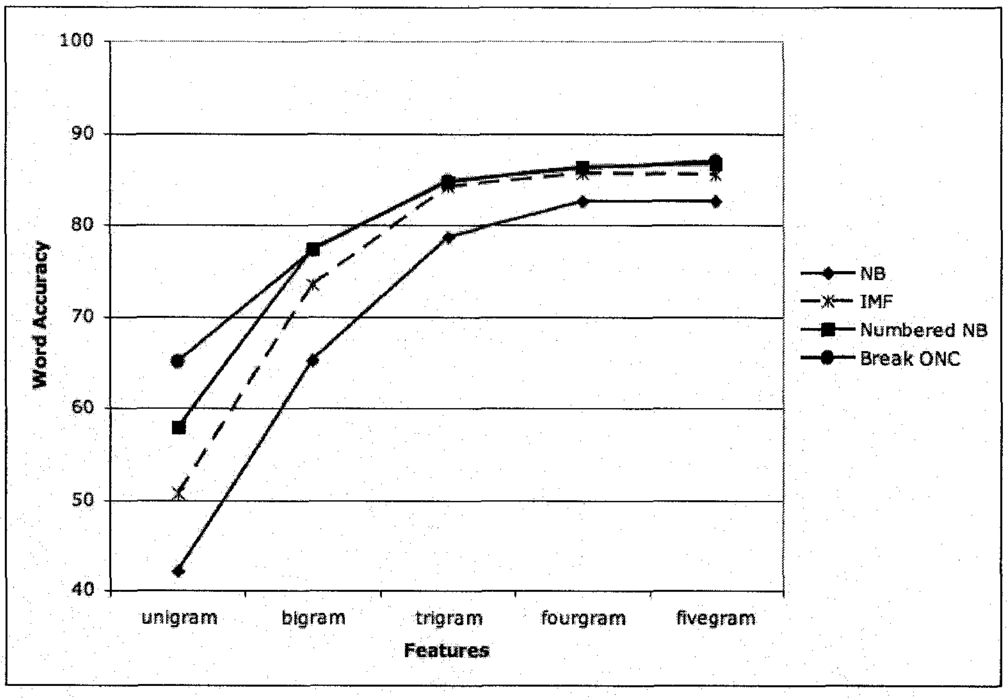


Figure 5.4: Word accuracy as a function of N-grams incorporated into the feature set.

feature space is dramatic, as the number of seen 4-grams in the whole of English CELEX is only 32,835. Adopting this representation keeps the number of features under 900K.

In addition to N-gram features, I also experiment with linguistically-derived features. Intuitively, we expect features like whether a letter is a consonant or a vowel to be important. However, experiments on the development set showed that performance was essentially unaffected by these features. Even more sophisticated features, such as probabilistic representations of the manner of articulation of a letter, produced only incremental gains (a few hundredths of a point). These linguistically derived features are so unhelpful because the SVM can learn generalizations from the N-gram features alone. Moreover, these linguistic features make my approach more complex and detract from its language-independence. Consequently, I do not explore this avenue any further.

Recall from Section 3.2 that the cost parameter (or c value) of the structured SVM allows for a trade-off between the accuracy on the training set and the complexity of the weight vector. This parameter needs to be tuned for optimal performance. I tested c values of 0.001, 0.01, 0.1, 1, and 10 on the development set, and found 0.1 to be optimal for most configurations (although some of the unigram feature sets did better with a different c parameter). I do not tune the ϵ parameter for the syllabification task, because values of ϵ lower than 0.5 have been found to not improve prediction accuracy [Herbst and Joachims, 2006]. All results reported in this chapter are based on a c parameter of 0.1 and an ϵ of 0.5.

5.4 Experiments and Results

All results in this section are from models trained on the optimal feature set of fivegrams across an 11-character window.

5.4.1 Comparison with SbA

My first experiment provides a direct comparison between my system and Syllabification by Analogy (SbA), the existing state-of-the-art for English orthographic syllabification [Marchand *et al.*, to appear]. After personal communication with Marchand, I trained and tested my system using the same training and test sets that he used for his results.

System	Word Accuracy	Syllable Break Accuracy
SVM-HMM – Break ONC	89.99	97.58
SVM-HMM – Numbered NB	89.45	97.49
Syllabification by Analogy	84.97	96.04

Table 5.4: Performance of SVM-HMM and SbA on common training and test sets drawn from CELEX.

Table 5.4 reports the results of this comparison on the CELEX dataset. For these experiments, both Marchand and I use a 14K training set and a 25K test set. The 14K training set was selected to be approximately the same size as the training data used for NETtalk. Our NETtalk experiments use a 13K training set and a 7K test set. Table 5.5 presents the NETtalk results.

System	Word Accuracy	Syllable Break Accuracy
SVM-HMM – Numbered NB	81.74	94.99
Syllabification by Analogy	75.56	92.27

Table 5.5: Performance of SVM-HMM and SbA on common training and test sets drawn from NETtalk.

SVM-HMM fares very well in this direct comparison. Relative to SbA, SVM-HMM reduces the error rate by 33.3 percent on CELEX, and by 25.8 percent on NETtalk. Moreover, SVM-HMM is more efficient, making it much more attractive for inclusion in an actual L2P system. Once a model is learned, my system can syllabify 25K words in about a minute, while SbA requires several hours. SVM-HMM training times vary depending on the tag set and dataset used. Training the Numbered NB tag set on 14K training examples takes approximately two hours on a single-processor P4 3.4 GHz processor. ONC tag sets train in about half that time. In either case, of course, training time is a one-off cost.

5.4.2 Learning Curve

My second experiment tracks SVM-HMM's learning curve as I increase the amount of training data available to it. Using a very small held-out test set of 5,000 randomly selected words, I gradually increase the training set from 1K to 60K. Figure 5.5 shows how performance increases.

As is evident, larger amounts of training data greatly improve performance. However, this curve must be interpreted with caution. Language is very productive, and many words will re-appear in a number of different forms. Thus, a more realistic representation of how Structured SVM will fare on truly unseen words like proper nouns must ensure that none of the test words appear in the training set in another form.

Fortunately, in addition to its word lists, CELEX includes lists of lemmas. Lemmas are like headwords in a dictionary, using one instance to represent the related forms of a various word [Crystal, 2003]. Thus, the CELEX lemma list will contain *educate*, but not *educates* or *educating*. However, it will still contain other parts

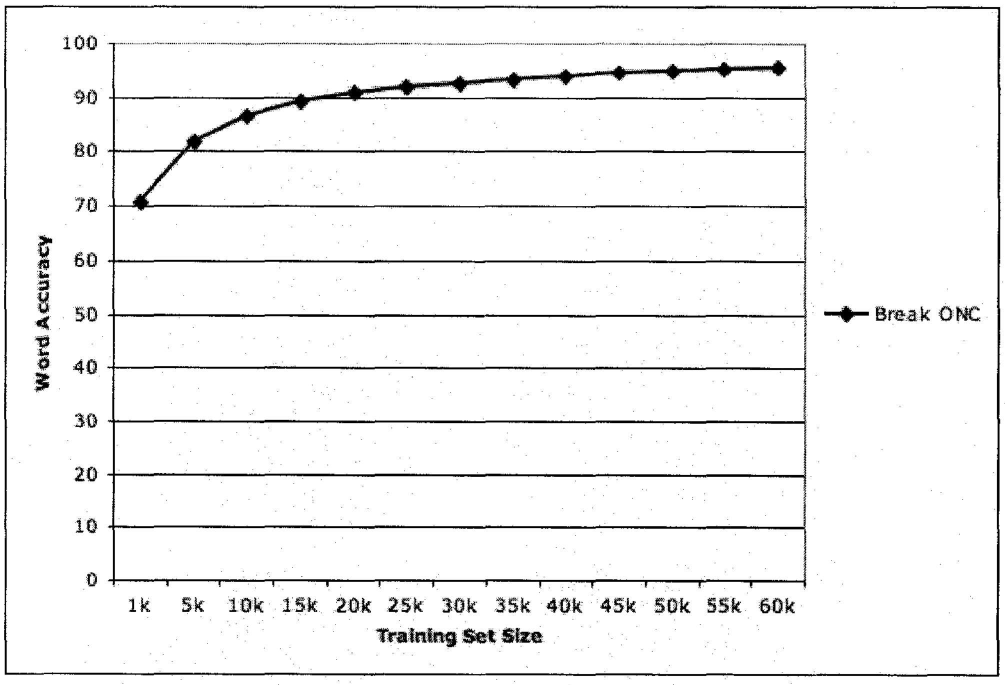


Figure 5.5: Word accuracy as a function of N-grams incorporated into the feature set.

