Improving Pronunciation Accuracy of Proper Names with Language Origin Classes

by

Ariadna Font Llitjós

B.A., Universitat Pompeu Fabra, 1996

M.S., Carnegie Mellon University, 2001

Master Thesis

August 2001

Committee members:

Alan W Black

Kevin Lenzo

Roni Rosenfeld

Acknowledgements

I would like to thank my advisor Alan Black for all his support and dedication, without him this thesis would not have been possible; Kenji Sagae for the insightful discussions about this thesis and, most importantly, for his patience and support; Guy Lebanon and Christian Monson, LTI colleagues, for the discussion about unsupervised clustering; and Toni Badia for having introduced me to the field of Natural Language Processing and for his support during all these years.

This work was supported by a “La Caixa” Fellowship.

Table of Contents

Abbreviations v
Abstract vi
Models at a glance vii

1. Introduction 1
   1.1. Speech Synthesis 3
   1.2. Grapheme to phoneme conversion 6
   1.3. Pronunciation of proper names 8
   1.4. Knowledge of ethnic origin of proper names 10
   1.5. Organization of this thesis 13
2. Letter-to-sound rules 15
3. Literature Review 18
   3.1. Pronunciation of proper names 18
      3.1.1. [Spiegel 1985] 18
      3.1.2. [Church 2000] 20
      3.1.3. [Vitale 1991] 21
      3.1.4. [Golding and Rosenbloom 1996] 23
      3.1.5. [Deshmukh et al. 1997] 24
   3.2. Grapheme to phoneme conversion 25
      3.2.1. [Black et al. 1998] 26
      3.2.2. [Tomokiyo 2000] 28
      3.2.3. [Torkkola 1993] 30
4. Proper name baseline 31
   4.1. Data 31
   4.2. Technique: CART 32
   4.3. Letter-phone alignment 34
   4.4. Testing and results 36
5. First Approach: Adding language probability features 39
   5.1. Letter Language Models and Language Identifier 39
      5.1.1. Multilingual Data Collection 40
      5.1.2. 25 LLMs and the Language Identifier 42
   5.2. Model 1: Language specific LTS rules 45
      5.2.1. Results for model 1 46
   5.3. Model 2: Building a single CART 49
      5.3.1. Results for Model 2 53
6. Second Approach: Adding language family features 55

6.1. Model 3: Family specific LTS rules 56
6.1.1. Results for model 3 56
6.2. Model 4: Indirect use of the language probability features 58
6.3. Model 5: A 2-pass algorithm 59
6.4. Results for models 4 and 5 61
7. Third approach: Unsupervised LLMs 64
7.1. Unsupervised clustering 65
7.2. Basic data structure 67
7.3. Similarity measure 70
7.4. No clashes 74
7.5. Computational time and efficiency 77
7.6. Models 6 and 7 78
7.7. Results for models 6 and 7 80
8. Upper Bound and error analysis 83
8.1. Noise user studies 83
8.2. Language Identifier accuracy 84
8.3. Oracle experiment 85
8.4. Error analysis 88
9. Conclusions and discussion 94
10. Future work 96
10.1. Unsupervised Clustering 96
10.2. Probabilistic Model 97
10.3. Another model for the second, language family, approach 98
10.4. Behind the numbers 98
10.5. Use of (dynamic) priors 100
10.6. Proper names web site 101
10.7. Handcrafted LTS rules 101
11. References 102

Appendix A: CMUDICT Phoneme Set 108
Appendix B: Training data sample 110
Appendix C: List of proper names allowables 112

Abbreviations

ANAPRON	Analogical pronunciation system
CART	Classification and regression tree
CMU	Carnegie Mellon University
CMUDICT	The CMU pronunciation dictionary (http://www.speech.cs.cmu.edu/cgi-bin/cmudict)
DT	Decision Trees
EM	Expectation - Maximization (algorithm)
G2P	Grapheme-to-phoneme (conversion)
IT	Information Theory
LI	Language Identifier
LLM	Letter-language model (in this thesis, trigram-based)
LTS	Letter-to-sound (rules)
ME	Maximum Entropy
MI	Mutual Information
ML	Machine Learning
NAMSA	AT&T LTS rules system, pronounced “name say”, originally developed by Kenneth Church in 1985
OOV	Out of vocabulary (words)
TTS	Text-to-speech (system)
SS	Speech Synthesis
SR	Speech Recognition

Abstract

Pronunciation of proper names that have different and varied language sources is an extremely hard task, even for humans. This thesis presents an attempt to improve automatic pronunciation of proper names by modeling the way humans do it, and tries to eliminate synthesis errors that humans would never make. It does so by taking into account the different language and language family sources and by adding such information as features into the pronunciation models, either directly or indirectly.

This approach does result in an improvement of pronunciation accuracy, however in order to assess the true goodness of this approach, we would need to develop a more accurate language identifier. Ultimately, the data we would like to have in order to train our models is a list of proper names tagged both with their phonetic transcription and with the language they come from.

A new approach this thesis begins to investigate is the unsupervised clustering of proper names to derive *language* classes in a data-driven way. With this approach, no language classes (Catalan, English, French, German, etc.) need to be determined a priori, but rather they are inferred from the names and their pronunciation. The clustering method used takes into account letter trigrams as well as their aligned pronunciation at training time. Experiments using the classes derived from unsupervised clustering are still preliminary and have not yet yielded an improvement in pronunciation accuracy of proper names.

Models at a glance

This thesis describes several models to pronounce proper names. All the models implemented use the CART technique (see 4.2); but different models use different features to build it (shown in the *CART features* column in table 1), and they use these features in mainly two different ways (*method*):

- to split the data and build specific CART for each subcorpus (direct use of the features), and
- to add to a single CART together with the n-gram features built from all the data (indirect use of features)

Table 1 can be used as a roadmap through the thesis, to help the reader remember what the different models are, what features they include and how are they built.

Models	CART features					Method	
	n-grams	25language LLM	5-family LLM	25 + 5 LLM	Unsupervised LLM	Direct use of features	Indirect use of features
Baseline (chapter Chapter 4)	X						X
Model 1 (section 5.2)	X	X				X	
Model 2 (section 5.3)	X	X					X
Model 3 (section 6.1)	X		X			X	
Model 4 (section 6.2)	X		X				X
Model 5 (section 6.3)	X			X			X
Model 6 (section 7.6)	X				X		X
Model 7 (section 7.6)	X				X		X

Table1: the models described in this thesis at a glance

Chapter 1
Introduction

Speech processing has two main research areas: speech recognition and speech synthesis. Over the last decade, speech processing has become ubiquitous, fueled by the increasing demand for automated systems with spoken language interfaces. Speech processing has progressively penetrated our daily lives; a good example of this is the fact that almost every customer service in the United States has some kind of automated system with a speech recognition component and, less often but increasingly more so, a speech synthesis component (usually delegated to dynamically generated prompts).

Speech recognition is the process of converting speech to text. This involves correctly segmenting and tagging the acoustic signal into phonemes, and then into words and sentences[1]. Speech synthesis is the reverse process, namely the conversion from text to speech; this process will be described in detail in the next section. Currently, speech processing still faces many challenges, being able to correctly capture and generate natural speech is an extremely hard task. One of the reasons why speech synthesis is not as widespread as speech recognition is that it usually sounds quite unnatural and hearers therefore usually prefer to listen to recordings of a human voice.

In this thesis, we address the problem of generating an accurate pronunciation[2] in the important domain of proper names. We use well-established statistical methods, as well as a new unsupervised approach, to automatically improve pronunciation accuracy.

Having accurate pronunciations for proper names is of particular interest for speech recognition, speech synthesis, as well as for all applications that have a speech recognition or a speech synthesis component, such as dialog systems, speech-to-speech

[1] State-of-the-art Speech Recognition Systems achieve this by employing acoustic modeling, language modeling, adaptation, etc.

[2] Through this thesis when we talk about pronunciation, we mean machine generated pronunciation, and not pronunciation as generated by humans.

machine translation, (reverse) directory assistance, automated customer service and any state-of-the-art application that deals with natural language processing.

This thesis proposes and evaluates a procedure in the context of speech synthesis; however, this by no means limits its contributions to speech synthesis. As Spiegel envisions already in his 1985 report, the ultimate goal is the merging of synthesis and recognition technologies in such a way that when a person indicates a synthesizer has mispronounced his or her name, the device asks "Excuse me, how do you pronounce your name?". Then, the person's response is analyzed by a speaker-independent, phonetically-based speech recognition system.

Since we know the letter sequence the person is uttering, we can use that letter sequence to constrain the phonetic extraction. Additionally, this kind of interaction mimics what humans do when confronted with an uncommonly pronounced name.

The goal of this thesis is to find an automatic and efficient way to improve the pronunciation model for proper names.

We aim to approximate the pronunciation to the one a knowledgeable native American English speaker would use in normal conversation, which should not be confused with the correct native pronunciation in that particular foreign language (since that could be equally puzzling to the native American English hearer, if not more). For example, if we consider the name *Van Gogh*, what we want our system to output is not / *f ae1 n g o k* / or / *f aa1 n g o g* /, which some people may claim is the correct way of pronouncing it, but rather the American pronunciation of it: / *v ae1 n g ow1* /.

1.1. Speech Synthesis

As mentioned above, speech synthesis is the process of converting a stream of text into a speech signal. This process can be broken down into the following 4 steps (cf. figure 1):

1. **Text Analysis**: break text (string of characters) into words, also known as tokenization
2. **Linguistic Analysis I**: transcribe a word into a string of phonemes, also known as grapheme-to-phoneme conversion (G2P) + lexical stress
3. **Linguistic Analysis II**: add accent, duration and other prosodic features to the pronunciation
4. **Waveform analysis**: convert the pronunciation into a waveform

This thesis focuses on step 2 of the speech synthesis process, i.e. on how words are transcribed into phonemes. The next section describes this step in more detail.

$$\begin{tikzcd}
\text{Text} \arrow[d, Rightarrow, "\textit{tokenization}"] \\
\textbf{Words (graphemes)} \arrow[d, Rightarrow, "\textbf{\textit{G2P}}"] \\
\textbf{Phonemes} \arrow[d, Rightarrow, "\textit{Prosody modeling}"] \\
\text{Pronunciation} \arrow[d, Rightarrow, "\textit{Waveform generation}"] \\
\text{Speech}
\end{tikzcd}$$

Figure 1: Text-to-speech 4-step process

We illustrate the end-to-end process in the following example:

Input: "*This thesis focuses on step 2, i.e. on how words are transcribed into phonemes*"

Step 1[3]: *This*
thesis
focuses
on
step
2
i.e.[4]
…

Step 2:	**this** $\longrightarrow$	***/ dh ih1 s /*** **or**
		/ dh ih0 s /
	thesis $\longrightarrow$	***/ th iy1 s ah0 s /***
	focuses $\longrightarrow$	***/ f ow1 k ah0 s ih0 s /*** **or**
		/ f ow1 k ih0 s ih0 z /
	on $\longrightarrow$	***/ aa1 n /*** **or**
		/ ao1 n /
	step $\longrightarrow$	***/ s t eh1 p /***
	two $\longrightarrow$	***/ t uw1 /***
	i.e. $\longrightarrow$	***/ ai1 iy1 /***
	…	

see Appendix A: CMUDICT Phoneme Set

Note that G2P mapping is not always one-to-one. Not only are there many phonemes a letter can be converted into, but also, sometimes, there is more than one correct pronunciation for any given word depending on the context. Sometimes it is just a matter

[3] Quite easy for English, Catalan, and Portuguese, among many others, but very hard in languages like Chinese.
[4] Assuming we have an abbreviation data base

of stress, which might depend on the pragmatics of the sentence, namely what the speaker is trying to emphasize; at other times it depends on speaker idiosyncrasies (idiolect). Only to cover the first 7 words, we would need to have a lexicon for numbers, as well as an abbreviation pronunciation database. A good example of tokens with multiple pronunciations depending on context is numbers; we want our synthesizer to distinguish between '\$1999' and the year '1999', say, since most fluent speakers of English will say 'nineteen hundred (and) ninety-nine (dollars)[5]' and 'nineteen ninety-nine' respectively.

> **Step 3**: Add prosodic features to the string of phonemes such as phrasing (chunking in speech), intonation (accents and F0 generation), duration (length of segments) and power (loudness).
>
> **Step 4**: Generate waveforms; synthesize the string of phonemes (the result from step 2), taking into account the prosodic feature specification from step 3 (see below).
>
> **Output**: Speech (the waveform)

Waveform generation typically involves concatenating speech from an existing (prerecorded) database in order to synthesize a given pronunciation[6]. There are different units that can be selected for the concatenation process: phones, diphones, sub-word units or cluster based units.

Diphone selection is the safe option but it does not sound natural. It requires one example of each phone transition in the language (it assumes that two-phone context is sufficient). **Unit selection**, on the other hand, is the exciting option, and it is famed for its high quality as well as for its low quality, but it is difficult to build. It requires many examples of the units in different contexts; as well as the selection of the appropriate

[5] Note that synthesizing this string correctly involves changing the order of the tokens '\$' and '1999'.

[6] When concatenating speech, one needs to be careful about the joins. For a great example of badly joined speech synthesis, go to http://www.cs.cmu.edu/~awb/talks/rinss.html, to the "Waveform generation" section, and click on the example "How not to do word level concatenation".

units phonetically and prosodically. Both techniques require units to be well labeled and recorded.

1.2. Grapheme to phoneme conversion

The conversion of words into phonemes can be done in one of three ways:

- by looking the words up in a **dictionary** (with possibly some limited morphological analysis),
- by applying morphological and rhyming **analogy strategies**, or
- by applying **letter-to-sound (LTS)** rules.

All three methods have advantages and disadvantages. **Dictionaries** are the most reliable source for pronunciation[7]. Nevertheless, even with the larger pronunciation dictionary available, there will still be a substantial number of unknown words. The number of out-of-vocabulary (OOV) words depends on the domain and genre of communication. As will be described in section 1.3, most proper nouns are rare. When a word is not in the dictionary, we need to use other strategies to *guess* the pronunciation of that word.

Analogy strategies work very well when there are similar (or identical) words in the dictionary; however such strategies are computationally expensive (especially when having to look through a very large dictionary for possible matches), and when a match is not found, it fails to generate a pronunciation. Either if a match is found or not, analogy strategies need to be combined with LTS rules, since most matches are partial, and we need LTS rules to generate the pronunciation for the part of word that is not contained in the match.

[7] Experiments have shown that the use of the dictionary cuts the error rate approximately by half [Vitale 1991] and [Yvon et al. 1998].

LTS rules are a *generalization* of how letters are mapped into sounds (Chapter 2 describes this in more detail). LTS rules tend not to work as well as dictionary lookups or analogy strategies, since they fail for irregular words. Nonetheless, any speech synthesis system that wants to be able to output a pronunciation for any given word requires LTS rules, which (everything else having failed) will generate the best guess, given the training data.

Most speech synthesizers adopt a hybrid technique, using large pronunciation dictionaries when possible, turning to morphology and rhyming analogies when a match is found, and falling back to LTS rules when everything else fails. LTS rules are thus the last resource in current text-to-speech (TTS) systems, and tend to be applied for rare, out-of-vocabulary (OOV) words, which are typically harder to pronounce (see next section).

In this thesis, we concentrate on this last resource, and try adding several types of relevant information to the baseline pronunciation model to improve word accuracy of proper names. If we wanted to make a fair comparison between the full capability of our TTS system and other systems, as done in [Golding and Rosenbloom 1996], we would first look up words in a large dictionary (the CMUDICT contains more than 56,000 names). If the name were not in the dictionary, we would then fall back to the proper name LTS rules. For the rest of this thesis, we concentrate on the harder task of improving LTS rules.

1.3. Pronunciation of proper names

What makes pronunciation of proper names difficult, or special, in comparison to words that are regular entries in the lexicon of a given language? Various reasons are given in the research literature ([Vitale 1991], [Church 1986 & 2000], [Coker et al. 1990], [Jannedy and Moebius 1997] and [Yvon et al. 1998]):

- Names can be of very diverse etymological origin and can surface in another language without undergoing the slow process of assimilation to the phonological system of the new language. It is not uncommon, to encounter the same name with 2 or 3 different degrees of assimilation and therefore 2 or 3 different pronunciations within a language (one of many examples of this is the surname *Dubois* in American English, which can be pronounced as / *d uw0 b oy1 s* / or as /*d uw0 b w aa1*/).

- The number of distinct names (types) tends to be very large. For English, a typical unabridged collegiate dictionary lists about 250,000 word types, whereas a list of surnames compiled from an address database (supplied by Donnelley Marketing) contains 1.5 million types (75 million tokens). Zipf's law [Zipf 1929] indicates that a relatively small number of high frequency words can cover a high percentage of words in any given text. However, this law is less favorable for names, since the tail of low frequency names is substantially flatter and larger, so it takes a much larger list of names to achieve a given level of coverage. For example, covering half of the surnames in the United States of America requires a list of more than 2,300 names, whereas for ordinary words, the same 50% level of coverage can be achieved with a dictionary of only 141 words [Coker et al. 1990].

- Name pronunciation is known to be idiosyncratic; there are many pronunciations contradicting common phonological patterns, as well as alternative pronunciations for certain grapheme strings.

- Names have a restricted morphological structure. They are not equally amenable to morphological processes, such as word formation and derivation or to morphological decomposition as regular words are, and the methods of derivation are more diverse and language specific. This further contributes to the coverage problem for names.

- Stress patterns for proper names, however, tend to be quite fixed in different languages, and thus relatively easy to determine [Church 1986 & 2000]. Finnish, Italian and Japanese names always have penultimate stress, whereas French names have ultimate stress, and English and German names have initial stress.

As mentioned in 1.2, current TTS systems tend to relay on large dictionaries, even for proper names. Generally speaking, nothing ensures correct pronunciation better than a direct hit in a pronunciation dictionary. However, for the reasons exposed above, this approach is not always feasible for proper names. In short, we are not dealing with a memory or storage problem, but with the need to provide approximately correct pronunciations for OOV names. In unrestricted text we cannot foresee all the proper names before hand, and therefore we need to offer a mechanism to pronounce them, even when they are not in our lexicon.

On the other hand, "excruciatingly correct" pronunciation is a very difficult goal to achieve, and the level of accuracy desired can vary quite widely depending on the application. It is likely that the required degree of accuracy will strongly depend on the closeness of association between the users and the people whose names are synthesized (I have a very close association to my name and will want it correctly synthesized anytime I hear it; whereas the sales representative checking my account will care much less [Spiegel 1985].

Furthermore, one could argue that, in real text, foreign proper names account for a small percentage of all the words, and so improvement in this area would have no significant impact on the overall accuracy of the system. However, most current speech applications such as reverse directory assistance, automated operator services, catalog ordering, customer records and navigation systems, to name just a few, crucially depend on accurate pronunciation of proper names. Moreover, even if the amount of foreign names were relatively small for some given application, getting them right would substantially improve the perceived quality of the system.