System	Training Data	Word Accuracy	Syllable Break Accuracy
Break ONC	30K words	92.74	98.27
Numbered NB	30K words	92.30	98.22
Break ONC	~30K lemmas	91.35	97.91
Numbered NB	~30K lemmas	91.21	97.93

Table 5.6: Performance on a dataset drawn from lemmas is a more realistic approximation to how a syllabification system will fare on unseen words.

of speech derived from the same root, such as *educator* and *educational*. Nonetheless, duplication across the training and test sets will be greatly reduced. I therefore report Structured SVMs performance trained and tested on the CELEX lemma list. This dataset contains 35K words. I hold-out 10 percent words for testing, and train using the remaining words. Once again, duplicates, multiple-word entries, and words containing apostrophes are removed. Table 5.6 shows results on the lemma set and contrasts it with results taken from CELEX word lists. Performance falls off by about a percentage point, but word accuracy rates remain high.

5.4.3 Other Languages

Next, I apply my system to the other languages available in CELEX, Dutch and German. I apply the same feature set and the Numbered NB tag set to these new languages. For each language, I report results on a held-out 25K test set. I use both a small (50K) and large (250K) training set to learn the models. Table 5.7 shows the results of each.

System	Training Data	Word Accuracy	Syllable Break Accuracy
German	50K words	98.81	99.81
German	250K words	99.78	99.97
Dutch	50K words	98.20	99.71
Dutch	250K words	99.45	99.92

Table 5.7: Performance of SVM-HMM on the German and Dutch portions of CELEX.

Performance on German and Dutch is clearly better than performance on English. This is because syllabification is a more regular process for German and Dutch.

I performed no direct comparisons on German and Dutch, but there is some previous work that uses the same data set. Bouma's [2002] finite state approach achieved 96.49 percent word accuracy when trained on 50K examples taken from Dutch CELEX. My system achieves 98.20 percent word accuracy. With a larger model, trained on about 250K training examples, Bouma achieves 98.17 percent word accuracy, as compared to 99.45 percent for SVM-HMM. My system produces better performance, but Bouma's system has the advantage of producing concrete linguistic rules.

In German, Demberg's [2006] HMM-based approach scores 97.87 percent word accuracy, using about 250K CELEX training points. My word accuracy on that amount of training data is 99.78 percent.

It is difficult to draw definitive conclusions from these numbers, because the different systems do not use the exact same training and test data. However, my discriminative models out-perform the competitors on similar datasets, even though I do not tune my system to apply them to the new languages.

System	Training Data	Word Accuracy	Syllable Break Accuracy
German	45K lemmas	98.21	99.70
Dutch	107K lemmas	97.53	99.62

Table 5.8: Performance on CELEX lemmas is somewhat lower.

As with English, very large training sets create problems with overlap. I apply SVM-HMM to the lemma datasets for German and Dutch. There is quite a disparity in the size of these sets: the Dutch lemma set is 120K, while the German lemmas set is only 50K. In both cases, I randomly select 10 percent as a held-out test set, using the remaining lemmas for training. Results from these models are listed in Table 5.8. We can see that, for Dutch in particular, the overlap between training and test sets is inflating the word accuracy scores somewhat. However, on the whole, accuracy in these languages is still extremely high.

5.4.4 Letter-to-Phoneme Problem

One of the motivations for this work is to improve performance on the Letter-to-Phoneme (L2P) problem. Incorporating my syllabification models into an existing state-of-the-art L2P system [Jiampojamarn *et al.*, 2007] produces a noticeable improvement in L2P accuracy.

The L2P system in question explores using many-to-many letter-to-phone alignments as part of a prediction system in place of the traditional, strict one-to-one alignments. The L2P system first 'chunks' an input letter stream, predicting whether a letter generates a phoneme by itself, or whether it combines with an adjacent letter to produce a single sound. Given these chunks, a local classifier predicts the most likely phones for each chunk. A Markov model is then used to select the most likely sequence of phonemes, based on the candidate phones proposed by the local classifier.

To incorporate my syllabification models into this L2P system, the initial chunking and final Markov phases proceed as usual. However, the context window used by the local classifier is expanded to include syllable boundary information for the letters preceding and following the focus chunk. Recall from Chapter 2 that in gold standard syllabifications, short, stressed vowels tend to be followed by a coda rather than a syllable boundary. Because of this fact, phone prediction accuracy (especially for vowels) is likely to improve with the presence of syllable information.

To test the improvement syllabification can generate, the L2P system was run in three different ways:

(1) in its normal operation, with no syllabification model,

(2) using the gold standard syllabification found in CELEX, and

(3) using the SVM-HMM learned model to syllabify the words.

The gold standard L2P results can be viewed as an upper bound on the contributions of my syllabification models. All L2P results are reported in terms of word accuracy.

The L2P system was tested using ten-fold cross-validation, training on 90 percent of available data and testing on 10 percent for each fold. I used the Break ONC and Numbered NB English models, trained on 15K CELEX words, and the Numbered NB German and Dutch Models, trained on 50K words, as my syllabification models. I opted to use syllabification models trained on a smaller amount of data because both the syllabification and L2P models are trained on the same CELEX dataset. Using most of CELEX to train the syllabification model would be almost the same as using the gold standard, and not indicative of how the process would perform on real unseen data. The mean word accuracy over all ten folds is reported in Table 5.9.

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

Table 5.9: Adding gold standard syllabification information improves the performance of a state-of-the-art L2P system; using syllabification derived from a learned model captures some of that potential gain.

In English, perfect syllabification increases L2P word accuracy 1.62 percentage points. My Numbered NB syllabification model increases L2P accuracy by .88 points, while the Break ONC model increases accuracy by 0.92 percent. This represents over half of the potential gain — a particularly notable result because, to my knowledge, it is the first instance of a learned orthographic syllabification model actually improving performance on the L2P task in English. In Dutch, the potential gain is 1.47 percentage points, using the gold standard. My model increases the L2P score by 1.04 points, capturing over 70 percent of the available increase. In German, perfect syllabification produces only a small gain of 0.39 percentage points. Experiments show that my learned model actually produces higher word accuracy than the dictionary syllabification. This anomaly may be due to errors or inconsistencies in the dictionary syllabifications that are not replicated in the model output. My German results are somewhat at odds with Demberg's [2006] L2P results, which generated statistically significant L2P improvements of one full percent by adding her syllabification models. However, Demberg's base L2P accuracy is considerably lower, with word accuracy scores below 75 percent.

On the whole, I find my syllabification models never worsen L2P performance, and can produce noticeable improvements.

5.5 Conclusions

Most previous work on syllabification of letters has used a simple binary NB tag scheme, either implicitly or explicitly. In this chapter, I introduced a new tag scheme, Numbered NB, that produced superior results. I have also applied ONC tag schemes to the task; although this type of tagging is common in the phoneme domain, it is novel for letters. Unlike Numbered NB tags, ONC tags implicitly require every syllable to have a vowel. In spite of this extra constraint, the performance of ONC tag schemes lagged behind Numbered NB tags. However, a hybrid of NB and ONC tags performed very well.

This chapter has also shown the importance of complex, overlapping features. Expanding the context window used to predict each tag is an important component of good performance. However, adding N-gram features over the context window is at least as valuable in terms of increasing word accuracy. These features have the great advantage of being entirely language independent.

As components of a discriminative training method, the feature and tag sets introduced in this chapter produce a state-of-the-art syllabification system. In a direct head-to-head comparison in English, my system reduced the error rate by one-third, relative to the previous best system. Although direct comparisons in other languages were not possible, my system produced results that are competitive with the best published accuracies in German and Dutch. In tandem with a state-of-the-art L2P system, my English and Dutch syllabification models were able to produce notable increases in L2P accuracy.

Chapter 6

Syllabification of Phonemes

Linguists view syllabification of phonemes as entirely divorced from "syllabification" of letters. Nevertheless, from a computational perspective, the task is virtually identical: divide strings of symbols into segments based on underlying (possibly noisy) rules. Thus, the discriminative models outlined in Chapter 5 can be easily adapted to the new domain.

Indeed, *a priori*, I expected my system to perform rather better with phonemes. There is considerable ambiguity in the orthographic domain, which complicates the letter task. For instance, there are 22 vowel sounds in the English language, and several different letters combine in a myriad of ways to represent these sounds. Worse, the same letter combinations do not consistently represent the same vowel sounds. Consider the words `[br1d]` (*braid*) and `[{l|_I|br1|Ik]` (*al—ge—bra—ic*) — in the former, the letter combination *ai* maps to the single phoneme `[1]`, while in the latter it maps to two phonemes, `[1]` and `[I]`. Conversely, each vowel sound is uniquely represented by a single phoneme. By definition, an English syllable has only a vowel sound (peak), so we can deterministically find the nucleus in the phoneme domain, where we cannot in the letter domain. In addition to making the SVM's job easier, this greater transparency in the phoneme domain makes rule-based methods a viable option.

In Section 6.1, I explore several rule-based approaches to syllabification, implementing common theories of syllabification as outlined in Section 2.2. I compare the results of those rule-based methods with my discriminative approach in Section 6.3. In Section 6.2, I explain how the SVM-HMM outlined in Chapter 5 is adapted to the phoneme domain.

6.1 Rule-based syllabification

Rule-based methods are an appealing approach to syllabification in general because syllabification is a regular process that linguists have attempted to formalize as rules. Moreover, rules are more efficient than data-driven methods, because they require neither training data (which can be difficult to find) nor training time. All the methods presented in this section apply rules to classify phonemes as being either onsets, nuclei, or codas.

I did not pursue rule-based methods in the orthographic case because letters are so ambiguous, and previous work has found that rule-based methods in English perform extraordinarily poorly [Marchand *et al.*, to appear]. Furthermore, syllabification of letters is not an accepted phonological process as it is for phonemes, and dictionaries vary widely in how they syllabify orthographic forms. Thus, accuracy of orthographic syllabification is very much only with respect to the gold standard set out by a particular dictionary, which can be influenced by a number of different factors, and not necessarily any underlying linguistic principle. Of course, accuracy in phonemic syllabification is still relative to a given standard, but in a larger sense, we can view the efficacy of rule-based methods for phonemic syllabification as a measure of how well a dictionary ascribes to a linguistic principle of syllabification.

My baseline rule-based method is **Max Onset**. This approach simply maximizes the onset for every syllable, without any regard for the legality of the onsets involved. Only word-final consonants will be labelled as codas by Max Onset. Obviously, there are a number of consonant combinations that are illegal as English onsets, so word accuracy performance for Max Onset ought to be rather poor. However, the method should provide a good measure of how strictly a language follows

Vowels	9
Syllabic Consonants	9
Glides	8
[r]	7
[l]	6
Nasals	5
[s]	4
Voiced fricatives	3
Voiceless fricatives	2
Voiced plosives	1
Voiceless plosives	0.5

Figure 6.1: The Sonority Scale employed by Base Sonority

the maximal onset principle. Presumably, in a language like Hawaiian, where codas are prohibited, and any consonants are, by default, part of an onset, Max Onset would be sufficient to perform perfectly accurate syllabification.

The sonority sequencing generalization (SSG) is implemented as **Sonority**. This method uses an approximation of Selkirk's [1984] original sonority scale. However, Selkirk's original sonority scale is incomplete, leaving out a number of sounds altogether. In Sonority, these missing sounds have been added by grouping them with their most closely related sounds actually appearing in the scale. Thus, [N], not appearing in Selkirk's original scale, joins [m] and [n] to form the Nasals group. Sonority also does not register any difference between types of vowels, assigning them all the same sonority value. This is a departure from Selkirk's original hypothesis. The actual sonority scale used by Sonority is presented in Figure 6.1.

Recall that SSG alone is not sufficient to syllabify a word: a combination like [rpl] (two liquids separated by a plosive) can be divided as [r|pl] or [rp|l] and still follow the generalization. The Base Sonority implementation chooses the largest onset such that each phone in the onset has a sonority value higher than the preceding phone. Accordingly, given a word like [p3r|plEks] (*perplex*), the program will output *p3r—plEks*, maximizing the onset of the second syllable, rather than the coda of the first.

Note that Sonority can be viewed as an instantiation of Hammond's optimality theory. Sonority is a higher ranking constraint than maximal onset, so maximal onset can be violated in order to satisfy SSG. However, whenever possible, both implementations will produce valid sonority sequences and maximal onsets.

The **Legality** approach implements the legality principle. Legality can be considered a data-driven method of sorts: given a list of words, we designate all word-initial consonant clusters to be valid onsets, and all word-final consonant clusters to be valid codas. However, Legality does not require a separate labelled training set, so it is more akin to other rule-based methods. As with the sonority approach, I combine the legality principle with maximal onset, so the largest legal onset is always preferred to alternatives. As presented in Section 2.2, the legality principle considers both legal onsets and legal codas. However, the notion of a 'legal' coda is decidedly more ephemeral than that of a legal onset — coda rules are often broken. And, there may be cases where a three-phone consonant cluster cannot be divided into a valid two-phone coda or two-phone onset, based on the words seen. Consequently, my Legality implementation does not take into account legal codas, and seeks only to maximize the legal onset, relegating all remaining consonants to the preceding coda, without regard for their legality.

6.2 Adapting SVM-HMM to the Phoneme Domain

There are very few modifications required to apply SVM-HMM to syllabification of phonemes.

As with letters, the primary data set is CELEX. CELEX, unlike some other dictionaries, does not use the same syllabifications in the phoneme domain as the the letter domain. For example, CELEX syllabifies the word `[@b1sIz]` (*abases*) as *a—bas—es* in the letter domain, and `[@|b1|sIz]` in the phoneme domain. Note in particular how the syllabification of phonemes follows the maximal onset principle rule more strictly in the final syllable. By contrast, if the end of line break for *abases* were to come before the *s*, that might imply a different kind of *a* sound, so the onset is not maximized.