There are several types of proper names: street names, place names, company names and people names. In this thesis, we concentrate on the latter kind, people proper names, but most of the research done here will probably generalize to other kind of proper names.

1.4. Knowledge of ethnic origin of proper names

The intuition behind this thesis is that knowing the ethnic origin of proper names provides clues on how to pronounce them. This is clearly true for humans, and the question is whether this information can be used to improve automatic pronunciation accuracy.

A specific example of how language knowledge would help in determining the pronunciation of names can be found in the letter sequences 'cc' and 'sch'. If we know that the language is Italian (*Bocaccio*), we probably want to transcribe 'cc' as /ch/, whereas if language is English (*McCallum*), we want to transcribe 'cc' as /k/. Similarly, if we know that the language is Italian or English we might want to transcribe "sch" as /sk/ (*Schiavone* or *Schooley*), whereas if the language is German, the right transcription would be /sh/ (*Schoenenberg*).

Previous research on this area seems to indicate that ethnic origin of proper names does help improve pronunciation accuracy. In the 80s and early 90s, pronunciation of proper names was a major concern for most speech synthesizers. Researchers such as Spiegel, Church and Vitale magnificently described the difficulty of pronouncing proper names and their attempts to overcome this problem (see 3.1).

Our hypothesis *that knowledge of language origin improves pronunciation accuracy* is directly inherited from their work; however, one advantage of the work presented here is that of having more computational power than 10 years ago, which allows us to work with a much larger amount of data and to approach the LTS rules problem in a much more automatic way.

Another advantage is that evaluation methods have become much more formal and strict over the last decade. Work done in the late 80s and early 90s on speech synthesis systems was hardly ever formally evaluated. Evaluation expectations are now much higher, and any research work currently undertaken in this area (and most other areas) undergoes much greater scrutiny. This thesis is an attempt to provide an objective and exhaustive description of the experiments and evaluations made to improve pronunciation of proper names considering language origin information.

It is possible that, since many of the languages names originate from have a more standardized pronunciation than English (e.g. Polish, Italian, Japanese), knowing the origin of an unknown word may allow more specific rules to be applied. This hypothesis is supported by results reported in [Black et al. 1998], [Jannedy and Moebius 1997] and [Yvon 1994] for German and French, languages which have easier LTS rules than English. Spanish and Japanese are two further examples of languages with straightforward LTS rules.

In some cases, we even expect our system to outperform native English speakers, since what we aim for the *educated pronunciation* of foreign proper names in American English. One example is the pronunciation of Chinese names. Few native English speakers know how to pronounce them correctly, but there seem to be precise English rules for pronouncing such names. Our system would pronounce those names in the

Americanized, educated way, hopefully achieving higher pronunciation accuracy than average American speakers.

Even if American English native speakers do not always know how to pronounce foreign proper names correctly, there seems to be "an American way" of pronouncing foreign names (what Kenneth Church calls *pseudo-foreign accent*, see [Church 2000]), which people are able to recognize as the (mis)pronunciation of those foreign names. This Americanization of the names is what we are after, and we actually believe that, if our system had ethnicity class information available, we could do an even better job, and synthesize the educated pronunciation of these foreign names.

Previous experiments on large lists of words and pronunciations have shown that when a lexicon has more foreign words than another (CMUDICT vs. OALD in [Black et al. 1998]), this has quite an impact on speech synthesis accuracy. Such experiments report a difference of 16.76% on word accuracy[8] due to foreign words, a great proportion of which is likely to be proper names, since they are harder to predict without any higher level of information.

The reader should keep in mind that the US is very ethnically diverse, and has a very large number of people with foreign proper names (at least family names), some of which have kept their original pronunciations, and some of which have been completely assimilated, making it impossible to get their pronunciation always right. And by *right* we mean the exact same pronunciation it would be used by the person him/herself. It is a very common practice in the US to ask people how they pronounce their name. So we cannot expect a machine to miraculously guess the right pronunciation either, nor to achieve foreign pronunciation fluency, but rather to get as close as possible to what an educated native American English speaker would say.

[8] Letter and word accuracy are described in section 4.4.

1.5. Organization of this thesis

In Chapter 2, we introduce the concept of letter-to-sound rules. Chapter 3 is a literature review of previous work done in the areas of automatic pronunciation of proper names and G2P conversion. Chapter 4 describes the proper name data as well as the basic technique to build LTS rules, used to build all the models presented in this thesis. It then described the baseline model for pronunciation of proper names and establishes the lower bound in pronunciation accuracy against which the rest of the experiments in this thesis will be compared.

Chapter 5 describes the set of experiments that take into account language origin information. It first introduces the notion of letter-language models and describes the implementation of a Language Identifier (5.1.2). Then it presents two models: the first uses the language information directly and builds individual, language specific LTS rules (5.2); the second makes an indirect use of language information by building just one set of LTS rules (5.3).

Chapter 6 describes the set of experiments related with language family information. First, it presents the same experiments done in chapter 5 but with the family language models instead. Then it goes on to describe a 2-pass algorithm, which classifies names first into the 5 family languages, and then into the 25 letter language models (described in chapter 4) belonging to that family only.

Chapter 7 explores a way to generate *language* classes in a fully data-driven way, by using unsupervised clustering, as a novel approach to model pronunciation of words. It defines similarity of clusters (7.3) and introduces the notion of clash between clusters (7.4). Section 7.5 discusses some computational and efficiency issues of clustering, and section 7.6 describes a few preliminary experiments.

Chapter 8 talks about the upper bound in performance for this problem, and describes some error analysis experiments performed on the training data. Chapter 9 presents the conclusions of this thesis. Finally, Chapter 10, points out a few future directions.

Chapter 2
Letter-to-sound rules

The problem we are ultimately trying to solve is assigning a phoneme to a letter; we have $|\text{phoneme}|^{\text{word length}}$ choices for each word we want to know the pronunciation of. In this case, there are 101 phone combinations (see section 4.3), and thus the size of the hypothesis space becomes $101^{\text{word length}}$. If we do not introduce any further constraint, the random performance for a list of words of average length 5 would be about $1/101^5$.

It is widely known that the context around a letter has a strong impact on its pronunciation; for example, none of the 'e's in *eel* will be pronounced the same as the 'e' in *less*.

A simple statistical trick that assigns the most probable phone to each group of 3 or 4 letters, achieves a significant improvement of about 46% word accuracy on our proper names training data, which is already much better than random. (If we assigned each letter the most likely phone, we would get less than 0.5%)

Being able to ask questions about a three-letter window to the left and to the right of the letter under consideration, significantly improves pronunciation accuracy. This is what classification and regression trees (CART) do (details on how this is done are given in 4.2). The CART technique has been used successfully by a number of researchers to obtain general letter-to-sound (LTS) rules for English (British and American), French, German and Thai[9] ([Chotimongkol and Black 2000], [Black et al. 1998] and [Jiang et al. 1997] among others). This is also the technique used to build all the pronunciation models presented in this thesis.

[9] The difficulty and appropriateness of using LTS rules is very language dependent.

The process followed to build the baseline, as well as the LTS rules in [Black et al. 1998], [Chotimongkol and Black 2000][10], is not fully automatic, but the manual intervention required is small and may easily be done even by people with very little knowledge of the language being dealt with. It involves the following 7 steps:

- Pre-processing lexicon into suitable training set
- Defining the set of allowable mappings from letters to phonemes
- Constructing the probabilities of each letter/phoneme pair
- Aligning letters to an equal set of phonemes/epsilons[11]
- Extracting the data by letter suitable for training
- Building CART models for predicting phone from letters (and context)
- Building additional lexical stress assignment model (if necessary)

All except the first two stages of this process are fully automatic. The process as implemented in this thesis will be described in more detail in Chapter 4.

Traditionally, people have used handcrafted LTS rules, but this requires extensive linguistic knowledge and it is very time consuming and prone to error. It may take a person 2 years to manually build LTS rules for a language. Until not long ago, people that worked on this tedious task were proud of the pronunciation accuracy achieved by their manually build LTS rules. Nevertheless, automatically built LTS rules have recently been shown to achieve comparable or even higher accuracy than manual LTS rules. An example of this is reported in [Chotimongkol and Black 2000] for Thai, where the accuracy using traditional handcrafted models is 65.15%, whereas the accuracy achieved with the automatic approach went up to 68.76%.

[Black et al. 1998] technique achieves 57.80% word accuracy on the CMUDICT (see paper review in the next chapter for more details). This number is not on the exact same training data, but rather on a superset containing the list of proper names used in this

[10] The process of building LTS rules is described in detail in the Festivox on-line manual: http://festvox.org/

[11] A letter is transcribed as epsilon (ε) when it is not pronounced; the reader can think of epsilon as silence.

thesis, and it does not take lexical stress into account. If we take lexical stress into account on the same data, the whole CMUDICT, word accuracy drops to about 51%.

The accuracy of the LTS baseline model (using the same technique) for just proper names is slightly higher. This might at first strike us as surprising, since (foreign) proper names are believed to be harder to pronounce than common words (see reasons in section 1.3); however, this can probably be explained by the fact that we are already giving the information that they are proper names to the CART. The baseline achieves 54.08% word accuracy, which represents about an 8% relative improvement over the simplest method of assigning the most probable phone to each letter taking into account letter context (see chapter 4 for the description of the baseline).

Chapter 3
Literature Review

In this chapter, we review the most relevant research papers for this thesis. First, we comment on work done in the domain of proper name pronunciation [Spiegel 1985], [Church 2000], [Vitale 1991], [Golding and Rosenbloom 1996], [Deshmukh et al. 1997]. Then we describe related work in the more general domain of G2P conversion [Black et al. 1998], [Tomokiyo 2000] and [Torkkola 1993]. There are many more papers that describe different G2P approaches, which are referenced at the end of the thesis; however, they had no great impact on the development of this thesis so we do not review them in this chapter.

3.1. Pronunciation of proper names

3.1.1. [Spiegel 1985]

Pronouncing Surnames Automatically

In the September 1985 Bell Communications Research report, Murray F. Spiegel assesses the magnitude of the problem of pronouncing proper names automatically and evaluates four speech synthesizers of the time on the 2,000 most common American surnames (even though he notes that there is a huge variability regarding the most common surnames from region to region and even from city to city). He uses data from the Social Security Administration (1974) and from Smith's work *American Surnames* (1969).

In this report, Spiegel claims that an approach that would guarantee the correct automatic pronunciation of virtually everyone's name in the U.S. might involve a database of every person's name and its phonetic transcription, which could easily exceed 250G bytes and rules it out for not being feasible any time soon.

Currently, 250G bytes of storage are more than feasible, and AT&T has indeed shot for this approach, storing a huge list of proper names and their pronunciations. Even though AT&T's proper names synthesis system is one of the best, the problem is still far from being completely solved. One reason for that is that many names have several correct pronunciations, and this will depend on the region, the different language sources they have, and ultimately how each person that has that name pronounces it. Being able to discriminate between all the possible pronunciations of a name according to the context is still beyond the scope of any speech synthesis system, and we will not attempt to solve it in this thesis.

Spiegel's choice of error criterion was to accept any plausible pronunciation, based on interviews with 3-4 co-workers on ambiguous surnames (verbatim, page 113), and thus, as he admits, the results are a somewhat *liberal* acceptance of pronunciation. Errors were categorized into mild, worst (*embarrassingly wrong*) and due to poor synthesis.

When weighting the error rates for all names by their frequency-of-occurrence, Spiegel obtained the following percentages:

A 17%	B 24.8%	C 21.5%	D 33.3%

Less than 20% of the total errors were mild errors, and synthesis errors were not significant except for synthesizer C; so the majority of the errors were actually due to "embarrassingly wrong" pronunciations, errors that no human would ever make.

Spiegel states that the synthesis rules should make errors that are similar to human mistakes, since when errors do not follow this dictum, service users will have a difficult time confirming which name is actually being synthesized. An example of error that does not mimic human mistakes can be found in the name *Gutierrez*, which was synthesized as / *g i s h e e r z* / by the best synthesizer, A. Even though this dictum seems quite easy to accomplish, it is difficult to do for all possible surnames.

In his error analysis, Spiegel points out that rules contained in commercial synthesizers require considerable improvement in the assignment of proper stress, which is what probably lead Kenneth Church to do research on this subject (see next section).

3.1.2. [Church 2000]

Stress Assignment in Letter to Sound rules for Speech Synthesis

In the November 2000 AT&T Research technical memorandum, Kenneth Church describes his ACL'85 paper with the same title. He restricts his discussion of LTS rules for foreign words to stress; however, he shares the main insight of this thesis. He points out that, even if one might expect that a loan word would be stressed using either the rules of the language it was borrowed from or the rules of the language that it was borrowed into, in fact, it is often the case that neither ones nor the others account for the stress of loan words. Church gives Italian and Japanese surnames as examples of this.

Church also believes that speakers of English adopt what he calls a *pseudo-foreign accent*. That is, when speakers want to communicate that a word is non-native, they modify certain parameters of English stress rules in simple ways that produce bizarre "foreign sounding" outputs. Thus, if an English speaker wants to indicate that a word is Japanese, he might adopt a pseudo-Japanese accent that marks all syllables heavy regardless of their shape.

The pseudo-accent approach presupposes that there is a method for distinguishing native from non-native words, and for identifying the etymological distinctions required for selecting the appropriate pseudo-accent. To decide the relevant etymological distinctions he used a statistical trigram model. This is also the method used in this thesis as the basis for the Language Identifier (see 5.1.2).

Church concludes his memorandum pointing out that etymology is not just used for stress assignment and that other phonological rules seem to be conditioned by etymology.

Although Church's work was mostly focused on stress and etymology distinctions (Old English, Middle English, Old French, Middle French, etc.), instead of pronunciation in general and ethnicity origin classes (Catalan, English, Scottish, German, Dutch, Swedish, etc.), it is clear that his observations are valid and relevant in the context of this thesis. His work corroborates the need to account for the origin of proper names if we want to achieve more natural speech synthesis.

Bell Labs TTS initial system had 15 languages, and was trained on hundreds to thousands of examples (Church, personal communication). Unfortunately, Church did not publish any results, nor did he carry out any formal evaluation of his statistical model (this was not really expected in the late 80s early 90s). This emphasizes the importance of the work done in this thesis, since we report our experiments and evaluation following up on Spiegel's and Church's original idea, which can be used as a reference.

3.1.3. [Vitale 1991]

An Algorithm for High Accuracy Name Pronunciation by Parametric Speech Synthesizer

In this paper, Tony Vitale describes the name pronunciation software developed at DEC. In a nutshell, their system first performs a dictionary lookup, and for names not found, the system identifies the language of the name and applies LTS rules for that language.

Language identification is done in two steps. First, a set of filter rules is applied. The filter rules determine what language the name is from, or at least what languages is *not* from, based on the characteristic letter patterns in that name. If multiple candidate languages remain, the system chooses among them via trigram analysis.

As described in this paper, the data set contained 34,337 names and four test sets of 100 names each were used to evaluate the system. The dictionary covered all frequent names (functors) and 2,000 most common surnames.

Evaluation of the resulting pronunciations was done by linguists, which marked them as being reasonable or not (verbatim: "for the purposes of evaluation, a pronunciation was accepted if linguists *felt* that the segmental phonemic output and stress placement were *reasonable*" page 273).

Vitale's work is closely related with the work in this thesis; however there are 3 key differences:

- Vitale's system has a fairly complex and sophisticated language identification component, which requires extensive handcrafting, whereas the models presented here have much simpler and fully automatic language identification.
- Vitale's system has a set of language specific, handcrafted LTS rules, whereas we propose to have just one set of LTS rules which contain as much language information as the data-driven algorithm judges to be useful.
- Evaluation of the system is quite vague and subjective, and it does not render to fair comparison.

Even if Vitale's evaluation is not as rigorous as one might wish, his work is very valuable for this thesis, since his experiments show that knowing the etymology of a name improved pronunciation accuracy of the system.

3.1.4. [Golding and Rosenbloom 1996[12]]

A comparison of Anapron with seven other name–pronunciation systems

This paper presents an experiment comparing eight systems: a name-pronunciation system (Anapron), three early 90s state-of-the-art commercial systems (Bellcore, Bell Labs and DEC) two variants of a machine learning system (NETtalk), and two humans.

Anapron (Analogical pronunciation system) applies adapted MITalk rules to generate a first approximation to the pronunciation of the name, and it draws analogies from names in its case library (5,000 entries) to cover exceptions to the rules. This is very similar from what TTS (Bell Labs) does, although TTS applies cases before rules.

The other two commercial systems (The Orator System from Bellcore and DECtalk^{TM}), do rule-based reasoning plus dictionary lookup of names and name morphemes, and they map a case to other occurrences of the same case, rather than mapping a case more generally to any name that is deemed compellingly similar.

The two NETtalk-based systems (connectionist networks trained with backpropagation) work purely from cases, and do not use rules at all.

The experiment involved running all systems on the same 400-name test set (from the flat tail of the frequency of frequencies curve in the Donnelley corpus, i.e. rare names), and piping the resulting pronunciations through DECtalk^{TM} speech synthesize, in random order. A panel of 14 test subjects judged the acceptability of the pronunciations.

The results show that the humans performed better than any other system, but only achieved an overall accuracy of 93%[13] (thus were judged by their peers to generate

[12] Even if this was published in 1996, this work was already in progress during 1991 [Vitale 1991].

[13] For each name, judges had to score the pronunciation as (1) clearly acceptable, (2) somewhere in between and (3) clearly bad. The results reported represent the percentage obtained from adding up 1 and 2.

pronunciations "that no one would say" for 28 names in the test set). The three commercial systems, Orator, DECtalk^{TM} and Bell Labs TTS, were judged to perform between 93% and 98%. Anapron's overall performance reached 86%, and the NETalk-based systems performance fall between 78% and 67% (one of the main causes the authors point out for this is that they have a tendency to assign illegal stress patterns).

This paper is very valuable for this thesis, since the authors performed a very carefully set evaluation, and even if the commercial systems exact implementation details remain a black box, the results can be used as a reference. However, the reader should keep in mind these results were obtained by listening to the pronunciations and by relatively subjective human scoring (equivalent to what we report in our user studies), rather than by exact match of the transcription output of the systems against a pronunciation dictionary. The latter is a much more objective and precise way of evaluating pronunciation and is the one adopted to evaluate performance in this thesis.

3.1.5. [Deshmukh et al. 1997]

An advanced system to generate pronunciation of proper nouns

Deshmukh and his colleagues at the Institute for Signal and Information Processing at the ECE Department in Mississippi State University take a neural network (NN) solution to the learning of proper noun pronunciation. They used hand transcribed data for their experiments.

They trained a stochastic NN to model the mapping of text spelling n-tuples to phonemes. In particular, they used a multiple hidden layer Boltzmann machine (trained with backpropagation) and a multiplayer perceptron. They compare his results with the ones of a decision trees (see 4.2).