I also report results on the NETtalk data set. Most of NETtalk's truly bizarre orthographic syllabifications are an artifact of mapping phonemic syllabifications into the letter domain. Consequently, NETtalk is not nearly as problematic for syllabification of phonemes.

I experiment with most of the same tagging schemes for both phonemes and letters. It is not necessary to explore ONCE tags in the pronunciation domain, because there are no silent phonemes. For the letter case, generating ONC tags is not straightforward. Some knowledge of the language is required to deal with things like syllabic consonants and silent letters. By contrast, with phonemes, generating ONC tags from syllabified data is deterministic. All vowels are labelled as nuclei; all consonants preceding the nucleus of a syllable are tagged as onsets and all consonants following the nucleus have coda tags. Consequently, ONC tags are a much more natural choice for syllabification of phonemes, and can be applied much more easily across languages.

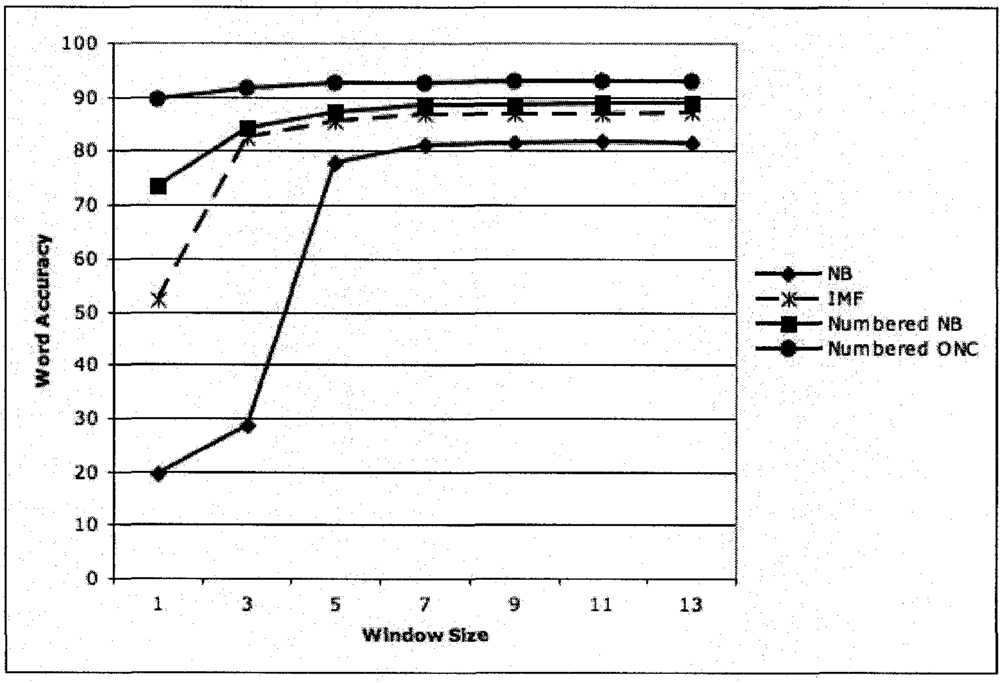


Figure 6.2: A window size of 9 is optimal for the phoneme domain.

I replicate the development set experiments from Chapter 5 to determine the optimal tag set and feature set. I use the phonetic equivalents of the 5K training set and 10K development set I employed in the letter domain.

Figure 6.2 shows how window size affects word accuracy on several tag sets. Unlike the letter domain, word accuracy really stops improving once the window size extends beyond nine characters. Hence, for the phoneme system, I will use a symmetric window of only four characters on either side of the focus letter.

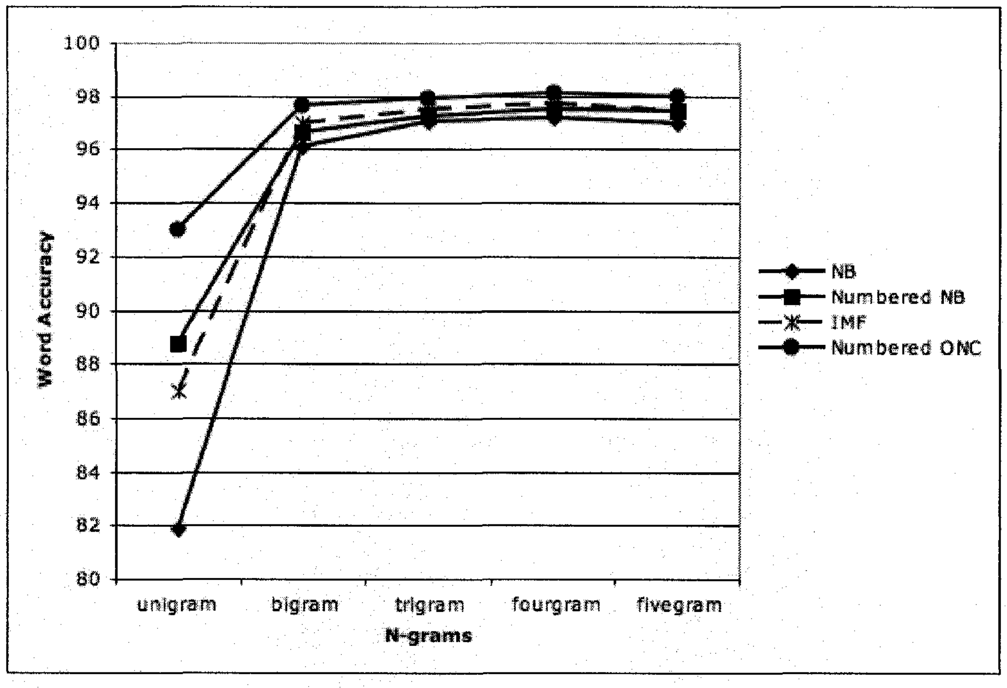


Figure 6.3: Fivegrams cause overfitting in the phoneme domain. Note that the y-axis has been truncated to improve readability.

Figure 6.3 demonstrates how higher order N-gram features affect the results. Here again, we see a deviation from the letter results: fivegrams cause word accuracy to decline, due to overfitting. Therefore, the feature set for the pronunciation domain will be limited to all the unigrams, bigrams, trigrams, and fourgrams within a nine-character window.

Selecting the best tag set is also more straightforward when working with phonemes. In the first place, Numbered NB tags are not any simpler than ONC tags in the new domain. We can deterministically generate ONC tags and easily apply them to other languages. Numbered NB tags still have the weakness of not requiring a nucleus in every syllable. In the development set output from Numbered NB tags, there are 18 words that contain impossible syllables consisting of only a consonant. With Numbered ONC tags, there is only one. And, there is negligible scope for improvement

Tag Set	Word Accuracy	Syllable Break Accuracy
NB Tags	97.16	99.32
IMF Tags	97.70	99.38
Numbered NB Tags	97.49	99.36
ONC Tags	97.86	99.35
Numbered ONC Tags	98.15	99.44
Reverse ONC Tags	98.03	99.40
Break ONC Tags	98.01	99.45

Table 6.1: Development set results for the phoneme domain.

by combining tag sets. Generally, if using a particular tag set produces the correct syllabification, all of the tag sets will be correct. Even with an oracle to tell us which tag set to choose, we would only be able to improve word accuracy from 98.15 to 98.40. Given all these factors, I report the results for only the highest-scoring tag set, Numbered ONC, throughout the rest of this chapter.

6.3 Experiments and Results

For the most part, in this section I will duplicate all the experiments reported in Section 5.4. Unfortunately, I do not have access to SbA results on identical data, so direct comparisons are unavailable.

6.3.1 English CELEX Data

My first set of experiments compares the performance of the three rule-based approaches and SVM-HMM against the CELEX gold standard. SVM-HMM is trained using 30K randomly selected words not appearing in the test set. This same training set is used to define the legal onsets for the Legality script. The test set is made up of 5K words not appearing in the training set. The training and test sets contain the same words as the datasets used for the learning curve comparison experiments in Section 5.4.2. As before, I report performance in terms of word accuracy and syllable break accuracy. Table 6.2 presents the results of each method.

As expected, Max Onset does not perform especially well. However, it still

Method	Word Accuracy	Syllable Break Accuracy
Max Onset	61.38	87.40
Legality	93.16	98.04
Sonority	77.72	93.15
SVM-HMM	98.86	99.69

Table 6.2: Syllabification performance in terms of word and syllable accuracy percentage.

syllabifies over 60 percent of words correctly, and achieves reasonably high tag accuracy. Thus, it appears that CELEX is fairly consistent with the maximal onset principle. Sonority is also not a very strong performer, although this too in not unexpected. Selkirk [1984] never claimed that the values in her sonority scale were definitive — only that the general ordering of sounds was correct. More tuning of the values in the sonority scale may well improve the performance of a sonority-based method; this is an avenue for future research.

The word accuracy score for Legality is about five percentage points lower than the discriminative method. However, Legality runs more quickly, and does not require labelled training data. Although out-performed by more sophisticated systems, the rule-based method is a viable alternative for languages with a paucity of labelled training data.

Notwithstanding the success of Legality, SVM-HMM is still far and away the best performer. In fact, the discriminative model's word accuracy exceeds the best rule-based approach when it is trained using only 5000 data points, as is evident in Figure 6.4. What is more, SVM-HMM does increasingly better as the amount of training data grows; the rule-based methods are unaffected by the amount of data available.

Of course, as with letters, there is considerable duplication within the CELEX dataset, as various forms of a word will appear multiple times. However, I find that the results on words are very much in line with comparable scores trained and tested on on English lemmas. Using a 90/10 train/test split, SVM-HMM scores 98.36 percent word accuracy on the lemma list, a decrease of only half a percentage point compared to a model trained on a comparable number of words.

I cannot make any direct comparisons between my work and previous work in

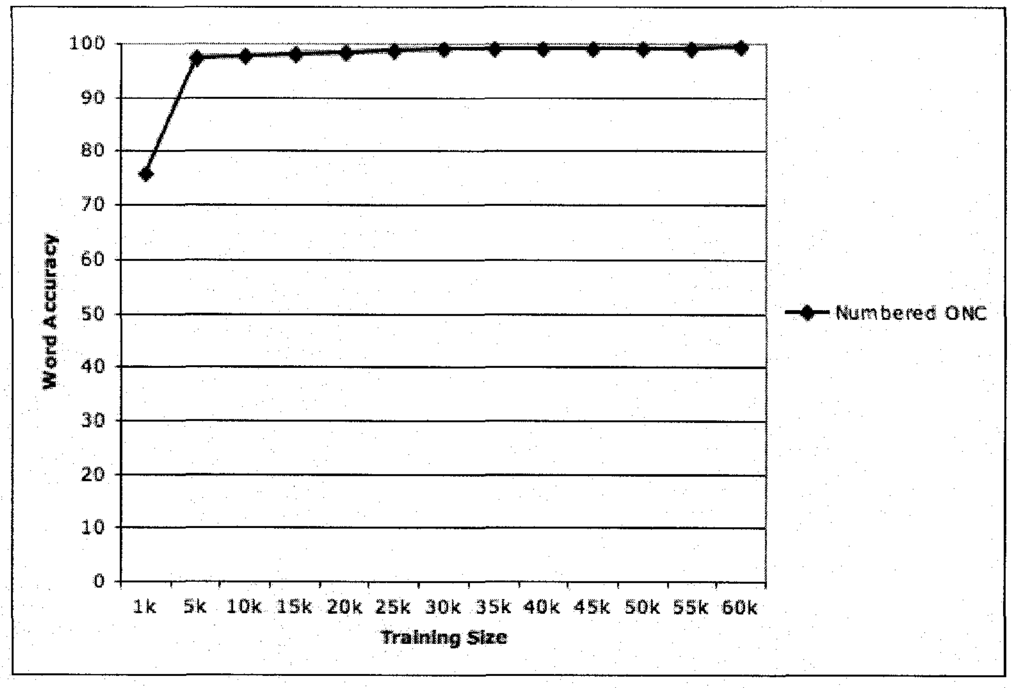


Figure 6.4: SVM-HMM requires relatively little data to achieve near-perfect word accuracy.

the phoneme domain. However, Müller's [2006] PCFG method achieved only 92.64 percent word accuracy on the CELEX dataset, which lags not only SVM-HMM, but the Legality method as well. Van den Bosch's [1997] Learned EBG approach is much closer in performance to my discriminative method, scoring 97.78 percent word accuracy on CELEX phonemes. However, van den Bosch uses almost twice as much data to train his system, and SVM-HMM still has an error rate that is nearly 50 percent lower. This is strong circumstantial evidence that SVM-HMM is the superior system.

6.3.2 NETtalk Data

My second set of experiments repeats the previous tests on the NETtalk dataset. The rule-based scripts developed on the CELEX set were applied directly to NETtalk without any tuning. The results reported here are on a 7K held-out test set. Both Structured SVM and Max Legality used a 13K training set, comprised of words not appearing in the test data. Unfortunately, even for phonemes, NETtalk has many errors. For instance, [*D8r|@|b6|t] (*thereabout*) is listed as the NETtalk gold standard syllabification. This is clearly incorrect, as the plosive consonant [t] can never form a syllable by itself. These errors are problematic for my discriminative method because SVM-HMM will learn that syllables without nuclei are acceptable. The existence of these errors further underscores my belief that the NETtalk dataset is unsuitable for the syllabification task.

Method	Word Accuracy	Syllable Break Accuracy
Max Onset	33.64	72.44
Max Legality	53.08	81.96
Sonority	48.00	79.48
SVM-HMM	92.99	97.79

Table 6.3: Results on the NETtalk dataset.

Table 6.3 presents my results on NETtalk. The main thing that stands out is the very low scores for the rule-based methods. Word accuracies have fallen off by more than forty percent. In particular, the score for Max Onset indicates that NETtalk follows the maximal onset principle only sporadically. For instance, the gold standard for [f5t|5] (*photo*) does not have a maximal onset in the second syllable, but the gold standard for [f5—tQn] (*photon*) does. Such irregularities imply that even a NETtalk-specific rule-based system, designed specifically to capture NETtalk's peculiar syllabifications, would not fare very well. Certainly, such a system would require far more hand-tuning than is desirable.

SVM-HMM, as a data-driven approach, is better equipped to deal with somewhat inconsistent syllabification. However, the erratic syllabifications still cause problems, as the word accuracy scores are well below what SVM-HMM can score on the CELEX dataset — on a comparable number of training points, CELEX word accuracy is still around 98 percent. Nonetheless, SVM-HMM does outperform SbA. Marchand, Damper, and Adsett find that SbA scores only 88.53 percent word accuracy on NETtalk phonemes [Marchand *et al.*, to appear]. My discriminative approach clearly outperforms their best score.

6.3.3 Other Languages

Next, I apply Structured SVM to other languages. To serve as a baseline, I also apply Max Onset and Legality to other languages, as there is no technical reason preventing it. However, the rule-based scripts are not expected to perform very well, as the maximal onset principle holds much more strongly for English than most other languages. A linguist familiar with any of these languages would likely be able to develop rule-based methods that far surpass any of my English-centric attempts.