Preliminary evaluation on 128 names, each four letters long, was not very favorable to the NNs approach. The Boltzmann machine had 52.13% name error rate (which corresponds to 47.87% word accuracy), whereas the decision tree (DT) had a name error rate of 44.85% (55.15% word accuracy)

Evaluation on the full data set of 18,494 surnames and 24,040 pronunciations, where 15,000 names were for training and 3,494 for testing, the Boltzmann machine name accuracy fall way down to 29.56%, and the DT name accuracy fell to 53.85%. And these results do not even take stress into account.

The conclusions they reached are that DTs tend to perform better, but NNs tend to generalize better on unseen data; and ANNs are CPU expensive (days of training), but do not require large amount of memory, whereas DTs are quick to train but need large amount of storage.

3.2. Grapheme to phoneme conversion

This section describes a couple of **symbolic learning** approaches to G2P conversion. We concentrate on the **instance-based learning** techniques, for these are the ones used in this thesis. The reader should keep in mind, however, that there are other symbolic learning techniques such as memory or case-based learning [Daelemans and Bosch 1997] and [Busser et al. 1992].

Furthermore, approaches to G2P conversion are not limited to symbolic learning. Researchers have also applied Finite State Machines ([Bouma 2000] and [Knight and Graehl 1998], among others). Many neural networks (NN) have tried to apply ANN techniques to the task of converting graphemes into phonemes. Examples of this are the commonly cited [Sejnowski and Rosenberg 1987] who developed the first "network that learned how to pronounce English", [Fukada et al. 1998], [Hain 1999], [Dietterich, et al. 1995], as well as the work reviewed in the previous section by [Deshmukh et al. 1997], among others. The reason we do not review in more detail this branch of the literature is that ANN methods have not achieved state-of-the art performance, and the results neural network researchers report are usually quite low. The interested reader can find a comparison between state-of-the-art systems and NN systems is the [Golding and Rosenbloom 1996] reviewed in the previous section.

Finally, some commercial systems have adopted analogy-based methods to solve the problem of finding the pronunciation of rare words, such as proper names. An example of this is ANAPRON, a system that has been briefly described in the previous section when reviewing [Golding and Rosenbloom 1997]. Other work in this area has been done by [Coker et al. year] and [Yvon 1994].

3.2.1. [Black et al. 1998]

Issues in Building General Letter to Sound Rules

This paper points out the usefulness of having LTS rules for any real TTS system and explains the main steps involved in building LTS rules with CART trees in festival [Black 2000], which is the technique used to obtain the baseline for foreign proper names.

Black and his colleagues tried the LTS rules technique on four lexicons, the Oxford Advanced Learners Dictionary of Contemporary English (OALD) (British English), CMUDICT version 0.4 (US English), BRULEX (French), and the German Celex Lexicon, and obtained the following results:

Lexicons	Correct	
	Letters	Words
OALD	95.80%	74.56%
CMUDICT	91.99%	57.80%[14]
BRULEX	99.00%	93.03%
DE-CELEX	98.79%	89.38%

[14] This accuracy is achieved when ignoring stress.

That is the main reason we decided to build the baseline with the same version and the same technique that was used in this paper, so that a fair comparison could be established, and we would have a better idea of the usefulness of implementing our hypothesis. One could argue we should have used a newer version the CMUDICT (0.6), but that would defeat the purpose of building a system comparable with previous experiments. We feel that having this comparability justifies this choice of not using the last updated version of CMUDICT.

Their explanation for CMUDICT's significantly worse accuracy is that it contains many more foreign names, which are harder to pronounce. This view is supported by many other researchers (see section 1.2); however, the accuracy of the baseline obtained for the foreign proper names list from the same CMUDICT version is higher, which could be explained in terms of constraining the search to proper name specific LTS rules only, as opposed to general LTS rules.

Another relevant finding for this thesis is that some experiments showed that the best model for unknown words is not the best model for the test set lexicon. This leads us to believe that this should also be true for foreign proper names.

Finally, they report that including part-of-speech information in the phone prediction models themselves improved the accuracy of the model, and that stress cannot be predicted very well on local context alone in English. Following this idea, Model 1 includes sliding trigram probabilities as features of the CART trees, together with the three previous letters and the three next letters, to see whether this improves LTS rules accuracy above the baseline.

3.2.2. [Tomokiyo 2000]

Applying Maximum Entropy to English Grapheme-to-Phoneme Conversion

In his final project for the *Topics in Statistical NLP* course at CMU, Takashi Tomokiyo applied two different well known machine learning techniques to the problem of producing a phoneme sequence given the spelling of a word: decision trees (with the C4.5 algorithm) and a Maximum Entropy (ME) model. He used the already aligned CMUDICT and the OALD pronunciation dictionaries from [Black et al., 1998], and split the data into two different ways[15].

For the C4.5 experiment, the letter accuracy on the test set achieved for the CMUDICT was 87.5%, whereas the letter accuracy for the OALD lexicon was 93.5% (both with pruned trees and split-B). Note that this is lower than the letter accuracy reported in [Black et al. 1998] reviewed above, where CMUDICT letter accuracy was 91.99%, and OALD letter accuracy was 95.80%.

For the ME model, Tomokiyo manually selected the features, and used a 7-letter window of context. He found out that by adding bigram and trigram features, the accuracy of the ME model increased. For the CMUDICT, the highest letter accuracy achieved on the test data was 88%, and for the OALD lexicon, 93.5%. So, once he added the bigram and trigram features to the ME model, the accuracy was either the same or slightly better than with C4.5[16].

[15] **Split-A**: use every tenth word in the lexicon as test data, and the rest as training data (same as in [Black et al. 1998], [Chotimongkol and Black, 2000], and in this thesis). **Split-B**: apply stemming with Porter stemmer and pick the first 10 of every 100 stems and use all the words mapped to the stem as test data, the next 10 stems as the heldout data, and the other 80 as the training data.

[16] The reason for that might be that the ME model, unlike decision trees, does not split the training data, so parameters of less informative features are also trained using all the training data. Moreover, it does not impose an induction bias (no assumption is made about unseen events), whereas DTs have the soft bias of preferring shorter trees.

After an error analysis, Tomokiyo observes that the existence of foreign words is a major cause of errors, and so he proceeds to focus only on the foreign words problem. He tried two different ways of adding ethnic origin or language class information: character-based n-gram model and k-means clustering, and adding n-letter suffixes as additional features for the ME model; he tried both C4.5 and ME techniques.

For C4.5 and unpruned trees, the accuracy went up 0.1% with the clustering approach and 0.3% by adding suffixes as additional features, whereas for the ME model, adding 3-letter suffixes improved letter accuracy in 1.4% (up to 89.9%).

Finally, he argues that a significant percentage of the error was due to inconsistencies in the lexicons, especially in the usage of secondary stressed vowels, and so he suggests that cleaning the lexicon manually will be necessary for better performance. The data that he worked on is the same data we used for the experiments reported in this thesis.

This work directly relates to our Model 1 (see chapter 5); however, it does not take advantage of a very valuable piece of information we have at hand: the pronunciation of the words. In chapter 7, we describe an attempt to use both the spelling as well as the pronunciation of the word to perform unsupervised clustering and get classes that maximally differentiate the pronunciation. Rather than pure ethnicity groups, what we would like to have is classes that best account for the differences in the pronunciation of proper names (e.g. Japanese and Italian names, which have similar LTS rules, may end up in the same class).

3.2.3. [Torkkola 1993]

An efficient way to learn English grapheme-to-phoneme rules automatically

Torkkola's approach to G2P is using Kohonen's concept of Dynamically Expanding Context. Their method constructs rules that are most general in the sense of an explicit defined specificity hierarchy (the amount of context around the symbol to be transformed, weighted towards the right).

The corpus used for these experiments was 20,008-word corpus provided in the public domain by Sejnowski and Rosenberg, also used in the NETtalk experiments.

The difference with our models is that Torkkola did not take stress into account, and more importantly, they don not make use of the pronunciation at training time. The results reported are 90.08% letter accuracy, which is slightly lower than the results obtained by [Black et al. 1998].

Chapter 4
Proper name baseline

This chapter describes the data and tools used throughout this thesis. The baseline model is the most basic model, which uses only letter context features, and serves to compare all other models implemented in this thesis, and to better monitor improvement and usefulness of specific features.

The LTS rules for the baseline model, as well as for all other models in this thesis, were implemented following the steps described in the Festvox Manual (see http://festvox.org/).

The baseline achieves 54.08% word accuracy, which represents about an 8% relative improvement over the simplest method of assigning the most probable phone to each letter taking into account letter ngram context (see Chapter 2).

4.1. Data

For the proper name baseline and the rest of the experiments reported in this thesis, we used a list of proper names that originally came from Bell Labs' directory listings, which contains the 50,000 most frequent surnames and 6,000 names in the United States (as of 20 years ago), and which is now part of the CMUDICT with stress (version 0.4[17]). See Appendix B: Training data sample.

The list of 56,000 proper names was developed about 20 years ago by David Schultz and Beth Schulz (AT&T Bell Laboratories, Indian Hill Park) initially using LTS rules and

[17] Even if there already exists a newer version of CMUDICT, version four was chosen for comparability reasons, since the experiments in [Black, et al., 1998] were made using CMUDICT-0.4.out, and that allowed us to have a general baseline on top of the foreign names only baseline. This way, in order to prove our hypothesis correct, we only need to show improvement on these baselines, using the exact same version.

hand corrected by Jill Burnstein. This list effectively substituted Church's NAMSA (pronounced "name say") system, which was delegated to OOV names.

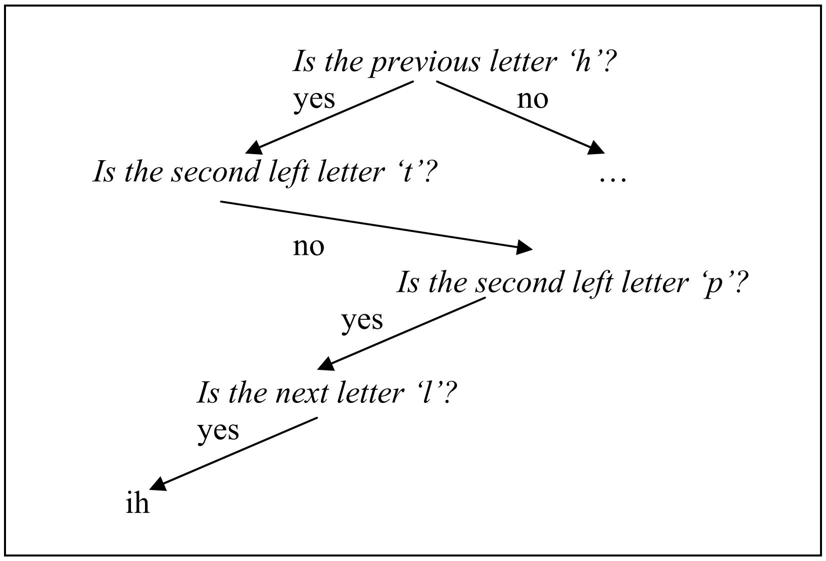
We held out every tenth name in the 56K list for testing and used the remaining 90% as training data. For a sample of the data, see appendix B.

4.2. Technique: CART

For the pronunciation models described throughout this thesis, we use the training data to build decision trees (DTs), also known as classification and regression trees (CART), to directly yield LTS rules. This technique is described in detail in [Breiman et al. 1984].

Decision trees are a symbolic, instance-based, learning approach. They constitute a greedy, top-down classification method, which has great flexibility and generalization capability. The idea is to split the data in such a way as to maximize generalization of the training data and prediction of unseen (held out and test) data. The basic component includes a set of YES-NO questions and a procedure to select the best question at each node to grow the tree from the root.

The basic YES-NO questions for LTS conversion asks about the letter n-gram context of the letter to be transcribed. Examples of this are "*Is the second left letter 'h'?*" and "*Is the third right letter 'g'?*". A possible (very simplified) path for the letter 'i', from the root node of the tree to one of the leaves, can be seen in figure 2.



Is the previous letter 'h'?
yes
no
Is the second left letter 't'?
…
no
Is the second left letter 'p'?
yes
Is the next letter 'l'?
yes
ih

Figure 2: a path in the decision tree to transcribe the letter 'i'

This portion of this imaginary tree would correctly generate *ih* for the first 'i's in the word *Philadelphia / f* ***ih****1 l ah0 d eh1 l f iy0 ah0 /* if we don't take stress into account.

For the baseline, the questions were limited to only the three letters preceding and the three letters following the letter under consideration, based on the techniques described in [Black et al. 1998] and used in [Chotimongkol and Black 2000]. When growing the tree, the question that yields the most entropy reduction (or equivalent the most information gain) is chosen.

If we allow the node to have complex questions (combination of primitive questions), the depth of the tree will be greatly reduced and the performance will be improved [Jiang et al. 1997]. For example, a complex question could be "*Is the second left letter is 't' and the first left letter is 'i' and the first right letter is 'n'?*". This can capture 'o' in the common suffix "-tion" and convert it to the right phoneme /*sh* ***ah0*** *n*/. Complex questions can also alleviate the data fragmentation problem caused by the greedy nature of the information gain algorithm. However, the CART technique used in Festival and thus in this thesis does not use complex questions, since they are very computationally expensive to train.

We mostly use the CART technique as classification tree and not as a regression tree in the experiments described here, i.e. a leaf of the tree corresponds to a specific pronunciation rather than to a distribution of pronunciations.

CARTs have been widely used in speech recognition, language modeling as well as many other machine learning techniques. They tend to be good classifiers. However, there are a couple of issues one should be aware of when using DTs: (i) at each node, they split the data and this tends to result in a data fragmentation problem; (ii) they have a soft bias of preferring shorter trees. For a detailed introduction to decision trees, see chapter 3 in Tom Mitchell's *Machine Learning* book [Mitchell 1997].

4.3. Letter-phone alignment

In general, the number of letters in a word and the number of phones in its pronunciation are not a one-to-one mapping. In English, letters map to epsilon (i.e. silence; letter is not pronounced), a phone or occasionally two phones. The following three mappings illustrate this:

(a) Monongahela	/ *m ax n oa1 ng g ax hh ey1 l ax* /
(b) Pittsburgh	/ *p ih1 t ε s b ε er g ε* /
(c) exchange	/ *ih k-s ch ε ey1 n jh ε* /

where (a) has the same number of letters as phonemes, (b) has several letters map to epsilon, and (c) has the letter 'x' realized by two phones / k-s /. Stressed is marked by adding a 1 after the stressed vowel.

Currently there are fully automatic methods of finding the best single letter alignments (scattered epsilon algorithm, trained using EM), which obtain results comparable to the ones achieved when using a hand-seeded method. However, in this thesis, we used the hand-seeded method described in [Black et al. 1998] to have more control over the letter to phone mappings. This method requires the explicit listing of which phones (or multi-phones) each letter in the alphabet may correspond to, irrespective of context. Vowels have typically a much longer list of potential phones.

For the proper names baseline, we started with 50 phones (including stressed phones), and added multi-phone combinations that appeared in the mappings with foreign name letters. The final number of phone combinations in the list of allowables was 101 (see appendix C), since native American speakers seem to slightly adapt their set of phones to the foreign sounds. Most of the extra phones were vowel variations, and some were adding a vowel between two consonants. For example, for the letter 'e', the standard set of phones would be: / *ih ih1 iy iy1 er er1 ax ah1 eh eh1 ey ey1 uw uw1 ay ay1 ow ow1 y-uw y-uw1 oy oy1 aa aa1* /, but for this particular domain, / *ax ey-ih ao ae* / were also added, since they appeared as the pronunciation of 'e' in some of the foreign names (e.g. erle / *ao r ax l* /); and for the letter 'c', the standard set of phones would be: / *k ch s sh t-s* /, and / *s-iy k-ax g* / were added to cover alignments in names such as is *cdebaca* ⇒ / *s-iy d ih b aa k ax* /, and *csaszar* ⇒ / *k-ax s aa sh er* /.

The hand-seeded algorithm takes the list of allowables and finds all possible alignments between each entry's letters and phones. A table of probabilities of a phone given a letter is estimated, and the entries are re-aligned scoring each possible alignment with the generated probabilities. Lastly, the best (most likely) alignment is selected. This algorithm generates alignments close to what would be produced by hand and it is very rare to find alignments that would be considered unacceptable.

Finally, the baseline uses the aligned training data to train a decision tree for each letter, and then, all the letter trees are merged into a single set of LTS rules, which is tested on the test data.

4.4. Testing and results

In this context, letter accuracy is defined as the number of letters that are correctly converted to epsilon, a phone, or multi-phone. Word accuracy is defined as the number of words where all phones (once resplit) match exactly all the phones in the entry of the test data, i.e. a heldout part of the CMU dictionary of pronunciation. And that does not necessarily mean we are comparing it against the correct American, educated pronunciation, since the CMUDICT has been typed in by several people along the years, and so it contains some inconsistencies and inaccuracies.

This method of calculating the error takes into account epsilon alignments (see section 4.3 for the details), so even if the generated string is correct, if the epsilon is aligned differently than in the test data entry, an error is counted, and the same is true for stress. For example, if we are taking stress into account and the system outputs the correct pronunciation but does not get the stress right, it will be counted as an error. In the worse case, it can happen that the system outputs the correct pronunciation, but since it is incorrectly encoded in the CMUDICT, this is counted as an error.

As mentioned in Chapter 2, the proper name baseline (PN-baseline) on a subset of the CMUDICT data, namely the list of 56,000 proper names included in CMUDICT, is slightly higher than the CMUDCIT baseline (compare CMUDICT (5) and PN-baseline (5) in table 2). Also as mentioned above, this might at first strike us as surprising, since proper names are believed to be harder to pronounce than common words (see reasons in section 1.3); however, this can probably be explained by the fact that we are already giving the information that they are proper names to the CART.

We only take the first pronunciation for every entry for both training and testing purposes. This may make our task harder, since if our model outputs a correct pronunciation for a given name which is not the first pronunciation listed in the lexicon for that entry, it will be counted as wrong when doing the testing, because it will just check whether is exactly matches the first pronunciation given in the CMUDICT.

A major source of error is typically the stress assignment. The way we measure accuracy on the test set is, if the stress is misplaced by the CART, or if the output of the CART is correct but the lexicon entry has the stress misplaced, then it counts as an error on at the word accuracy level. To confirm that stress has a great effect on word accuracy, we also ignore stress at the end, and report the results.

Lexicon	Letters NS	Words NS	Letters Stress	Words Stress
OALD (1)	96.27%	74.69%	95.80%	74.56%
OALD (5)	93.70%	63.15%	n/a	n/a
OALD (8)	92.89%	59.63%	n/a	n/a
CMUDICT (5)	**91.99%**	**57.80%**	**91.99%**	**50.14%**
BRULEX (French)	99.00%	93.03%	n/a	n/a
DE-CELEX (german)	n/a	n/a	98.79%	89.38%
Thai	n/a	n/a	95.60%	68.76%[18]
PN-baseline (1)	**88.72%**	**58.97%**	**88.72%**	**52.47%**
PN-baseline (5)	**89.02%**	**60.48%**	**89.02%**	**54.08%**
PN-baseline (8)	**88.70%**	**58.76%**	**88.70%**	**52.88%**

Table 2: Previous results plus results for the baseline model for proper names

In table 2, NS stands for "no stress", that is, we ignored stress on vowels; and (1), (5) and (8) are the stop values for the CARTs. If the stop value is 5, the leaves of the tree have at least 5 examples, that is, the tree overfits the data more than if the stop value was 8 or higher. In addition, as the stop value gets lower (overfitting increases), the size of the tree gets larger. It has been shown in [Black et al. 1998] that a stop value of 5 is optimal for

[18] This result takes tones into account [Chotimongkol and Black, 2000]. For predicting tones, hand-written rules were used, since it was straightforward (no ambiguity). Their results reveal that this statistically trained approach out-performs the traditional rule-based model for which the accuracy reported is only 65.15%. Chotimongkol and Black also found that the bigram model offers the best compromise between accuracy and space, and that syllable boundaries are useful in the prediction.

general lexicons. This table suggests that 5 is the right stop value for proper names as well, since both letter and word accuracy decrease.

There is always some variability in the way different transcribers transcribe different words, and one mapping is not necessarily more correct than another. This is particularly true for foreign words, which have been transcribed by different transcribers, or may have common segments pronounced differently for historical or regional reasons. Such inconsistencies will place an upper bound on the amount of improvement possible using the CMUDICT.

User studies carried out during this thesis, show that our training data contains about 15% noise[19] (misspellings, transcribing inconsistencies, wrong pronunciations); however, it is not feasible to hand correct all the errors, so we assume our data is correct for the purpose of the experiments described here. The reader should keep in mind, though, that this will lower the upper bound on performance of our models.

[19] See section 8.1 for a detailed description of how the user studies were conducted.

Chapter 5
First Approach: Adding language probability features

This chapter describes our first approach to improve pronunciation accuracy over the baseline model just described. As mentioned in the introduction (cf. 1.4), it is our belief that knowledge of language origin information can account for the differences in pronunciation of proper names. The training corpus is a mixture of different languages, and we want to find a way to classify it into language classes.

Our training data is transcribed with the pronunciation; however, it does not contain any language nor country of origin information. The first step to be able to build a pronunciation model with language origin features is thus to build a language identifier and run it on our list of names.

5.1. Letter Language Models and Language Identifier

To build a language identifier, we need to build language models[20] for all the languages we want to identify. Since we are working with strings of letters in isolation (list of names), the only way to discriminate between one language and another is to use letter frequency. We decided to use frequencies of trigrams (groups of three letters) for the purpose of discriminating between languages.

Letter language models (LLMs) effectively are relative frequency databases of letter trigrams. To build such letter trigram databases we need to collect multilingual data, and have a reasonable sized corpus for each language we want to be able to identify.

[20] A language identifier can also be built using filters as in [Vitale 1992] or other discriminating techniques.

5.1.1. Multilingual Data Collection

A priori, the language distribution of the training data was unknown, so we proceeded to hand label 1% of the training data (516 names) to have a better idea of what languages would be good to cover with the LLMs.

Our intuition is that closely related languages will have similar LTS rules, however, the criterion followed was to tag names as fine grained as possible, since it would always be much easier to collapse them later if needed (if a class does not have enough examples, or if there were not significant difference in pronunciation between two classes).

Unfortunately, we found that only 43% of that data could be tagged confidently[21]. The language distribution for that small portion of the data was the following[22]:

German (96), English (73), Italian (43), French (40), Polish (23), Swedish (13), Scottish (12), Spanish (10), Hebrew (7), Dutch (6), Catalan (6), Danish (5), Japanese (5), Irish (5), Russian (5), Chinese (2) and Portuguese (2)

We then proceeded to collect data for these languages (except for Irish and Scottish) plus a few more to build the LLMs and effectively build a language identifier and add the relevant features to the baseline CART.

After a significant amount of language specific preprocessing (converting all special characters to their ascii equivalent), the data used for the LLMs was the following:

[21] The data is ascii only, and all the accent marks and special characters (which are excellent clues for language identification) had been removed. Other reasons hand labeling was hard was that many of the names belong to many different languages, and that many of the names in the list were really quite rare.

[22] Because of the way we hand labeled the data, it might have resulted into slightly boosting the languages we are more familiar with and the languages for which we had native speakers to look at the list first.

- **18 general text corpora** from the *European Corpus Initiative Multilingual Corpus I* [ACL, 1994]:

  Croatian, Czech, Danish, Dutch, English, Estonian, French, German, Hebrew, Italian, Malaysian, Norwegian, Portuguese, Serbian, Slovenian, Spanish, Swedish and **Turkish**

  with sizes ranging from 255K to 11 billion words, and

- **7 proper names corpora** built by crawling the web automatically, using the *Corpusbuilder* [Ghani et al, 2001], and manually:

  Catalan, Chinese, Japanese, Korean, Polish, Thai and **Indian.**

  with sizes ranging 500 to 6198 proper names.

For languages with a non-roman alphabet, such as Indian, Chinese and Thai, we collected names in their romanized form. For Indian, where there is a strong tradition to use English in academic circles, finding romanized names on the web was not a problem; for languages like Chinese and Korean, this was a bit harder.

Not being able to label 57% of the data was not satisfying, and at this point, the need for an automatic language identifier became unavoidable. We first looked into the language identifier built by Michael Shamos and Eric Burns, which has been mostly trained on North European languages. The language identifier returned probabilities for 59% of the names (304) and for the small set of languages it was trained on, so the language distribution predicted by this classifier was very different[23]:

[23] Since Shamos and Burns' language identifier is also trained on Afrikaans, and Afrikaans is a Dutch dialect with a strong English influence, which allows more letter sequence than any of these languages alone, most words always had a high probability of being Afrikaans. I overlooked those probabilities for this application. In particularly, for the 26 words that had Afrikaans as the most probable language, I just counted the second most probable language for the distribution.

English (320), Dutch (92), Danish (62), German (60), Norwegian (33), Swedish (8)

Note, that the fact that a language allows a given sequence of letters (a particular word) does not mean that that word is a proper name in that language. In this case, I had a Danish native speaker go over the list of proper names, and he only recognized 5 Danish proper names in the whole list. However, Shamos and Burns' language identifier found that 62 names were most probably Danish, according to its trigram database.

Another problem with the labeling of proper names is that, determining what languages a name belongs to, is much harder than one might think at first, since many languages share a subset of proper names which are often written in the same way. The bias of the hand labeler unavoidably results in significantly boosting language they are most familiar with.

5.1.2. 25 LLMs and the Language Identifier

Every language specific LLM consists of a table of all the possible trigrams and their relative frequencies estimated from the corpus for that language (sliding trigrams) and taking into consideration whether it occurs at the beginning or at the end of the word. For the 25 LLMs, we used Laplace smoothing, i.e. we added one count to all the trigrams and normalized by the number of trigrams (tokens) times the size of the vocabulary (types). This only made a significant difference for the proper names corpora, since such corpora did not have enough data to reliably estimate all the trigrams for those languages.

The LI creates a LLM on the fly for the input word (or document) and, for every trigram in the input, it calculates the probability of it belonging to all the languages by multiplying them by the relative frequencies for those trigrams in each one of the languages (LLMs):

For each language database (L) calculate the following probability:

$$P(L \mid input) = \sum_{input\ trigram} \frac{C(trigram)}{\sum_{input\ trigram'} C(trigram')} \cdot P(trigram \mid L) \qquad (0)$$

where *C(x)* is the number of times *x* occurs, and input is the input name.

Then the LI sorts the languages by their reverse probabilities, i.e. it returns the language with higher probability for that name first. In short, given a name, a document or a list of names (in batch mode), the LI outputs a language distribution of the following sort:

```
classify.pl -t "Ariadna Font Llitjos"
catalan-pn:     0.12734
portuguese:     0.07164
spanish:        0.06871
hebrew-pn:      0.06252
french:         0.05533
swedish:        0.04472
polish-pn:      0.03987
english:        0.03711
italian:        0.03704
danish:         0.03480
indian-pn:      0.03315
thai-pn:        0.03271
serbian:        0.03244
norwegian:      0.03212
czech:          0.03064
croatian:       0.02958
slovenian:      0.02490
estonian:       0.02195
turkish:        0.02195
malaysian:      0.02007
dutch:          0.01637
korean-pn:      0.01586
japanese-pn:    0.01558
chinese-pn:     0.01242
german:         0.01203
```

where “-pn” stands for proper names, and is used to distinguish LLMs trained on general text from the LLMs trained on proper names only.

Our implementation is a variation of the algorithm presented in [Canvar and Trenkle 1994], which only takes trigrams into account, as opposed to n-grams for n = {1,2,3,4}, and assigns probabilities to the languages. The reason for only using trigrams is that, for language identification purposes, trigrams have been shown to suffice [Vitale 1991], [Church 2000] and [Sproat et al. 1999].

On the other hand, I realize that we would probably benefit from unigram and bigram distributions, as well as from 4gram and 5gram statistics, for they tend to be language specific. For example, languages like Portuguese and Catalan do not have the letters 'y' and 'k' and 'w'; Japanese does not have the letter 'l'; "-dorf" "-haus" and "-mann" are typical German suffixes. Neither of this is captured by our current LLMs, but could be easily incorporated using filters as in [Vitale 1991] or by using n-gram LLMs and implementing a backoff model for classification.

When running our Language Identifier (LI) over the 304 hand labeled names, I get the following distribution[24]:

German (41), Hebrew (31), Chinese (27), Malaysian (24), Korean (23), Polish (20), Danish (20), Dutch (18), Spanish (18), Indian (15), Italian (14), Japanese (13), French (12), Norwegian (10), Portuguese (9), English (9), Estonian (7), Swedish (7), Turkish (5), Czech (3), Thai (3), Slovenian (3), Catalan (3), Croatian (2) and Serbian (1).

Even though this looks much better than Shamos and Burn's language identifier distribution (cf. 5.1.1), the distribution of languages is actually quite different from the *truth* (hand labeled data). Just to give an example, there are 73 English names, but our LI only assigned English as the most likely language to 9 names only.

[24] Note that our LI tags all the names with language probabilities, unlike Shamos and Burns' language identifier, and also unlike humans.

From the hand labeled examples, we can estimate the accuracy rate of our Language Identifier. Running the LI over the hand labeled examples, we found that 59 names were tagged by our LI with right language as first, and 40 names were tagged with the right language as second. Overall, the LI assigned the right language, as first or second choice, 99 times. That means that the accuracy of the LI is only 33%, it assigns the right language, as either the first or the second choice, only one third of the times.

This number is rather low, and it would help our experiments to have a more accurate LI. However, this would involve a large amount of manual work into hand-tuning it, and we are interested in automatically building language classes to see how much we can improve pronunciation accuracy, without requiring neither a year of dedicated work, nor much linguistic knowledge.

5.2. Model 1: Language specific LTS rules

Our first model implements the most widely used approach to generate pronunciations for proper names taking language into account. First, it classifies a name using the Language Identifier, then it sends the name to the corresponding language specific LTS rules. This is the approach taken by [Vitale 1991], [Church 2000], [Jannedy and Moebius 1997] and [Yvon 1994].

The first step was to separate the language classified training data into 25 language specific corpora according to what was the most probable language, and to try training language specific LTS rules for each of the 25 corpora.

Unfortunately, It was immediately apparent that this could not work for all the 25 languages, since in spite of having about 50K transcribed names in the training set, there is not enough training data to reliably build a CART for all the languages. We could reliably detect (using positive filters):

– 5,863 English names (training set: 5,277 test set: 586)

– 2,818 German names (training set: 2,537 test set: 282)

– 1,312 French names (training set: 1,181 test set: 130)

– 1,251 Polish names (training set: 1,126 test set: 125)

–… (numbers go down quickly)

5.2.1. Results for model 1

Due to a data sparseness problem, the results obtained are only reliable for English and maybe for German (cf. chart 1). The pronunciation accuracy of the French CART is lower than Polish, even when it had a bit more data to train on. This can probably be explained by the fact that French LTS rules are harder than Polish LTS rules.

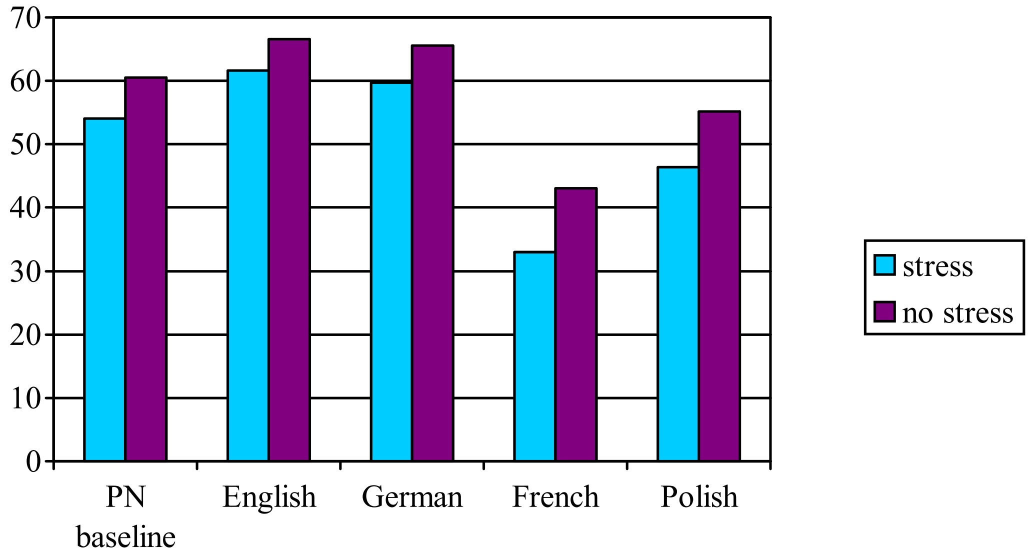


Chart 1: name accuracy for language specific LTS rules corresponding to Table 2

In chart 2, we show the results obtained after training English CARTs on decreasing amounts of data, which correspond to the same amounts of data we had for German, French and Polish (see above). These results show that there is a clear correlation between the amount of training data used to build the CART and pronunciation accuracy. Decreasing the amount of English data always resulted in a decrease of accuracy.

Languages	Letters NS	Words NS	Words Stress
CMUDICT	89.52%	57.71%	50.14%
PN-baseline	89.02%	60.48%	54.08%
English	91.89%	66.55%	61.60%
German	92.14%	65.6%	59.67%
French	83.89%	43.08%	33%
Polish	88.97%	55.20%	46.40%

Table 2: name accuracy for language specific LTS rules

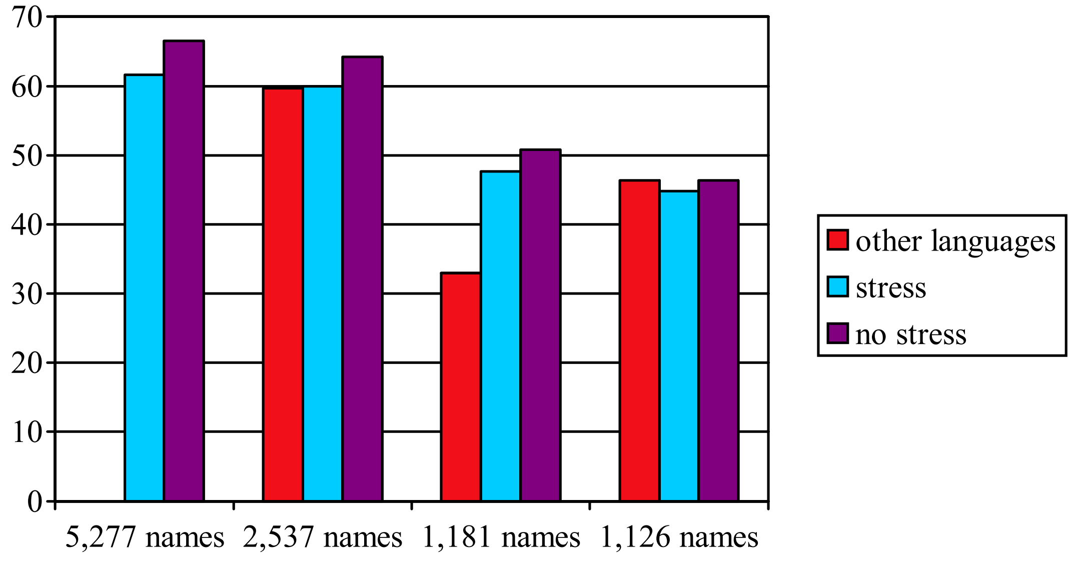


Chart 2: English CARTs results on decreasing amounts of training data, compared with German, French and Polish CARTs

When the English CART was trained on the same amount of data than the German CART (2,537 names), pronunciation accuracy went down by exactly the same percentage. Therefore, we can be quite confident that if we had double amount of German data, pronunciation accuracy for the German CART would go up to about 61.60%.

English CARTs	Letters NS	Words NS	Words Stress
5,277 names	91.89%	66.55%	61.60%
2,537 names	91.07%	64.18%	59.93%
1,181 names	87.57%	50.77%	47.69%
1,126 names	87.76%	46.40%	44.80%

Table 3: English CARTs results on decreasing amounts of training data, evidence for the sparseness problem

The decrease in accuracy for the English CART with the same amount of data than the French (1,181 names) was significant (more than 12%); however, pronunciation accuracy was not as low as in the case of the French CART.

When the English CART was trained on the same amount of data than the Polish CART (1,126 names), pronunciation accuracy went down slightly more than for the Polish data. This gives us the hint that having about 5,000 Polish transcribed names would most likely result in much higher pronunciation accuracy.

In addition the sparseness problem, there also is a language identification problem with this approach. A relatively small part of the data can be reliably classified as belonging to one language and not another. If the Language Identifier makes a mistake, the pronunciation is very likely to be completely wrong (imagine applying French LTS rules to a Japanese name).

Even if we had the sufficient training data to reliably build language specific LTS rules, there would be an efficiency issue, since instead of just having a set of LTS rules, we would have to store 25 different sets of LTS rules. [Vitale 1991] tried to overcome this problem by labeling specific rules with the languages they could be applied to, thus reducing the size of the model.

Even when using optimization techniques, this direct approach has yet another shortcoming. Because the system completely relies in the language identifier to be perfect, systems that have taken this approach end up having a very complex and sophisticated language identification component, which requires human intensive work.

This is clearly not the approach we would like to take, since our goal is to improve proper name pronunciation as automatically and efficiently as possible.

5.3. Model 2: Building a single CART

Alternatively to building LTS rules specifically for each language, we can use the language probabilities from the language identification process as features within the CART building process, allowing them to <u>affect the tree building only when their information is relevant</u> (recall that decision trees split the data according to what split is most informative, and so irrelevant features will not be selected as questions by the CART algorithm).