For both German and Dutch, I use CELEX data, selecting a 25K test set, and two different training sets, one 50K and the other 250K. Particularly with the 250K training set, there is still the issue of duplication between the training and test sets, so I also perform experiments on the phonetic lemma lists, using a 90/10 training test split. For the most part, training and test sets contain the same words as the equivalent experiments presented in Chapter 5[1].

Method	Training Data	Word Accuracy	Syllable Break Accuracy
Max Onset	N/A	19.51	74.58
Max Legality	N/A	79.55	95.38
SVM-HMM	50K words	99.26	99.85
SVM-HMM	250K words	99.87	99.97
SVM-HMM	45K lemmas	98.98	99.76

Table 6.4: Results for German.

Results for German are listed in Table 6.4. The results of the rule-based scripts demonstrate that the Maximal Onset Principle is quite weak in German. SVM-HMM performs extraordinarily well on German, reflecting the highly regular syllable structure of that language. This is underscored but the word accuracy on the lemma data, which is still near 99 percent, even without benefit of overlap between training and test sets.

Although developed on English, my system is competitive with recent implementations for German phonemes. As with English, direct comparisons are dif-

[1]In CELEX's Dutch phoneme list, several thousand proper nouns of foreign extraction, like *Liverpool* and *Venezuela*, have no syllabification information. These words are simply omitted.

ficult. However, in general, the previous systems use far more training data and achieve much lower accuracy. The most word accuracy scores for the most recent work in German are very similar to my own. Schmid, Möbius, and Weidenkaff's fifth-order HMM achieves 99.85 percent word accuracy using about 278K training points [Schmid *et al.*, to appear 2007], but they hand-tuned their smoothing algorithm to syllabification of German phonemes. Using a standard smoothing algorithm and fourth-order HMM, Demberg scores 98.47 percent word accuracy. Earlier results on German phonemes include Müller's work with PCFGs [Müller, 2001a; Müller, 2006], and Krenn's HMM-based implementation. Müller's earlier work requires a hand-crafted grammar; to evaluate her method, she uses a training corpus of two million tokens, more than 90 percent of her test data appears in the training corpus, and her data covers only a small fraction of the words appearing in CELEX. Even so, her best word accuracy is only 96.88 percent on CELEX-derived syllabifications [Müller, 2002]. Müller's 2006 work is evaluated using 10-fold cross-validation on the full CELEX dictionary. This later work does not require a hand-crafted grammar, but only scores 90.45 percent word accuracy [Müller, 2006]. Krenn [1997] holds out approximately 20,000 words for testing, using the rest of CELEX for training purposes. Krenn does not evaluate at the word level, but her tag accuracy is only 98.34 percent. When trained on 250K words, my system achieves tag accuracy of 99.98 percent.

Results for Dutch are listed in Table 6.5. Performance in Dutch is rather weaker than in either German or English. This is noteworthy because SVM-HMM's orthographic syllabification is much better for Dutch than for English. Indeed, the structured SVM actually performs slightly better on Dutch letters than on the equivalent Dutch phonemes. Unfortunately, I know of no previous quantitative results for Dutch, so not even indirect comparisons with previous work are possible.

Method	Training Data	Word Accuracy	Syllable Break Accuracy
Max Onset	N/A	23.44	74.58
Max Legality	N/A	64.31	95.38
SVM-HMM	50K words	97.79	99.53
SVM-HMM	250K words	99.16	99.82
SVM-HMM	105K lemmas	98.14	99.59

Table 6.5: Results for Dutch.

6.4 Conclusions

In this chapter, I applied three different rule systems to syllabification of phonemes. Simply implementing maximal onset is insufficient, and the untuned sonority method presented here produces only mediocre results. However, combining maximal onset with the legality principle allows for word accuracy scores that are only five percentage points lower than my discriminative method. This is much higher than Marchand, Damper, and Adsett's previously-reported results for rule-based phoneme implementations on the NETtalk dataset. I would argue, therefore, that performance of rule-based scripts is as much a function of the dataset used as the gold standard as of the rules themselves.

Notwithstanding the success of the Legality method, the main contribution of this chapter is my discriminative approach. I applied SVM-HMM to syllabification of phonemes, with great success. The process was very similar to that for letters, except that the optimal feature set and tag set changed in the new domain. In English, my system outperforms earlier implementations by Müller and van den Bosch (on CELEX), and Marchand, Damper, and Adsett (on NETtalk). Unlike the rule-based approaches, my discriminative approach is not language-specific. Experiments show the performance of SVM-HMM is competitive with the best systems designed for German phonemes. Overall, my SVM-based system is a language independent implementation that produces state-of-the-art results in both English and German.

Chapter 7

Conclusions

7.1 Summary

This thesis presents an efficient, machine-learning-based method for automatic syllabification. Taking a discriminative approach to the problem, my system outperforms all existing systems on both English letters and phonemes. Experiments in this thesis also show that my system can achieve word accuracies in excess of 99 percent for German and Dutch phonemes and letters. Treating syllabification as a tagging task, my system labels each character in a word so that the sequence of labels implies the word's syllable breaks. When the problem is so formulated, a structured support vector machine turns out to be an excellent formalism for predicting syllable boundaries. It allows for an extensive feature set, and can be applied to any language for which there is available training data, in either the letter or phoneme domain.

On the whole, I am convinced that automatic syllabification is best approached using discriminative training techniques. Experiments show that SVM-HMM performs better on the syllabification task than predecessors like HMMs. It is more language- and domain-independent than PCFG-based approaches, and produces higher performance. SVM-HMM also out-performs data-driven techniques like

Syllabification by Analogy and Exemplar-Based Generalization, two lazy learning approaches that maintain a list of all available training data. This finding demonstrates that abstraction of training data does not invariantly degrade performance on the syllabification task, contrary to the hypothesis put forward by Marchand and Damper [Marchand and Damper, 2005; Marchand *et al.*, to appear].

In applying structured SVMs to automatic syllabification, I found that the labelling scheme used to indicate syllable boundaries crucially affects performance. In the letter domain, the usual approach has been to assign a binary label to each letter, indicating whether the letter is at a syllable boundary or not. Experiments in Chapter 5 show that this approach is decidedly weaker than alternatives. I considered several alternative tagging schemes, including a positional model that numbers each letter within a syllable, and an onset-nucleus-coda model that attempts to map each letter to its function within the syllable. Surprisingly, experiments indicated that the positional tag scheme performs better than the onset-nucleus-coda option. Several factors likely influence this finding. First, the positional model represents the syllable boundary with a single tag, rather than relying on the combination of two tags. This allows the structured SVM to explicitly minimize the error on syllable breaks, which is the desired metric. Second, the unpredictable relationship between English orthography and sounds introduces some noise in the onset-nucleus-coda gold standard, which probably affects performance somewhat. This is all the more likely because in the less-noisy phoneme domain, onset-nucleus-coda tags outperform the positional tags.

Incorporating my orthographic syllabification models into a state-of-the-art letter-to-phoneme conversion system increased L2P accuracy for both English and Dutch. To my knowledge, this is the first time a learned model has been successfully employed to improve L2P performance in these languages. As such, this represents an important contribution of this work.

In the phoneme domain, I presented several rule-based techniques. My implementation of the legality principle achieved word accuracies above 90 percent. This makes the Legality method an important alternative for syllabifying strings of phonemes, especially when labelled training data is unavailable.