The intuition why this approach might work, is that by adding language probability features to the CART, we are adding information at the word level, rather than just about the local context of the word (like the baseline does). Even if the LLMs upon which the LI relies are also trigram based, the language probability is the result of multiplying the relative frequencies for all the trigrams in that name.

With this approach, we need only build one set of LTS rules. Therefore, the pronunciation model does not make a hard decision about how the name should be pronounced according to the Language Identifier, but rather passes on some of the features result form the LI to the CART, and the latter uses them in the classification if it estimates them relevant.

Given a name, the LI outputs the probabilities of that word belonging to all 25 languages. By adding features to the CART, we are effectively enriching the parameter space (the more parameters, the more data we need to reliably build a CART), so we want to be careful about what features to include to the baseline. Including all the language probabilities would clearly not be appropriate.

Therefore, we decided to add only 6 features to each name:

- Most probable language
- Highest probability
- Second most probable language
- Second highest probability
- Difference between the two highest probabilities
- Length of the word

Even if the second highest probability can easily be inferred from the highest probability and the difference between the two, in order for these features to be used by the CART, we need to pass them explicitly.

The motivation behind having the length of the name as a feature in our model is that it's a good discriminator between Chinese surnames, which tend to be very short, and Thai surnames, which are typically very long.

If you are wondering how the decision tree knows what features are related to what other features (for example, the 2nd feature is related to the 1st feature, but we do not input this information to the CART), the answer is that the DT automatically picks on the correlations among features, that is what DTs are good at doing. Now, it can be that, in practice, it combines the 2nd feature with the 3rd feature in a path. In such cases, the reason for this counter intuitive splitting is that it probably improves performance, so it does not really matter if it makes little sense.

An example of aligned labeled data, which is the input to the CART, can be found in figure 3.

```
( aaker nil ( ε aa1 k ε er )  norwegian  0.13770   swedish. 0.13359   0.00411  5)
( aamodt nil ( ε  aa1  m ax ε t )  norwegian  0.13483   japanese-pn  0.09314   0.04169   6)
( aardema nil ( ε  aa r d eh1 m ax )  dutch  0.10950   estonian   0.09347   0.01603   7)
( abair nil ( ax b eh1 ε r )  hebrew-pn   0.23815   turkish  0.08722   0.15093   5)
( abate nil ( ax b ey1 t ε )  norwegian   0.08586   portuguese  0.08076   0.00510   5)
( abdalla nil ( ae1 b d ae1 ε l ax )  catalan-pn 0.11733  italian.train 0.10279   0.01454  7)
( abel nil ( ey1 b ax l )  hebrew-pn  0.15192   catalan-pn  0.14440   0.00752  4)
( abella nil ( ax b eh1 ε l ax )  catalan-pn  0.16144   italian  0.10516   0.05628   6)
( abendroth nil ( ae1 b ih n d r ao th ε ) hebrew-pn  0.11470 indian  0.09089   0.02381  9)
...
```

Figure 3: aligned labeled data, input to the Model 1 CART

Note that for the most part the LI is pretty reasonable but it makes some mistakes (e.g. it assigns Japanese as the second most probable language to “aamodt”, the second name on the list)

A small example to illustrate the resulting LTS rules, follows:

```
(c
 ((p.name is s)
  ((p.p.name is t)
   ((ch))
   ((n.name is h)
    ((n.n.name is o)
     ((lang is german) ((_epsilon_)) ((k)))
     ((_epsilon_)))
    ((k))))
         …
```

If the letter context of ‘c’ is ‘scho’ and the name is labeled as being German, the ‘c’ is silent, otherwise it is pronounced /k/. Examples from the data that would be generated by this rule are:

Schooley + English $\Rightarrow$ / *s* ***k*** *uw1 l iy* /

Schobert + **German** $\Rightarrow$ / *sh aa1 b er t* /

Schoenemann + **German** $\Rightarrow$ / *sh aa1 iy n m ax n* /

Sure enough, there are some inconsistencies in the training data:

Schau / ***sh*** *ow1* / (+ German)

Schaun / ***s k*** *hh aw1 s* / (+ German)

Different levels of assimilation also pose a challenge for LTS rules:

Bombardier / *b aa1 m b er d* ***ih1 r*** /

Cartier / *k aa1 r t* ***iy er*** /

Unfortunately, the language probability features for the 25-language identifier did not yield significant improvement (55%), so we also tried building a 4-language classifier for English, German, Polish and French using only the names that had been reliably detected as belonging to one of these four languages (10,121 names). The 4-language model did yield almost a 3% improvement over the baseline (56.94%); however, since it included Polish and French for which there might be not enough training examples (see 5.2), we also built a 2-language model with English and German names.

The 2-language model achieved a significant improvement of more than 7% over the baseline (61.18%), and was only trained on confidentially labeled English and German names (7,814). There might be several reasons for this. For one thing, binary classification problems are always much easier than 10-fold or 25-fold classification problems. In the 25-language model, there is a probability of one over 25 to correctly classify a name (1 / number of languages), whereas in the 2-language model, there is a much higher probability (0.5) of assigning the right class to a name. In the latter case, all the decision tree has to learn from the language features is whether a name is German or not.

Moreover, the 4-language model and the 2-language model were not trained on all the training data, but only on a small, cleaner subset of it, which could be confidently classified as belonging to those languages.

In spite of it being an oversimplification of the actual problem we are trying to solve, the improvement yielded by the 2-language model does show the feasibility of the indirect approach to improve pronunciation accuracy of proper names. These smaller models trained on cleaner data constitute evidence that given enough names which could be safely classified as belonging to one of the 25 languages, we would probably achieve significant improvement from the baseline.

5.3.1. Results for Model 2

Even tough the 25-language pronunciation model did not achieve any significant improvement, smaller models with a lower number of languages show that the indirect approach of adding language features to the CART yields substantial improvements:

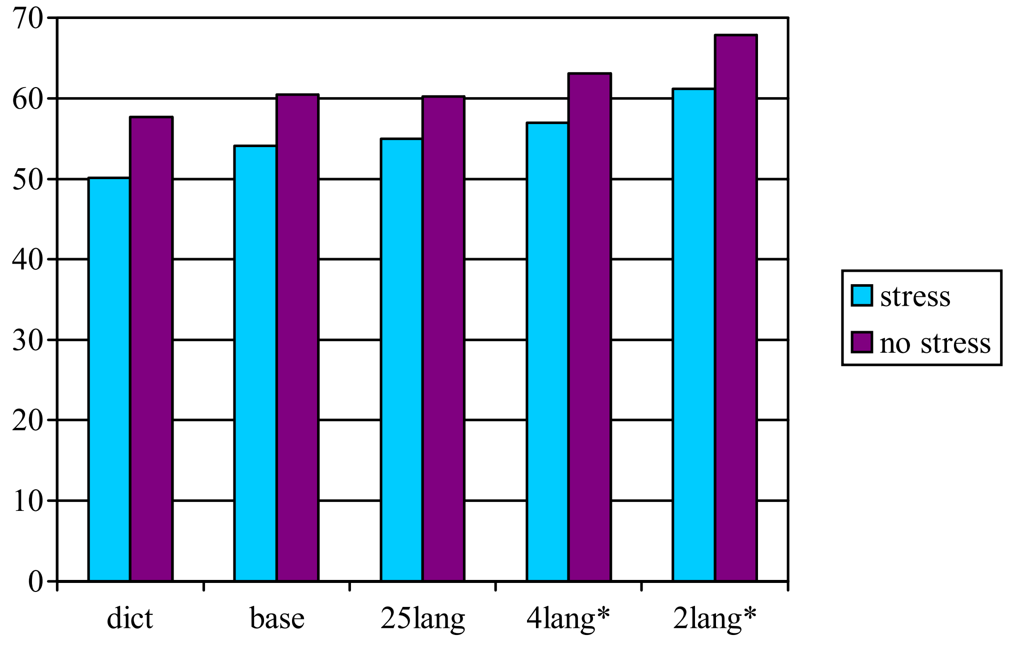


Chart 3: name accuracy of several variations of Model 2 indirectly incorporating language information[25]

[25] The stars by the 4-language and the 2-language models indicate that they were trained on a cleaner subset of the data.

Languages	Letters NS	Words NS	Words Stress
CMUDICT	89.52%	57.71%	50.14%
baseline	89.02%	60.48%	54.08%
25 languages	89.11%	60.23%	55.10%
4 languages*	90.08%	63.08%	56.94%
2 languages*	92.31%	67.87%	61.18%

Table 4: name accuracy of several variations of Model 2 indirectly incorporating language information

All the accuracy numbers reported in Table 4 were achieved using a stop value of 5. Since the parameter space had grown from the baseline (6 new features were added to the CART), we also did some experiments with a stop value of 8 for the baseline and the 25-language model.

	Letter	Word with stress
baseline (5)	89.02%	54.08%
baseline (8)	88.70%	52.88%
25-languages (5)	89.11%	55.10%
25-languages (8)	88.95%	54.17%

Table 5: name accuracy for different stop values

Table 5 shows that, even with a larger parameter space (25-languages model), there is probably no overfitting when using a stop value of 5, and that a higher stop value (8) results in a decrease on performance.

Ideally, we would like to estimate how well Model 2 will work given perfect language identification. In practice, however, hand labeling the training data turned out to be very hard and it would be too expensive to hand label enough examples to reliably train a CART on them. The 4-language and 2-language experiments done on a subset of the training data are an attempt to get a better feel for the upper bound of this approach.

Chapter 6
Second Approach: Adding language family features

It is conceivable that having 25 different LLMs adds too much confusability to the decision tree. Intuitively, an educated native American English speaker may know whether a name is likely to be Asian vs. Germanic, but might not really be able to tell whether it is Chinese or Korean, say.

This supports the hypothesis that knowing the family of languages might be enough for improving pronunciation accuracy. Hence the second approach of this thesis is to add language family information to the CART.

In order to have language family features, we proceeded to build 5 family LLMs which contain all the data for the previous 25 LLMs:

- **ASIAN**: Chinese, Japanese, Korean, Malaysian and Thai
- **GERMANIC**: Danish, Dutch, English, German, Norwegian and Swedish,
- **ROMANCE**: Catalan, French, Italian, Portuguese and Spanish
- **SLAVIC**: Croatian, Czech, Polish, Serbian and Slovenian
- **OTHER**: Estonian, Hebrew, Indian and Turkish

As it is to be expected, a 5-fold classifier is much more accurate than a 25-fold classifier. In particular, when classifying the 304 hand labeled examples as belonging to one of these 5 families, we found that classification accuracy went up to 73%. This is a 40% improvement over the 25 languages Language Identifier. We are now in a better position to rely on the LI than the previous two models.

6.1. Model 3: Family specific LTS rules

Since we have less of a sparseness problem when classifying according to language families, it is worth building 5 CART, one per family, and see what effect this has on pronunciation accuracy.

We followed the same procedure as for model 1 (cf. 5.2), but this time loading the 5 family LLMs described above, instead of the 25 language specific LLMs, to the Language Identifier. We run the LI on all the training data, and separated the names into family subcorpora, according to what family had the highest probability.

Once we had 5 family subcorpora of the training data, we proceeded to train 5 different CARTs, one per family. The results varied widely depending on the family.

6.1.1. Results for model 3

In general, the results obtained by directly using the language probability features to separate the data into 5 families, did not improve pronunciation accuracy. Only the Germanic CART achieved significant improvement over the baseline, 3.33% improvement.

As we can see from both the chart and the table, the CART for the Romance, Slavic and other languages yielded very low pronunciation accuracy, well below the baseline. This can be explained by the fact that Germanic languages have easier LTS rules, but most likely, the poor performance of the other LTS rules is due to the noise introduced by the Language Identifier.

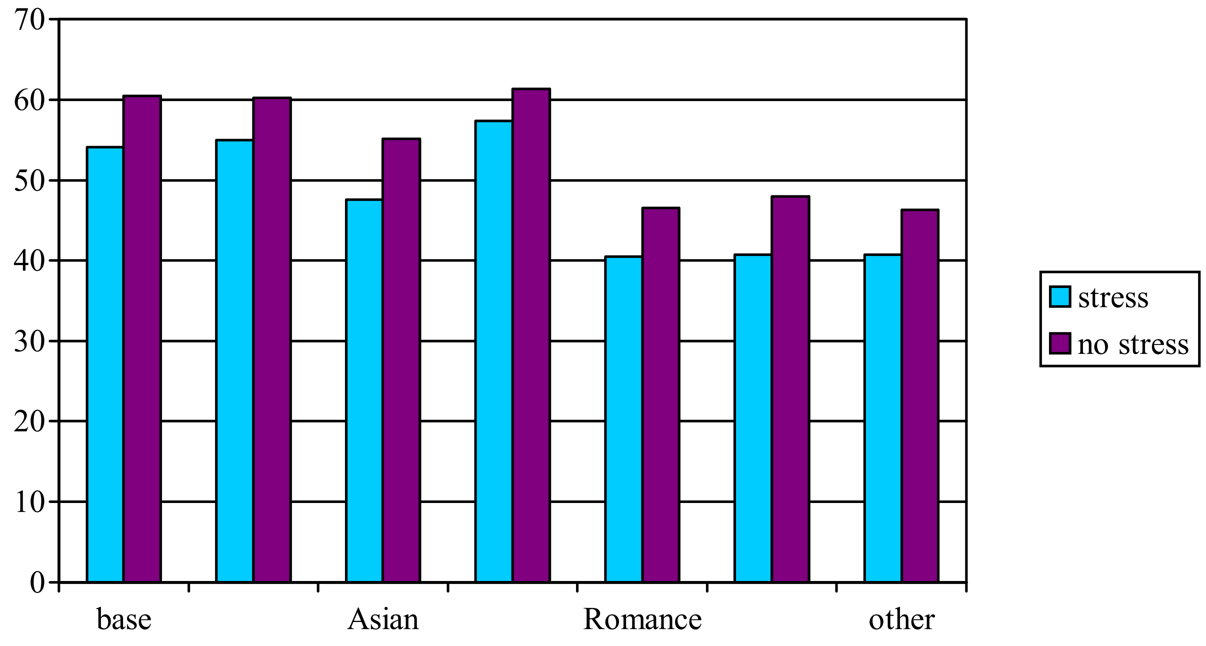


Chart 4: family specific CARTs compared with the baseline and model 2

Families	Letters NS	Words NS	Words Stress
Asian	84.50%	55.16%	47.60%
Germanic	90.47%	62.12%	57.41%
Romance	84.90%	49.53%	43.53%
Slavic	85.24%	49.24%	41.95%
other	85.38%	49.22%	42.55%

Table 6: name accuracy of the 5 family specific language models

The results from model 2 (cf. 5.3.1) when we built an English and German CART, confirm this. The 2-language CART was trained only on examples from two closely related languages. The results obtained by that model are 6.48% higher than the ones achieved by the Germanic CART. The difference between the 2-language CART and the Germanic CART is not only in the number of languages, but also in the kind of training data. Recall that the 2-language CART in model 2 was trained on names we could confidently classify as English or German; whereas all the CARTs presented in this chapter were trained on what the LI classified as belonging to one of the language families. If we had a more accurate Language Identifier, the results for the family CARTs would probably go up.

6.2. Model 4: Indirect use of the language probability features

If we substitute the 25 LLMs by 5 family LLMs model as input of the Language Identifier (cf. 5.1.2), there is a little bit more than 1% improvement from the baseline model (from 54.08% to 55.02%). But there is no improvement with respect model 2 (25 LLMs). So we tried combining the language family information with the 25 LLM information.

6.3. Model 5: A 2-pass algorithm

The second approach attempts to benefit from both the generalization from family languages as well as the 25 language specific LLMs. The 2-pass algorithm first classifies the training data using the 5-family-language model (which has 73% chance of assigning the right label), and then loads the language specific LLMs for the languages corresponding to that family to get the features that are going to be passed to the CART. The whole process of the 2-pass algorithm is illustrated in Figures 4 and 5.

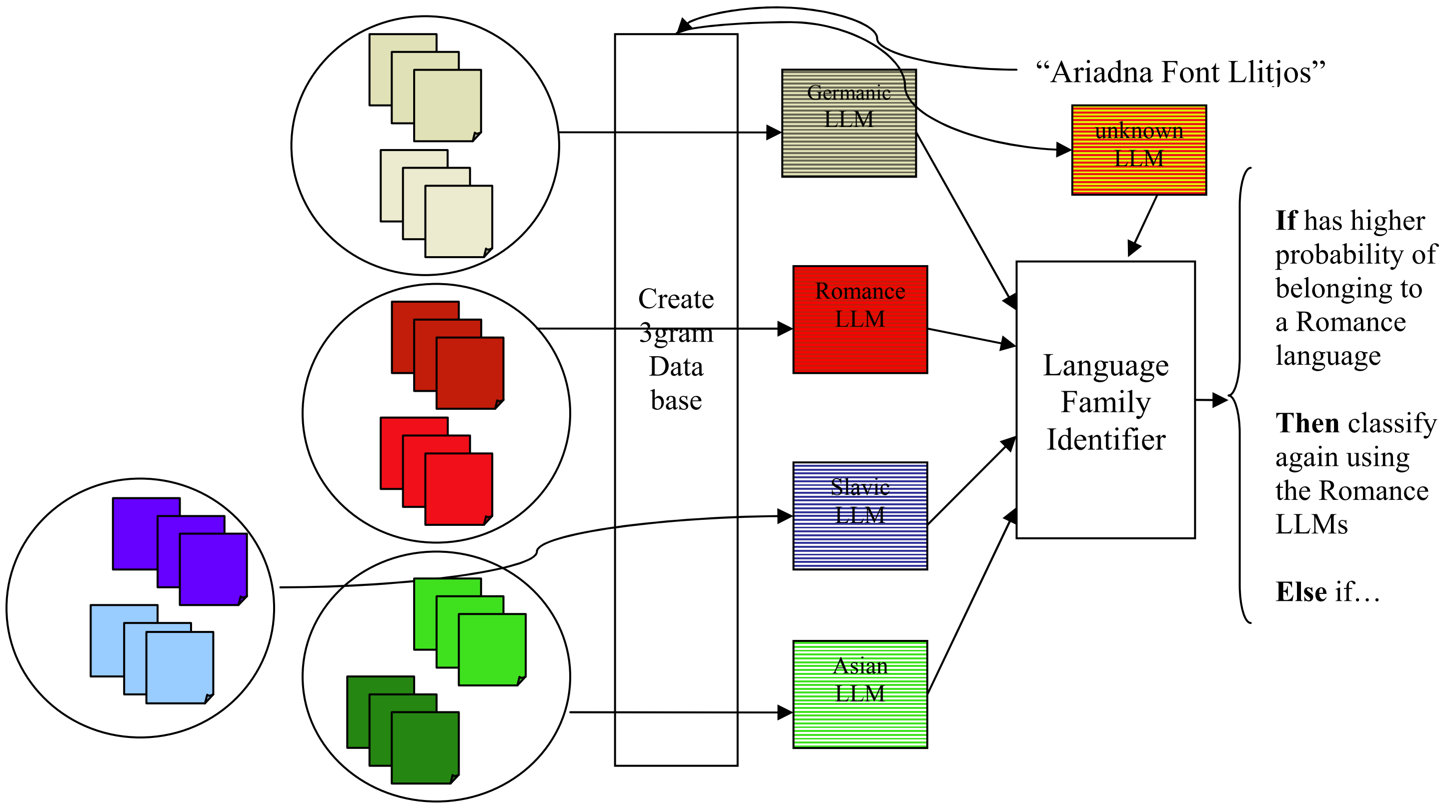

"Ariadna Font Llitjos"
Create 3gram Data base
Germanic LLM
Romance LLM
Slavic LLM
Asian LLM
unknown LLM
Language Family Identifier
If has higher probability of belonging to a Romance language
Then classify again using the Romance LLMs
Else if…

Figure 4: First step of the 2-pass algorithm for Model 2

Once, we know what family a name belongs to, the LLMs for the languages that belong to that family are loaded to the LI and the names are further classified as in Model 1 (see Figure 5).

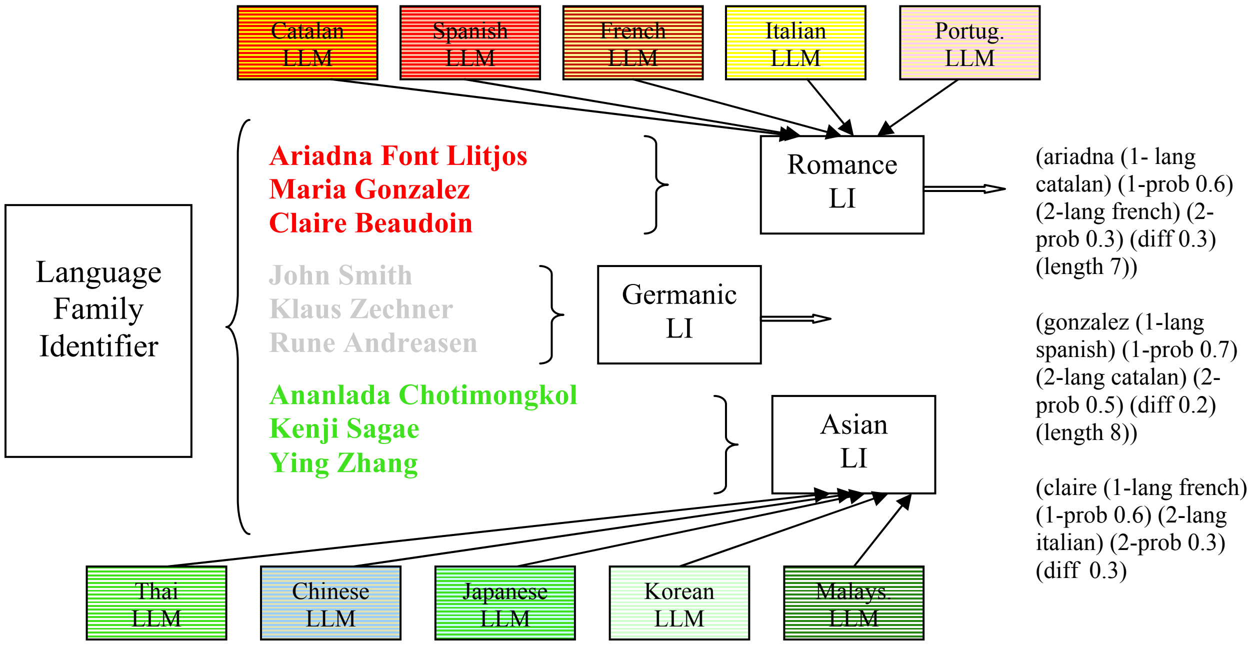

Catalan LLM
Spanish LLM
French LLM
Italian LLM
Portug. LLM
Language Family Identifier
Ariadna Font Llitjos
Maria Gonzalez
Claire Beaudoin
Romance LI
John Smith
Klaus Zechner
Rune Andreasen
Germanic LI
Ananlada Chotimongkol
Kenji Sagae
Ying Zhang
Asian LI
(ariadna (1- lang catalan) (1-prob 0.6) (2-lang french) (2-prob 0.3) (diff 0.3) (length 7))
(gonzalez (1-lang spanish) (1-prob 0.7) (2-lang catalan) (2-prob 0.5) (diff 0.2) (length 8))
(claire (1-lang french) (1-prob 0.6) (2-lang italian) (2-prob 0.3) (diff 0.3)
Thai LLM
Chinese LLM
Japanese LLM
Korean LLM
Malays. LLM

Figure 5: Second step of the 2-pass algorithm for Model 2

Unfortunately, this more elaborated model does not have any further significant impact on name accuracy (55.22%) with respect to models 2 and 4. However, by looking at a few labeled examples, language identification is more accurate than when using 25-language model (see figure 6).

```
( aaker nil ( ε aa1 k ε er )  norwegian  0.21012   swedish  0.20384   0.00628   5)
( aamodt nil ( ε aa1 m ax ε t )  norwegian  0.36645   dutch.train  0.25018   0.11627  6)
( aardema nil (ε aa r d eh1 m ax )  dutch  0.33899   norwegian  0.24076   0.09823   7)
 ( abair nil ( ax b eh1 ε r )  hebrew-pn  0.63557   turkish  0.23277   0.4028   5)
( abate nil ( ax b ey1 t ε )  norwegian  0.24919   german  0.18534   0.06385   5)
( abdalla nil ( ae1 b d ae1 ε l ax )  italian  0.29033   catalan  0.24526   0.04507  7)
( abel nil (ey1 b ax l )  spanish  0.39254   catalan  0.31410   0.07844   4)
( abella nil (ax b eh1 ε l ax )  italian  0.28800   catalan  0.23570   0.0523  6)
( abendroth nil ( ae1 b ih n d r ao th ε )  german  0.26986   danish  0.20575   0.06411   9)
…
```

Figure 6: aligned labeled data result from the 2-pass algorithm, input to the Model 2 CART

If we compare figure 6 with figure 3 (in section 5.3), we can see that the second best language using the 2-pass algorithm is no longer Japanese, but rather Dutch, which is more reasonable.

Unfortunately, the fact that language families allow for a coarser classification, and thus classification accuracy probably goes up, does not seem to improve pronunciation accuracy.

6.4. Results for models 4 and 5

Chart 5 and Table 7 show the results for the baseline model, model 2, model 4 and model 5. It is interesting to note that models 4 and 5 are comparable to model 2, and thus improve pronunciation accuracy by about a percentage from the baseline model.

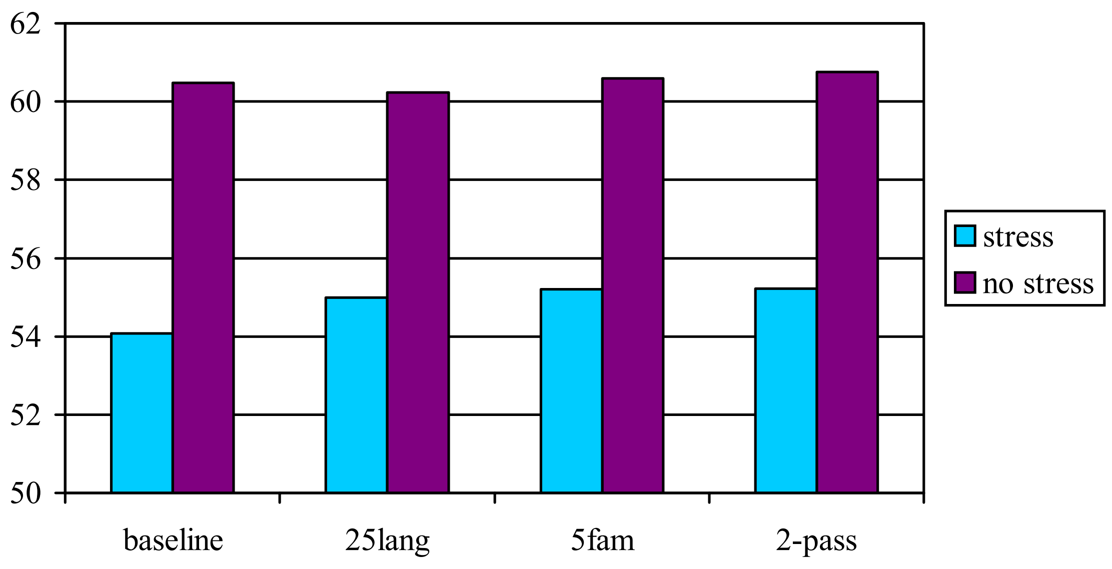


Chart 5: Comparing name accuracy of model 2 and the models 4 and 5

That model 2 (25 languages) and model 4 (5-fam-languages) are comparable means that making the finer distinctions between the 25 languages does not result in an improvement on pronunciation accuracy, and that the same performance can be achieved by a coarser grained classification of only 5 languages.

Languages	Letters NS	Words NS	Words Stress
CMUDICT	89.52%	57.71%	50.14%
baseline	89.02%	60.48%	54.08%
25 languages	89.11%	60.23%	55.10%
5 fam-languages	89.20%	60.60%	55.02%
2-pass algorithm	89.24%	60.76%	55.22%

Table 7: Comparing name accuracy of model 2 and the models 4 and 5

That they are comparable with model 5 (cf. 6.3) means that the 2-pass algorithm does not yield any further significant improvement over the simpler models 2 and 4. The 2-pass algorithm, however, does seem to result in a more accurate classification. Nevertheless, if the first pass makes a mistake, the name will only be classified within the wrong family, and with the current implementation, we can not recover from this error.

A possibly better approach of doing this, which remains as future work, is to set a threshold (determined empirically) for the output of the second pass. If the highest probability of a language within the language family is lower than the threshold, then we backup and classify the name using the LLMs within the family with the second highest probability in the first pass.

Chapter 7
Third approach: Unsupervised LLMs

This chapter presents our third approach to improving pronunciation accuracy of proper names. This is the most novel approach presented in this thesis and it is still in development.

Traditionally, people working on grapheme to phoneme conversion have mostly looked at the written words, but not at the actual pronunciation (see literature review in chapter 3). Some researchers have reported a gain of a couple of percentage points in accuracy when incorporating the output phoneme(s) previous to the grapheme under consideration ([Yvon 1994], [Jiang et al.1997]).

We would like to find a data-driven approach which uses the pronunciation as well as the name at training time, and we do not wish to limit the pronunciation data to the 3 previous phones as done in [Jiang et al.1997], but rather to take full advantage of it.

The goal is to find a way to automatically cluster words that are pronounced in the same way (using the same set of rules) together. This is an interesting machine learning problem since we want to cluster mappings of words and pronunciations (and not only words or only pronunciations), and we are after cross-linguistic differences in mappings. This should cluster words belonging to the same language together, possibly would merge close related languages, and might put Japanese and Italian in the same cluster.

One of the things we have been looking at is unsupervised methods for clustering proper names in a pronunciation maximally consistent way. Namely, we would like to have clusters that contain all the names with a consistent pronunciation, in other words, that define a unique set of LTS rules.

7.1. Unsupervised clustering

Clustering is the grouping of words (or anything else) into *equivalence* classes that reflect commonality of some property. The main use of clustering in NLP is thus generalization. The basic idea is that we induce the classes from the data, instead of grouping the data in certain predetermined ways.

Another way of partitioning objects into groups is *classification*, the difference is that classification is *supervised* and requires a set of labeled training instances for each class. We do not know a priori what is the right number of classes, neither what those classes are (that is exactly what we want to learn through the clustering process), so ours is an *unsupervised* process.

Clustering of words has been done by a number of people before. [Brown et al. 1992] used a bottom-up algorithm using loss of average mutual information (MI)[26] to obtain syntactically (and somewhat semantically) motivated classes, and thus reduce the number of parameters when trying to predict the next word.

[Pereira and Tishby 1992] implemented a top-down, relative entropy-based algorithm to compare the distributions for two nouns (they extracted verb-object relations from a corpus, and the context vector for a noun contained the number of times each verb took the noun as its direct object). In this case, the clustering starts with one class, and splits it successively until the desired number of classes is realized. This creates a tree structure of clusters, and as one moves towards the leaves of the tree, the classes become more clearly semantic in nature.

[Grefenstette 1993] tried a similar approach, but characterized nouns by more properties than just the verbs for which they serve as an object. He applied a partial parser to find the major syntactic relations in the text, and stored all the other words that were related to

[26] As we cluster things together, we loose specificity in our predictions and thus the average MI decreases. The goal of [Brown et al. 1992] algorithm is to make this decrease as small as possible at every merge.

each noun. The metric used was a weighted Tanimoto measure [Tanimoto 1958]. The binary Tanimoto version is defined as the number of attributes shared by x and y, divided by number of unique attributes possessed by x or y.

As a first approach to automatically and efficiently use both the pronunciation as well as the name at training time, we implemented a bottom-up unsupervised clustering algorithm, also called *agglomerative clustering*, initially inspired by the [Brown et al. 1992] algorithm.

This is the reverse process of splitting the data into pronunciation coherent subcorpora (top-down clustering). The reason why we decided to implement a bottom-up algorithm instead a top-down one is efficiency. When doing top-down clustering, we have, in principle, an exponential number of ways to split the data (it is $O(C^N)$), whereas when merging with a greedy algorithm, we can only merge existing clusters, so there are only $O(C^2)$ ($C \cdot (C-1)/2$) merges to consider at any time (where C is the number of classes).

When running the bottom-up clustering algorithm, the initial 46,478 clusters resulted into 459 maximally consistent, final clusters. The final number of clusters indicates that if we were to continue merging, we would have to allow clashes between clusters.

When looking at the final clusters, we found that, for the most part, clusters contained several groups of consistently pronounced names. Cluster 1 shown in figure 7, contains several words that end in –ce pronounced / *s ε* /, as well as several words that end in –ett pronounced / *eh1 ε t* /. The reason why these two, apparently well differentiated, groups ended in the same cluster is that they did not have any clashing vector in common (cf. 7.2 and 7.4). The clustering process occurred as follows: after cluster 1 was initialized with *Alyce*, several other words with the –ce ending were merged to it. At some point, the name *Phettepla**ce*** was merged with all the –ce ending names (so far, so good). It so happens that *Phe**tt**eplace* also contains the trigram *ett*; so, after that, words containing the trigram *ett* became good candidates for merging with cluster 1, and since there were no clashes, they merged with it.

Cluster 1

alyce, cruce, dulce, hulce, salce, boyce, joyce, loyce, noyce, royce, roylance, cayce, dyce, nyce, dyce, nyce, dyck, teneyck, beatrice, dapice, delguidice, guidice, loguidice, bernice, perniciaro, bryce, darcey, erceg, kerce, pearce, pierce, perce, pesce, perce, pesce, peshek, anstice, eustice, brace, eustace, crace, face, pace, **phetteplace**, bracey, gracey, racey … truett, truex, truly, barlett, marlett, scarlet, collett, mallet, malloch, connett, monnett, coslett, hewlett, kett, shifflett, shiflett, shiplett, shiflett, shiplett, triplett, barnet, dejarnett, derossett, burtnett, dennett, gennett, jennett, senne, sennett …

Figure 7: Shows more than half of the names that were merged together in final cluster 1.

This is actually very common, and most clusters contain several distinct letter patterns. This is probably not how a linguist building LTS rules would group the training data, however, it might be that such clusters are still useful for determining the pronunciation of proper names.

7.2. Basic data structure

Since there is no literature on how to cluster words and their pronunciation, it is not clear what is the best way to do it.

First, we need to find the basic data structure that will lead the clustering process. In the field of natural language processing, unsupervised clustering algorithms have typically been used to cluster words into more or less syntactically and semantically motivated classes.

Our problem is fundamentally different from other NLP problems unsupervised clustering has been applied to. The word is probably not the unit we would like to apply a similarity measure to, and we want to include the pronunciation in the picture.

As a first approximation to the *right* unit, we split the names into subunits that could function as basic data structure in the clustering process. As subunits, we choose vectors of length 4 containing a letter trigram and the phone aligned to the middle letter (after the alignment process has taken place, see section 4.3).

The first step of the clustering algorithm, then, converts each word into a set of vectors of the from: L1L2L3P2, where L1L2L3 represent a letter in context (trigram), and P2 represent the phone aligned to L2.

We then proceed to initialize each cluster with all the vectors that constitute a name and its pronunciation, you can see an example of this in figure 8.

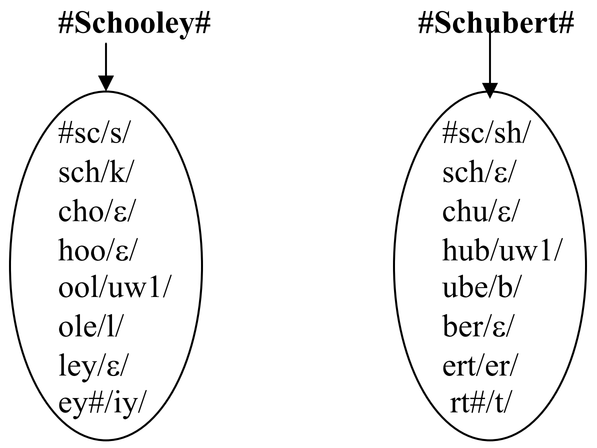


Figure 8: initial clusters containing all the vectors for a given name

The unsupervised clustering algorithm is sketched in figure 9; the following sections will define similarity measure and *clash*. The algorithm terminates when there are no more clusters that can be merged. In our case, we started with 46,478 initial clusters and we ended up with 459 clusters.

```
1. Convert each word into sorted vectors of the type: L1L2L3P2
2. Initialize each cluster with all the vectors corresponding to one of the words
3. Calculate the similarity between all possible cluster pairs
4. Consider merging the next two most similar clusters
      a. Compare all vectors in one cluster to all vectors in the other cluster
      b. If there aren't any clashes between vectors in the two clusters, merge (and
         delete) them and add the new cluster to the set of clusters
      c. Else, go back to 4
```

Figure 9: bottom-up, greedy unsupervised clustering algorithm

Clustering is a computationally intensive task. The worse case complexity for the algorithm described in table 9 is $O(V^5)$, since we need to calculate the similarity between any two clusters (V^2) after each merge (V-1; the number of clusters is reduced by 1 with each merge), and we need to sort the similarity scores for all possible cluster pairs (V^2). In this case V = 50,000 and the upper bound on complexity would be $3.125 \cdot 10^{23}$ (larger than the number of seconds since the beginning of the Universe).