7.2 Limitations and Future Work

One of the primary limitations of this work is the lack of direct comparisons. Apart from Syllabification by Analogy on English letters, I was unable to measure my system against existing systems operating on the same training and test data, because the training/test splits were unavailable. Nonetheless, the results in this thesis show that my system is certainly competitive with previous results reported in the literature. Arguably, the experiments show that my discriminative approach is superior. However, a systematic comparison of systems trained and tested on the same data is required before SVM-HMM can be definitively declared the best approach.

There are several possible avenues for future work. In the phoneme domain, tuning of the sonority scale used in the my sonority implementation may produce higher word accuracies. Some scope for improvement in the letter domain might be found by combining several tag sets in some sort of ensemble method. It would also be interesting to apply SVM-HMM to other languages, as appropriate data becomes available.

Most future work on letters likely lies in incorporating syllabification models into aspects of text-to-speech synthesis. In this thesis, automatic syllabification has been incorporated in to an L2P system as a pre-processing step. However, it might be fruitful to combine syllabification and letter-to-phoneme conversion, rather than treating them as serial processes. It would also be interesting to incorporate SVM-HMM syllabification into a prosody module of a TTS system, and compare the realism of the resulting speech.

Bibliography

[Altun *et al.*, 2003] Yasemin Altun, Ioannis Tsochantaridis, and Thomas Hofmann. Hidden Markov support vector machines. *Proceedings of the 20th International Conference on Machine Learning (ICML)*, pages 3–10, 2003.

[Baayen *et al.*, 1995] R. Baayen, R. Piepenbrock, and L. Gulikers. The CELEX lexical database (CD-ROM), 1995.

[Blevins, 1995] Juliette Blevins. The syllable in phonological theory. In John Goldsmith, editor, *The handbook of phonological theory*, pages 206–244. Blackwell, 1995.

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

[Collins, 2002] Michael Collins. Discriminative training methods for hidden Markov models: Theory and experiments with perceptron algorithms. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1–8, 2002.

[Crystal, 2003] David Crystal. *A dictionary of linguistics and phonetics*. Blackwell, 2003.

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

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

[Damper, 2001] Robert Damper. Learning about speech from data: Beyond NETtalk. In *Data-Driven Techniques in Speech Synthesis*, pages 1–25. Kluwer Academic Publishers, 2001.

[Deligne *et al.*, 2001] Sabine Deligne, François Yvon, and Frédéric Bimbot. Selection of multiphone synthesis units and grapheme-to-phoneme transcription using variable-length modeling of strings. In *Data-Driven Techniques in Speech Synthesis*, pages 125–146. Kluwer Academic Publishers, 2001.

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

[Fisher, 1996] William Fisher. Tsylb syllabification package. ftp://jaguar.ncsl.nist.gov/pub/tsylb2-1.1.tar.Z, 1996. last accessed 25 April 2007.

[Goldwater and Johnson, 2005] Sharon Goldwater and Mark Johnson. Representational bias in unsupervised learning of syllable structure. *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL)*, pages 112–119, 2005.

[Goslin and Frauenfelder, 2001] Jeremy Goslin and Ulrich Frauenfelder. A comparison of theoretical and human syllabification. *Language and Speech*, 44:409–436, 2001.

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

[Guenter, 29 November 2006] Joshua Guenter. Editor of pronunciation, Merriam-Webster Inc. Personal correspondence, 29 November 2006.

[Hammond, 1997a] Michael Hammond. Optimality theory and prosody. In *Optimality Theory: An Overview*, pages 33–58. Blackwell, 1997.

[Hammond, 1997b] Michael Hammond. Parsing in OT. Rutgers Optimality Archive, http://roa.rutgers.edu/index.php3, 1997. last accessed 25 April 2007.

[Hastie *et al.*, 2001] Trevor Hastie, Rober Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning.* Springer, 2001.

[Herbst and Joachims, 2006] Evan Herbst and Thorsten Joachims. SVM-HMM: Sequence tagging with support vector machines and its application to part-of-speech tagging. http://svmlight.joachims.org/svm_struct.html, 2006. last accessed 07 May 2007.

[Jiampojamarn *et al.*, 2007] Sittichai Jiampojamarn, Grzegorz Kondrak, and Tarek Sherif. 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, 2007.

[Kahn, 1976] Daniel Kahn. *Syllable-based generalizations in English Phonology.* PhD thesis, Indiana University, 1976.

[Kessler and Treiman, 1997] Brett Kessler and Rebecca Treiman. Syllable structure and the distribution of phonemes in English syllables. *Journal of Memory and Language*, 37:295–311, 1997.

[Kiraz and Möbius, 1998] George Kiraz and Bernd Möbius. Multilingual syllabification using weighted finite-state transducers. *Proceedings of the 3rd Workshop on Speech Synthesis*, pages 71–76, 1998.

[Kohler, 1966] K. Kohler. Is the syllable a phonological universal? *Journal of Linguistics*, 2:207–208, 1966.

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

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

[MacKinney-Romero and Goddard, 2006] René MacKinney-Romero and John Goddard. Syllabification using decision trees, early results on three languages. *3er Taller de Tecnologías del Languaje Humano*, pages 282–287, 2006.

[Marchand and Damper, 2005] Yannick Marchand and Robert Damper. Can syllabification improve pronunciation by analogy of English? *Natural Language Engineering*, 1:1–25, 2005.

[Marchand *et al.*, to appear] Yannick Marchand, Connie Adsett, and Robert Damper. Automatic syllabification in English: A comparison of different algorithms. *Language and Speech*, to appear.

[McCallum *et al.*, 2000] Andrew McCallum, Dayne Freitag, and Fernando Pereira. Maximum entropy Markov models for information extraction and segmentation. *Proceedings of the 17th International Conference on Machine Learning (ICML)*, pages 591–598, 2000.

[McDonald *et al.*, 2005] Ryan McDonald, Koby Crammer, and Fernando Pereira. Flexible text segmentation with structured multilabel classification. *Proceedings of the conference on Human Language Technology (HLT) and Empirical Methods in Natural Language Processing (EMNLP)*, pages 987–994, 2005.

[Müller *et al.*, 2000] Karin Müller, Bernd Möbius, and Detlef Prescher. Inducing probabilistic syllable classes using multivariate clustering. In *Proceedings of the 38th Meeting of the Association for Computational Linguistics (ACL)*, pages 225–232, 2000.

[Müller, 2001a] Karin Müller. 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, 2001.

[Müller, 2001b] Karin Müller. Probabilistic context-free grammars for syllabification and grapheme-to-phoneme conversion. *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 143–150, 2001.

[Müller, 2002] Karin Müller. Probabilistic context-free grammars for phonology. *Proceedings of the 6th Workshop of the ACL Special Interest Group in Computational Phonology (SIGPHON)*, pages 80–90, 2002.

[Müller, 2006] Karin Müller. Improving syllabification models with phonotactic knowledge. *Proceedings of the 8th Meeting of the ACL Special Interest Group on Computational Phonology (SIGPHON)*, pages 11–20, 2006.

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

[Schmid *et al.*, to appear 2007] Helmut Schmid, Bernd Möbius, and Julia Weidenkaff. Tagging syllable boundaries with joint N-gram models. *Proceedings of Interspeech*, to appear 2007.

[Sejnowski and Rosenberg, 1988] Terrence Sejnowski and Charles Rosenberg. NETtalk: a parallel network that learns to read aloud. In *Neurocomputing: Foundations of Research*, pages 661–672. MIT Press, 1988.

[Selkirk, 1984] Elisabeth Selkirk. On the major class features and syllable theory. In *Language Sound Structure*, pages 107–136. MIT Press, 1984.