As it will be described below, each comparison of two clusters involves comparing all the vectors in one cluster to all the vectors in the other cluster. Since there are 27 letters and 101 phones, the worse case number of vectors that would have to be compared each time is be $27^3 \cdot 101$, i.e. almost 2 million. In practice, the number of vectors per cluster is much lower than that (the number of letter combinations and letter-phone alignments is much smaller); final clusters, which are the larger ones, have only about 200 vectors on average.

Even then, the algorithm in table 9 has an upper bound on complexity of V^5, so it would be unfeasible to run it. There are some tricks that can be used to reduce the complexity of this algorithm and speed up the process, which will be described in section 7.1.3.

The reader might be wondering why limit the vector to containing only a letter triagram, instead of a 4gram or a 5gram. The reason for this is more practical than principled. Since we already have trigram LLM at our disposal, we would like to cluster the names in a way that, at testing time, we can run the Language Identifier on the names and assign them to the right cluster.

It would be interesting to explore taking into account more context and weight vectors by their length (shrinkage or backoff smoothing), but this is left as future work.

7.3. Similarity measure

Once we have all the 46,478 initial clusters ready, we need to establish a merging criterion. Typically the merging criterion is just to merge the two most similar clusters next, where similarity is defined with mutual information, distance in the vector space (Euclidean, cosine, etc.), Hamming or Levenshtein distance or Manhattan distance.

The first approach we took to this problem was to define similarity as **maximum overlap**, namely, the number of vectors two clusters have in common, which can be seen as a **simplification of mutual information**, as will be described below.

The intuition behind this similarity measure is that we want to try to merge those clusters which have most in common first. Take two (consistently pronounced) versions of the same name, *Sejnowski* and *Sejnowsky* say, we would like them to be merged first, since they should definitely end up in the same cluster. Such very similar words will have most units in common, probably all the vectors will be the same except for one or two, so our similarity measure should give these two clusters a very high score, which will lead to consider such clusters for merging early in the process.

More technically, mutual information (MI) tends to be high for similar units, since knowing one will be very informative about the other. If we apply the MI measure to clusters, we should get a high MI score for clusters that are most similar, in the sense that have many units in common.

For a shallow introduction to MI, as it relates to this problem, see chapter 9 in [Charniak 1993]; for a basic introduction to Information Theory (IT), including MI, see section 2.2 in [Manning and Schuetze 2000]; for a more thorough introduction to IT, see chapter 2 in [Cover and Thomas 1991].

MI captures both the amount of overlapping as well as the size of the Universe (n), i.e. the boundaries. For our problem, the Universe is fixed, and corresponds to the vectors contained in the 46,478 names.

MI between two clusters X and Y is defined as:

$$\sum_{X,Y} p(X,Y) \log \frac{p(X,Y)}{p(X)p(Y)} \tag{1}$$

Very often the expectation over X and Y is omitted from the MI formula, since what we are interested in knowing is the case where both random variables, X and Y, are equal to 1, rather than also considering the cases where X and Y are both 0; X is 0 and Y is 1, and Y is 1 and X is 0 (p(X=0, Y=0), p(X=0, Y=1), p(X=1, Y=0), respectively).

In other words, we are mostly interested in the case where the vectors are both in X and Y, and don't care so much for the cases where vectors are in neither X nor Y, in Y but not in X, or in X but nor in Y.

Therefore, for our purposes, we can simplify the MI formula and obtain:

$$\log \frac{p(X,Y)}{p(X)p(Y)} \tag{2}$$

since all we need is determine the amount of overlapping that there is between X and Y, i.e. the number of vectors that are both in X and in Y.

Moreover, since all we are interested in is comparing the amount of overlap between all the cluster pairs, and the logarithm is a monotonic function, we can omit the log from the formula as well, thus obtaining the following approximation to MI:

$$\text{sim(X,Y)} = \frac{p(X,Y)}{p(X)p(Y)} \tag{3}$$

where the maximum likelihood estimator (MLE) of $p(X,Y)$, is the number of vectors that are both in X and Y divided by n (total number of vectors in the Universe), and $p(X)$ is the number of vectors that are in X divided by n:

$$\text{sim(X,Y)} = \frac{\frac{\#\, vectors \quad in \ X \cap Y}{n}}{\frac{\#\, vectors \quad in \ X}{n} \frac{\#\, vectors \quad in \ Y}{n}} \tag{4}$$

which is proportional by a constant (n) to:

$$\text{sim(X,Y)} = \frac{|XY|}{|X||Y|} \tag{5}$$

where |XY| is the intersection of sets (clusters) X and Y, i.e. the number of vectors that are the same in X and Y; |X| is the number of vectors in X, and |Y| is the number of vectors in Y (see figure 10).

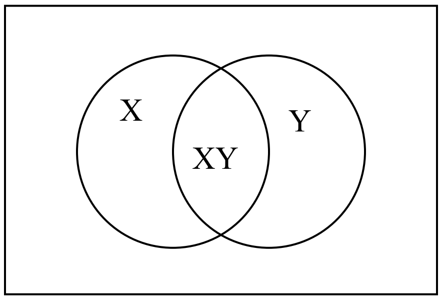


Figure 10: Venn diagram showing the intersection of clusters

This similarity measure is effectively normalized by the total number of vectors in the Universe (n); however, it is not appropriately normalized by how large the clusters are. If we take the hypothetic case where we are considering a cluster with 2 vectors (*Al*) and a cluster of 10 vectors (*Abrahamian*) to be merged to themselves, the measure above will favor the merge of the smaller clusters first:

sim(X, X)

when $|X| = 2$, $\text{sim}(X, X) = 2 / (2\cdot2) = 0.5$ but

when $|X| = 10$, $\text{sim}(X, X) = 10 / (10\cdot10) = 0.1$

For some purposes, it seems reasonable to have a similarity measure whose value is the same when applied to two instances of the same word, no matter what the word is. A measure that has this property is the following:

$$\text{sim2}(X, Y) = |X \wedge Y| / |X \cup Y| \quad (6)$$

where caret (^) indicates intersection of sets and U is set union. Since X ^ X = X U X for all sets, sim2(X, X) = 1 always. This general argument holds for any two clusters, independently of the amount of overlapping.

In our case, we will never be merging a cluster to itself, and we would like to favor the merging of smaller clusters first, to keep the merging process maximally consistent, i.e. adding the least possible number of vectors that are not shared by the two clusters. Therefore, for the purposes of merging pronunciation clusters, we will stick to the first similarity measure of equation 5.

It turns out this is the same as calculating the Hamming or Manhattan distance between bitvectors, where we count the number of positions in which the corresponding values of the two vectors are different; in this case, only the vectors with a distance of 0 (identical vectors) contribute towards the similarity score. Using bitvetors, we could use K-means to get the clusters.

7.4. No clashes

A further restriction we wish to impose on the merging process is that two clusters can only be merged if there is no clash. A clash is determined at the vector level. If two vectors clash, the pair of clusters under consideration (containing clashing vectors) cannot be merged.

A clash is defined as an inconsistency at the phonetic level; given the same letter context (same letters in positions 1 to 3), two vectors clash if their fourth position is not the same, i.e. if the phone is different (see figure 11).

```
If L1L2L3 = L1'L2'L3'
    if ph2 = ph2', continue to compare remaining vectors in the clusters, if
                   there are no more vectors, merge;
    else (ph2 != ph2') there is a clash, consider the next pair of most similar
                   clusters for merging
```

Figure11: definition of clash between clusters

If we were considering for merging the two clusters shown in figure 8[27], we would find that there are only two comparable vectors, the ones that have *#sc* and *sch* as the letter context. So these are the only ones that can block the merging of the pair of clusters (the rest only matter when calculating the similarity measure between clusters), and we proceed to compare their fourth position, the phones aligned to the second letter. See figure 12 for a graphical explanation.

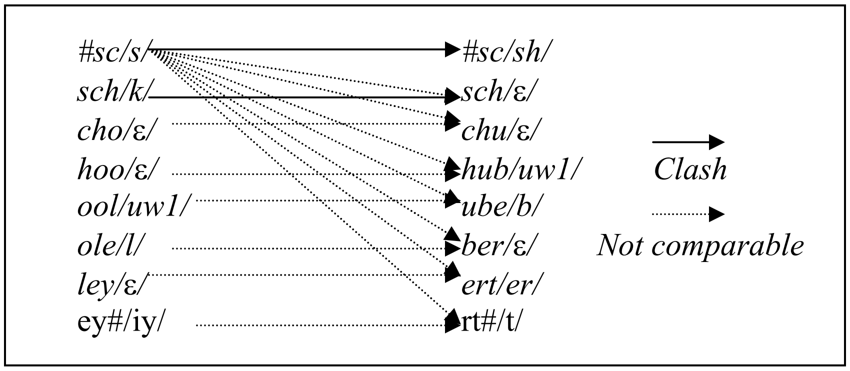


Figure 12: partial graph comparing vectors to check for clashes (every vector is compared with every vector in the other cluster)

The clusters containing the vectors for the names *Schooley* and *Schubert* can not be merged, since the two comparable vectors actually clash (it would have been enough to block the merge if only one pair of comparable vectors clashed).

If there are no comparable vectors within two of clusters, such clusters will, in principle, merge; however, the MI will delay the considering of such clusters for merging until the very end, since they have no overlap.

Our approach assumes that there are no clashes within the same word. We cannot prove it; however, we cannot think of a word that could have a clash other than nonsense words such as *deded / d eh1 d ax* d / or *bedabbed / b eh d ah1 b ax d /*.

[27] This is not very likely since they have no overlapping vectors.

Our current merging criterion does not prevent *aza/z/* and *aze/z/* to end up in the same cluster; however, when comparing the vectors, the current similarity measure does not consider them at all, since they do not have the same letter context. What we would like to do is to say that *aza/z/* and *aze/z/* should contribute to the similarity score with some weight smaller than 1 (identical vectors contribute to the similarity score with weight 1), but when comparing *aza/z/* - *ato/t/* or *aza/z/* - *sch/k/* similarity between clusters should not be increased.

Actually, by forcing the similarity measure to consider only identical vectors, we are not capturing the insight that if all the instances of a letter in a cluster, regardless of its context, are always aligned to the same phone, this mapping is maximally consistent. If we followed this argument, we would have to postulate that vectors like *aza/z/* and *aze/z/* contribute to the similarity score with a weight greater than 1.

Two binary random variables are not enough to capture any similarity other than identity, and so, vectors similar to the ones in X^Y but not in X^Y (e.g. if aze/z/ is in X^Y, and aza/z/ is in Y), will be ignored in the same way as vectors unrelated to those in X^Y are ignored (e.g. *sch/k/*).

Ideally, we would like to have a way of increasing the MI when there are similar vectors, even if they are not identical. This might be achieved by a modified version of Hamming or Manhattan distance, which instead of comparing each position to see if they are identical, positions one and three (L1 and L3) would only have to be from the same class (possibly vowel vs. consonant) to be counted as *equal*. Positions two and four (L2 and P2), would have to be the same, with the constraint that if the values in position 2 are the same, then the values in position 4 have to be the same to be counted as equal. In general, any measure that allows us to measure the distance between a vector and the intersection set X^Y is what we want (this might require fuzzy methods).

7.5. Computational time and efficiency

As mentioned before, clustering is a very computationally expensive task, and even when using a greedy algorithm, the brute force approach described in figure 9 is not feasible. [Brown et al. 1992] proposed a shortcut to calculate the loss of MI after every merge in constant time, which did not involve recalculating the MI between all pairs.

The idea of their shortcut is to keep track of the average MI values that change after two clusters have been merged only, to be able to re-estimate the loss of MI given a new merge (adding and subtracting the intersection points of the two clusters and of the new cluster with the rest of the clusters).

This shortcut is actually overkill for our task. There are other tricks we can use to reduce complexity and speed up the process, which should not result in significant decay of performance.

One thing we could do is take advantage of clashing restriction to reduce computation time by indexing the vectors with all the vectors it clashes with during the first pass of the algorithm (first time we calculate similarity between all possible pairs).

This would involve ORing of huge bitvectors (5·50K long on average); however, even though it would probably speed up the computation by a factor of 50 or a 100, we would at least need a speed up by a factor of a thousand to have a real effect in the complexity and computation time of the algorithm.

Another trick we can use to reduce the complexity of the algorithm by a factor of V^2 is to initially pick a cluster, the one with the lowest entropy say (fewer number of vectors), and then find which one is most similar to it, and try to merge them. If they cannot merge (there is a clash) take the next most similar one, and so on.

The latter is the trick we implemented, since it was very easy and it is our believe that all the initial clusters will merge with some other cluster, so adding some more greediness to the search should not drastically affect performance.

7.6. Models 6 and 7

The final clusters, which have a consistent[28] pronunciation, precisely describe the LTS rules for the words in the cluster. Unfortunately, with 459 clusters, there are only an average of 200 words per cluster. This is clearly not enough to effectively build a CART that would generalize over the 200 training examples per cluster.

So the direct use of *language*[29] features in this case is out of the question, we cannot build a CART for each cluster. We can try to build just a single CART using the *language* features indirectly as done for models 2, 4 and 5. This time, though, we cannot build a LLM for each cluster, since 200 words are not enough to reliably estimate trigram frequencies.

Model 6 effectively builds a single CART with all the names only tagged as belonging to one of the 459 clusters (without having run the Language Identifier on the training data). The results are disappointing; pronunciation accuracy is actually lower than the baseline model (see next section for the detailed results).

Ideally, having clusters with a maximally consistent pronunciation is exactly what we want to build LTS rules. However, adding the features in the CART does not improve pronunciation accuracy. It is possible that there is a better way to use these final clusters, but it remains an open question that will be researched in the near future.

[28] By consistent we mean with no contradictory LTS rules, that is given a letter context, the pronunciation will be the same.
[29] In a way, a cluster can be seen as an abstract language, one which has a maximally consistent pronunciation.

Natural languages are not really maximally consistent regarding the pronunciation, especially when it comes to pronouncing proper names (cf. 1.3), so it is possible that allowing a few clashes to occur during the clustering process will reduce the number of clusters to a more manageable size.

Starting from the 459 clusters, we proceeded to allow clashes gradually, and obtained the curve shown in chart 6.

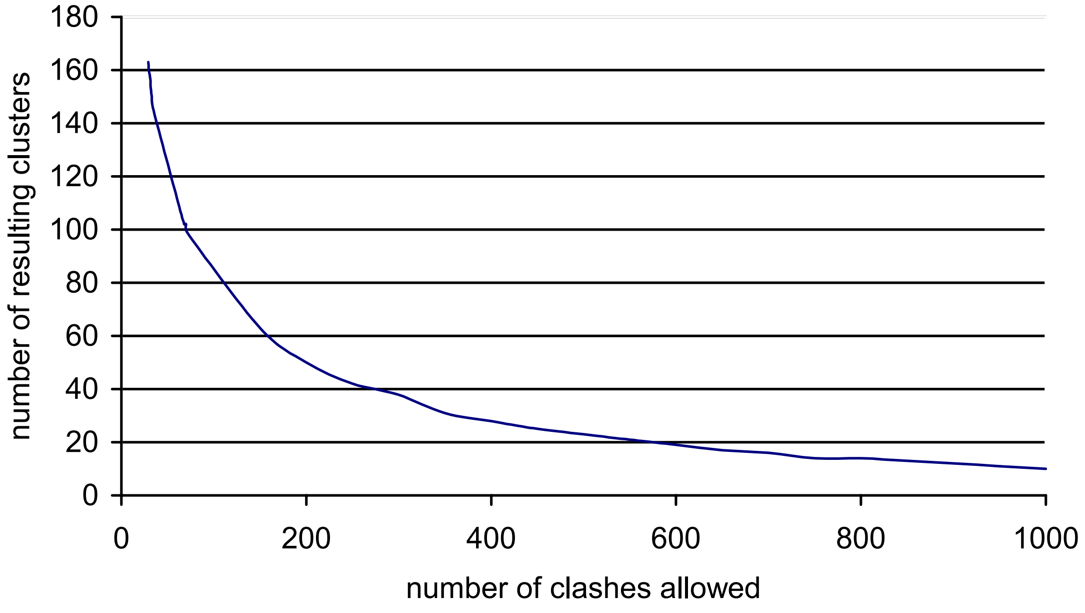


Chart 6: curve showing that allowing clashes reduces the number of resulting clusters

As expected, the number of clusters decreases as we gradually allow more clashes. However, to decrease the number of clusters to a manageable size, many clashes need to be allowed. Only after allowing about 200 clashes, the number of clusters is reduced to 50. It takes 1000 clashes to reduce the number of clusters to 10.

Having allowed several clashes, we are in the position to actually build LLMs from the words in each clusters and add the language probability features in the same way as it is done in models 2, 4 and 5 (cf. 5.1.2).

For model 7, we built several CARTs, varying the number of clusters, to see what was the best tradeoff between reducing the number of clusters and pronunciation accuracy. To our surprise we found no correlation between pronunciation accuracy and number of clusters. We expected to see pronunciation accuracy (gracefully) degrade, as we allowed more clashes to occur; however, the few data points obtained show that such correlation does not exits (see chart 8 in the results section). This probably means that allowing clashes is not the right solution to this problem.

Unfortunately, we did not have time to try other variations of these models, such as including other features to model 6 (such as length), and build the clusters without taking stress into account, which will most likely result in a smaller number of final clusters, since vectors that used to constitute a clash, will now allow two clusters to merge (e.g. *car/ah1/* and *car/ah/*).

Preliminary results for the models that used unsupervised clustering to find the languages are not encouraging. This might be due to several reasons: (i) the similarity measure used is not good enough, (ii) the use of the cluster information is not optimal, or (iii) such pronunciation maximally consistent clusters do not help to automatically improve pronunciation accuracy.

Much research still needs to be done to be able to confirm or contradict reasons (i) and (ii); however, we are quite optimistic that unsupervised pronunciation clusters can be used to improve pronunciation accuracy.

7.7. Results for models 6 and 7

Model 6's pronunciation accuracy is actually almost 4% below the proper name baseline. All variations of model 7 that we tried, i.e. with 99, 72, 50, 39, 25 and 10 final clusters, achieved better accuracy than model 6, but still slightly lower than the baseline (see table 7 below). The reason why model 6 is worse than all the versions we tried of model 7 is probably data sparseness; recall that we only had an average of 200 names per cluster.