[Spencer, 1996] Andrew Spencer. *Phonology: theory and description*. Blackwell, 1996.

[Sproat, 1998] Richard Sproat. *Multilingual Text-to-Speech Synthesis: The Bell Labs Approach*. Kluwer Academic Publishers, 1998.

[Taskar *et al.*, 2003] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. *Proceedings of the Neural Information Processing Systems Conference NIPS*, 2003.

[Treiman and Zukowski, 1990] Rebecca Treiman and Andrea Zukowski. Toward an understanding of English syllabification. *Journal of Memory and Language*, 29:66–85, 1990.

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

[van den Bosch *et al.*, 1998] Antal van den Bosch, Ton Weijters, and Walter Daelemans. Modularity in inductively-learned word pronunciation systems. *Proceedings of New Methods in Language Processing and Computational Natural Language Learning (NeMLaP/CoNLL)*, pages 185–194, 1998.

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

[Zhang and Hamilton, 1997] Jian Zhang and Howard Hamilton. Learning English syllabification for words. *Proceedings of the 10th International Symposium on Foundations of Intelligent Systems*, pages 177–186, 1997.

Appendix A

DISC Phoneme Encodings

Throughout this thesis, I use the DISC encoding to represent phonemes. The DISC encoding is convenient for computer applications because each phoneme is represented by a single ASCII character. The following tables map common DISC encodings for English phonemes to equivalent International Phonetic Alphabet (IPA) symbols. The provided orthographic examples assume received pronunciation, as is standard in the CELEX dataset.

DISC Encoding	IPA Symbol	Examples
`@`	/ə/	**a**chieve, pavem**e**nt
`#`	/aː/	m**ar**ket, d**a**nce
`u`	/uː/	l**u**minous, g**oo**se
`i`	/iː/	w**ee**k, m**ea**l
`$`	/ɔ/	f**or**t, **au**dio
`3`	/ɜː/	v**er**se, w**or**k
`I`	/ɪ/	f**i**ll, ever**y**
`E`	/ɛ/	ah**ea**d, l**e**ss
`{`	/æ/	m**a**n. g**a**s
`Q`	/ɒ/	s**o**ft, w**a**sh
`V`	/ʌ/	r**ou**gh, s**u**mmer
`U`	/ʊ/	br**oo**k, f**u**ll
`1`	/eɪ/	cl**ay**, b**a**ke
`2`	/aɪ/	g**uy**, **i**con
`4`	/ɔɪ/	t**oy**s, f**oi**l
`5`	/əʊ/	b**ow**l, s**o**
`6`	/aʊ/	cr**ow**n, **ou**t
`7`	/ɪə/	d**ee**r, cl**ea**r
`8`	/ɛə/	r**a**re, f**ai**r
`9`	/ʊə/	b**oo**r, man**u**re

Table A.1: English Vowels and Diphthongs

DISC Encoding	IPA Symbol	Example
p	/p/	**p**ot
b	/b/	**b**ake
t	/t/	**t**ag
d	/d/	**d**im
J	/ʧ/	**ch**ina
_	/ʤ/	**j**uice
k	/k/	**k**in
g	/g/	**g**et
f	/f/	**f**ire
v	/v/	**v**iolin
T	/θ/	**th**ank
D	/ð/	**th**ere
s	/s/	**s**ee
z	/z/	**z**ip
S	/ʃ/	**sh**ow
Z	/ʒ/	**g**enre
h	/h/	**h**ope
m	/m/	**m**int
n	/n/	**n**ap
N	/ŋ/	ri**ng**
r	/r/	**r**oad
w	/w/	**w**ax
j	/j/	**y**ellow

Table A.2: English Consonants

Appendix B

Implementation Details

For most of the experiments reported in this thesis, I used the Linux binary distribution of SVM-HMM available from http://svmlight.joachims.org.

To train an SVM-HMM for the automatic syllabification task, I use the following command line:

```
./svm_hmm_learn_hideo -e 0.5 -c 0.1 train.data model.dat
```

The `-e` option defines epsilon in equation 3.6.

The `-c` option defines the amount of acceptable error on the training data. This is a tunable parameter that can crucially affect performance. As an illustration, consider the data in Table B.1. These results are attained by training a letters model with a reduced feature set that includes only unigram features falling with a 5-letter window size. The Numbered NB tag set is used for output tags. There are two main things to notice. First, the word accuracy score can improve by nearly twenty percentage points simply by using the correct c-value. Second, the best c-value does not remain constant as the feature set changes. With a full complement of five-grams across an 11-character window, the optimal c-value is 0.1; with only unigrams within a three-character window, 10 is the best. During development, I systematically varied the c-value for each new feature set.

C Value	Word Accuracy
0.001	40.80
0.01	47.84
0.1	55.13
1	58.21
10	59.06

Table B.1: Results on the held-out letters development set, using numbered NB tags and a reduced feature set.

The `train.data` argument is the input training file. In Figure B.1, I present the input file for the features associated with the word *pro—ton*. For simplicity, I show only unigram features within a three-character window; Numbered NB tags are again used as the output tagging scheme.

```
N1 qid:3.1 17:1 28:1 73:1 82:1 124:1#p - proton
N2 qid:3.2 19:1 44:1 70:1 82:1 129:1#r - proton
B qid:3.3 16:1 46:1 75:1 98:1 124:1#o - proton
N1 qid:3.4 21:1 43:1 70:1 100:1 123:1#t - proton
N2 qid:3.5 16:1 48:1 69:1 97:1 109:1#o - proton
N3 qid:3.6 15:1 43:1 55:1 102:1 109:1#n - proton
```

Figure B.1: Sample feature set input

The first item in each line is the correct output tag. The string `qid:3.i` indicates the ith letter in the third training word. Everything following the # character is a comment. All the intervening numbers represent features.

The best way to represent letters (or phonemes) as features is to use a string of binary bits to represent each character. Thus, in Figure B.1, the notation `17:1` indicates that the 17th bit is turned on, in this case representing *p*, the 16th letter in the alphabet (the first bit in the string is reserved for representing the special end-of-word character). A unigram feature is therefore represented by a string of 27 binary bits, while a bigram feature is represented by a string of $27^2 = 729$ bits, and so on for higher-order N-grams. This is the limit case; as discussed in Chapter 5, we can greatly reduce the number of features by discarding bits for N-grams not actually observed in the training data. SVM-HMM assigns a value of zero to any feature not explicitly listed in the input file.

Obviously, this is not the most compact representation, but empirically it is the most effective. Possible alternatives, such as using ASCII codes to represent each letter, are less sucessful. This is because there is no meaningful linguistic sense in which the letter *b* is closer to the letter *a* than, say, the letter *p*. Indeed, in terms of the phonemes they produce, *b* and *p* are very similar; however, their associated ASCII codes imply that they are much further apart than *a* and *b*. Using a sequence of binary bits captures the qualitative difference between letters.

The `model.dat` argument is the output file where the learned model will be written. An associated file called `model_svmModel.dat` will also be generated.

To classify new test examples, use the following command line:

```
./svm_hmm_classify test.data model.dat classify.tags
```

The `test.data` argument is a feature file describing the test words, using the same format as Figure B.1. When using a held-out test set (*i.e.* when the true tags are available), SVM-HMM will report the accuracy on the test set. If the correct tags are not available, a dummy tag must be substituted.

The `model.dat` argument is the model file trained for the given feature set.

The `classify.tags` argument is the file where the output tags will be written. To convert these tags into syllable boundaries, I concatenate input words and output tag sequences and then apply regular expressions.