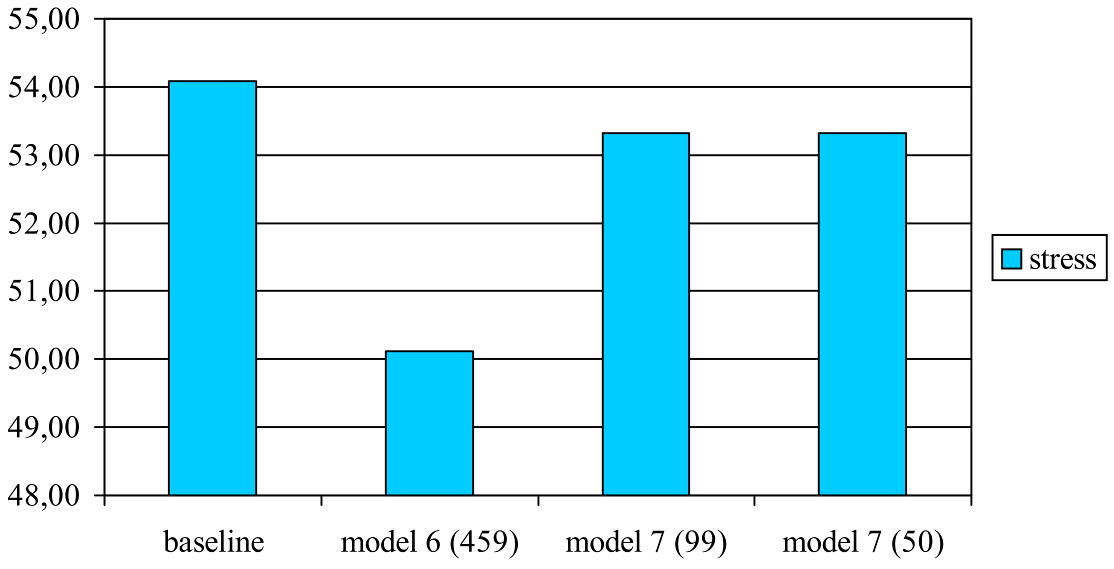


Chart 7: comparing model 6 and two models 7, one with 99 clusters and the other with 50 clusters, against the baseline.

Models	Letters NS	Words NS	Words Stress
baseline	89.02%	60.48%	54.08%
Model 6 (459)	88.02%	57%	50.11%
Model 7 (99)	88.76%	n/a	53.32%
Model 7 (72)	88.58%	n/a	52.39%
Model 7 (50)	88.76%	n/a	53.32%
Model 7 (39)	88.58%	n/a	52.39%
Model 7 (25)	88.28%	n/a	51.21%
Model 7 (10)	88.49%	n/a	52.31%

Table 7: comparing the baseline with model 6 and model 7 with different number of clusters (in parenthesis).

In chart 8, we plotted how pronunciation accuracy varies according to the number of clusters we used to build the LLMs in model 7.

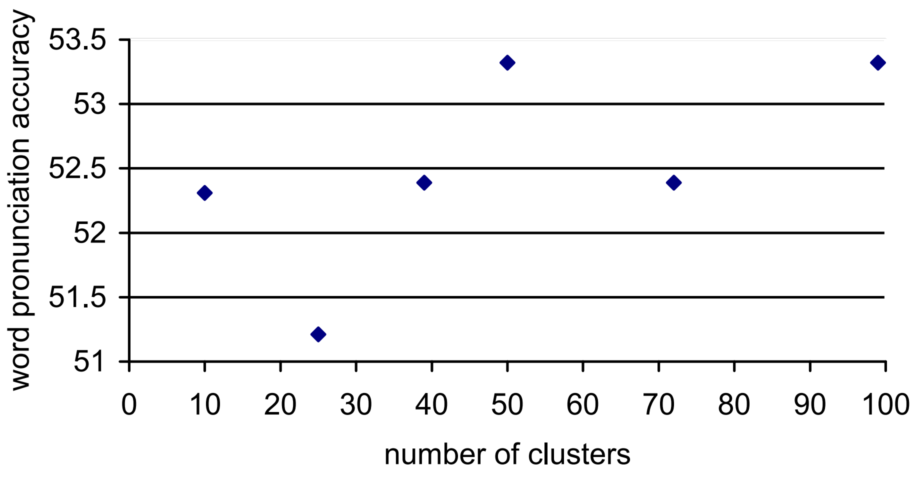


Chart 7: comparing model 6 and two models 7, one with 99 clusters and the other with 50 clusters, against the baseline

Chart 8: plotting pronunciation accuracy as the number of clusters decreases (i.e. as we allow more clashes)

Even though the preliminary results are not promising, we feel that this direction has still many paths that remain unexplored. It is conceivable that improvement in the clustering could be achieved using other similarity measures to find the best pair of clusters to merge; and instead of building CARTs using the cluster information, it would be interesting to try other machine learning techniques that do not suffer from data fragmentation, such as Maximum Entropy models.

Chapter 8
Upper Bound and error analysis

There are several factors that place an upper bound on the performance of our pronunciation models. Even if some work still remains to be done in analyzing the results obtained in this thesis and determining the upper bound on performance, this chapter addresses some of the issues and attempts to give or at least to hint at some answers.

8.1. Noise user studies

Previous to this thesis, there was no evaluation of the accuracy of the pronunciation of the list of 56,000 proper names. To assess the amount of noise in the training data, we carried out user studies.

We synthesized the pronunciation of 87 names from the training data, chosen at random, and asked the users to assess how accurate they were by scoring them from 1 to 3, where 1 is clearly acceptable (as an educated native American English speaker would pronounce it), 2 is possibly acceptable (somebody could say it like that) and 3 is clearly unacceptable (no one would pronounce it that way).

The users were first asked to listen to three sample names to get used to the speech synthesizer, and then they were asked to listen to the 87 names and score each of them as described above[30].

[30] The reader can look at how the user studies were set up by going to the following URL: http://www.cs.cmu.edu/~aria/user-studies/noise-intro.html

Four educated native American speakers (with a strong linguistic background) did the user studies. The results obtained were the following:

345 total number of scores given		
1 (clearly acceptable)	61.74%	(213)
2 (possibly acceptable)	22.90%	(79)
3 (clearly unacceptable)	**15.36%**	(53)

Users scored 84.64% (292) of the pronunciation as acceptable and 15.36% as unacceptable. This means that a little bit more than one pronunciation in every 10 is not acceptable in our training data.

This naturally places an upper bound on the models described in this thesis. To really assess the value of our models to improve pronunciation accuracy on proper names, we would need to train and test them on noise free data. Unfortunately, such a thing does not exist. Nevertheless, it would still be worthwhile running our models on different data and compare the results. This remains as future work.

8.2. Language Identifier accuracy

As mentioned in section 5.1.2, we can estimate the accuracy rate of our Language Identifier, from the hand labeled examples. When classifying the hand labeled names with the 25 languages, we found that the LI assigned the right language as first or second choice 33% of the time. If we only count the times when the LI assigned the right language as first, accuracy is even lower, 19.41%[31].

These numbers are quite low, and it would probably help our experiments to have a more accurate LI. However, this would involve a large amount of manual work into hand-

[31] The reader should keep in mind that humans were only able to label 43% of the training examples.

tuning it, and we are interested in automatically building language classes to see how much we can improve pronunciation accuracy, without requiring neither a year of dedicated work in building a language identifier, nor much linguistic knowledge.

As expected, the accuracy of the Language Identifier increases when classifying the names into 5 language families. In this case classification accuracy goes up to 73%, so it is much more reliable.

It is surprising that even though classification accuracy is much higher when using family LLMs, pronunciation accuracy of models 4 and 5 is not significantly higher than for model 2. In fact, model 3, which makes a hard decision about the language features, and uses them to divide the data into family specific subcorpora, performs much more poorly. This remains a puzzle, and we would like to further investigate it.

Finally, we should look into implementing n-gram LLMs and a backoff classifier to see if we get a more accurate language identifier that way.

8.3. Oracle experiment

In the same spirit of the work done in speech recognition (ROVER [Fiscus 1997]) and in machine translation (multi-engine MT System [Frederking *et al.* 1994]), it is interesting to estimate the accuracy of all our models combined, if we always pick the best model for each name in the test set. Work done in these other two research areas have shown the combination of several models/systems to perform better than any of the models/systems separately.

For our oracle experiment, we chose to combine the baseline model (cf. Chapter 4), a model with language features (model 2, cf. 5.3), a model with both language features and language family features (model 5, cf. 6.3) and a model with *language* classes result of the unsupervised clustering process (model 7, cf. 7.6).

When looking at the overlap of pronunciations, which these 4 models failed to generate identically to the transcriptions in the CMUDICT, we obtain the matrix shown in table 8.

	Baseline model	Model 2	Model 5	Model 7
Baseline model	100%	**36.42%**	**34.64%**	**37.72%**
Model 2	36.42%	100%	**40.18%**	**37.52%**
Model 5	34.64%	40.18%	100%	**37.9%**
Model 7	37.72%	37.52%	37.9%	100%

Table 8: Matrix showing the amount of overlap of failed names between any two models

The reason the amounts of overlap between the different pairs of models is so similar is that the amount of overlap among the baseline and models 2, 5 and 7 is 32.32% (1,664 names) to start with, and that the amount of overlap exclusive to only two any given models is typically rather small (see figure 13).

baseline + model 2: 24 names (0.46%)

baseline + model 5: 28 names (0.54%)

baseline + model 7: 196 names (3.79%)

model 2 and model 5: 157 names (3.03%)

model 2 and model 7: 28 names (0.54%)

model 5 and model 7: 39 names (0.75%)

baseline + model 2 + model 5: 98 names (1.89%)

baseline + model 2 + model 7: 89 names (1.72%)

baseline + model 5 + model 7: 98 names (1.89%)

model 2 + model 5 + model 7: 158 names (3.05%)

baseline + model + 2 + model 5 + model 7: 1,664 names (32.31%)

Figure 13: Number of names that only failed in some model combinations. For example, the number of names that only failed in both the baseline and model 2 is 24 (0.46% of all the names in the test set)

Since we have the number of names that fail in all models, we can easily calculate the pronunciation accuracy that could be achieved if we always picked the best model. Indeed, for any name that not all models failed, there would be at least one model that would generate the *correct* pronunciation, and thus such **oracle pronunciation accuracy would be 67.78%**, which is 12.57% higher than the accuracy achieved by the best model, model 5 (55.21%) (see chart 9).

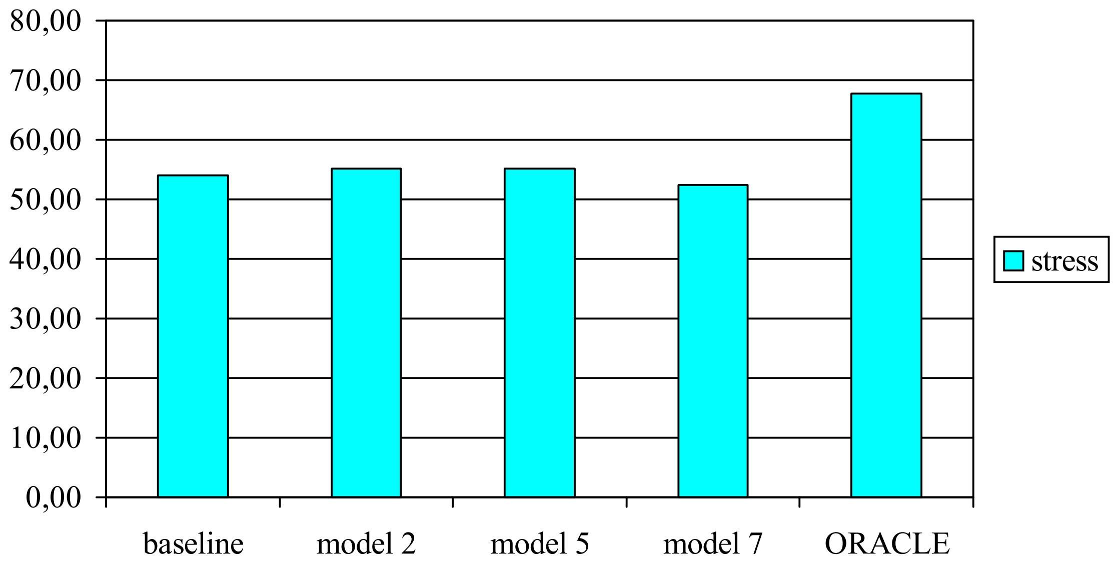


Chart 9: comparing the pronunciation accuracy of 4 models with the oracle model which always picks the best pronunciation among the output of the 4 modes

The result of the oracle experiment gives an **upper-bound** on the overall pronunciation accuracy that could be achieved with our models, since even with an excellent method of combining the 4 models, we cannot expect the accuracy to reach that of the oracle rate (some of the differences between the models are due to fluctuations, not systematic behavior).

8.4. Error analysis

By looking at the first 100 names that failed when using the baseline model, we found that errors could be classified into 4 main classes:

- different vowel (32% of all the errors)

E.g.: abreu (# **ax** b r _epsilon_ uw1 #) *(# **aa** b r _epsilon_ uw1 #)*[32]

- stress misplaced (14% of all errors)

  E.g.: abbate (# **aa1** _epsilon_ b **ey** t _epsilon_ #)

  *(# **aa** _epsilon_ b **ey1** t _epsilon_ #)*

- different vowel + stress misplaced (44% of all the errors)

  E.g.: abbruzzese (# **aa** _epsilon_ b r uw _epsilon_ t-s ey1 z iy #)

  *(# **ae1** _epsilon_ b r uw1 _epsilon_ t-s ey1 z iy #)*

- different consonant (9% of the errors)

  E.g.: alexopoulos (# ae l ih **g-z** aa1 p ax _epsilon_ l ih s #)

  *(# ae1 l eh1 **k-s** aa1 p ax _epsilon_ l ih z #)*

If we look more carefully at these classes, we can further classify the errors in subclasses.

- different vowel :
  - vowel expansions (12% of all the errors)
  - vowel reductions (13%)

    E.g.: abed (# **ax** b **eh1** d #) *(# **aa** b **_epsilon_** d #)*

    where the first error is a vowel expansion and the second error, a vowel reduction.
- stress
  - adding another primary stress (26%): results in an illegal stress pattern

    E.g.: agarwal (# aa1 g **aa** r w aa l #) *(# ae<u>1</u> g **aa1** r w ax l #)*

  - eliminating stress (24%): most of the times results in an unstressed word, i.e. an illegal stress pattern

[32] The first phonetic transcription given is the one in the CMUDICT, and the second phonetic transcription, in italics, is the one generated by our models.

E.g.: adan (# **ey1** d ax n #) *(# **aa** d ax n #)*

If we look at how many errors per word there are in the first 100 names, we see that 59 names had only 1 error; 32 names had 2 errors; 6 names had 3 errors, and 3 names had 4 errors. Therefore, more than 50% of the time, names that fail have only one error.

By looking at the errors that model 2 (cf. 5.3) made, it seems that even though there is a large overlap with the baseline with respect to the names that both fail to generate correctly (see previous section), the actual errors they make are only the same about half of the time.

When looking at the first 100 names for which the model 2 failed, 74 were the same as the ones that failed for the baseline model (19 names that failed in the baseline model did not fail in the model 2, and 7 names that failed in model 2 did not fail in the baseline). However, when comparing with the CMUDICT, only 53% of the errors were the same and the remaining 47% of the errors where different from the ones observed in the baseline.

In total, 26 errors made by the baseline model (17%) were corrected by model 2. An example of this is the name *Arocha*:

arocha (# **aa** r ow1 **k** _epsilon_ ax #)

baseline: *(# **_epsilon_** r ax **_epsilon_** _epsilon_ ax #)*

model 2: *(# **ae1** r aa1 **ch** _epsilon_ ax #)*

which gets the right pronunciation for "-cha" but still gets counted as an error by our evaluation method, due to the noise present in the training data. Note that the first vowel adds a second primary stress, however it is still a much better pronunciation than the one generated by the baseline.

Another example of this can be found in *Alane*:

alane (# **ax** l **ey1** n _epsilon_ #)

baseline: *(# **aa** l **aa1** n _epsilon_ #)*

model 2: *(# **aa** l **ey1** n _epsilon_ #)*

Conversely, 30 errors were introduced by model 2 (19%, included in the count of different errors above). An example of this is the pronunciations of the name *Anania*:

anania (# **aa** n **aa1** n iy ax #)

baseline: *(# **aa** n **ae1** n iy ax #)*

model 2: *(# **ae1** n **aa1** n iy ax #)*

which introduces an error for the first phone. Note that in this case, model 2 adds an error from the baseline but also corrects an error in the third phone.

Even though we had estimated a 15% noise with user studies (see section 8.1), by looking at the data more carefully, we realized that this was probably a very conservative estimate and that there is a lot more noise in the training data.

There were plenty of examples of names with a noisy transcription in the training data that were generated reasonably by our models. At least 14 of the vowel expansion errors, were actually due to errors in the CMUDICT, since the first vowel of these names is a schwa in the training data, for which the baseline model gives a more reasonable pronunciation:

abreu (# **ax** b r _epsilon_ uw1 #) *(# **aa** b r _epsilon_ uw1 #)*

acree (# **ax** k r _epsilon_ iy1 #) *(# **aa** k r _epsilon_ iy1 #)*

adele (# **ax** d eh1 l ey #) *(# **ae1** d iy1 l _epsilon_ #)*

This type of error is clearly due to the original phonetic transcriptions being generated by LTS rules. Even though these transcriptions were originally manually checked, it is only natural (when having to look at 56,000 names) that many errors went by unnoticed.

Other examples of errors in the training data that would have not been considered as errors of our models if we had performed a more subjective evaluation are:

addison (# ae1 _epsilon_ d **ax** s ax n #) *(# ae1 _epsilon_ d **ih** s ax n #)*

allender (# aa **l y** eh1 n d **ey** er #) *(# ae1 **_epsilon_ l** eh n d **_epsilon_** er #)*

Some consonant errors were due to inconsistencies in the alignments. Instead of there being only one way to align "–ll" followed by a vowel in the training data, we found two when looking at the first 100 names that failed in the baseline model:

allard (# ae1 **_epsilon_ l _epsilon_** er d #)
(# ae1 _epsilon_ _epsilon_ _epsilon_ er d #)

alley (# ae1 **l _epsilon_ _epsilon_** iy #)
(# ae1 _epsilon_ _epsilon_ _epsilon_ iy #)

which makes it much harder for a DT to learn the right transcription. Using n-grams to pick the best alignment could probably solve this.

Now that we have a better idea of what sort of errors our models make, we are in a better position to think of future work that would correct precisely the most common errors and thus significantly improve accuracy.

One obvious thing we could do to avoid stress errors, would be to have a word-level constraint of just allowing one primary stress. This would probably work for most of the cases, however, since the training data has been pre-processed in a way that secondary stress becomes primary stress, and thus some names in the training data do have two primary stress, it would not work for all cases.

The greatest source of errors is not stress alone, but stress combined with vowel modification. Since our models treat accented vowels the same as unaccented vowels, it is not clear that these kinds of errors are actually harder to correct than the errors that are due to stress or a vowel modification alone. However, it would probably be useful to try to tackle stress and other kinds of errors separately.

It is tempting to suggest to try training our models without taking stress into account and then trying to apply some heuristics, trained on the same data, to assign stress on a subsequent pass. However, experiments reported in [Pagel *et al.* 1998] actually indicate that pronunciation accuracy is much higher when modeling stress together with the phonemes. This is the reason our models treat accented vowels and unaccented vowels in the same way. Pagel and colleagues found that taking the phonemic context into account, as well as the graphemic context, improves pronunciation word accuracy. Adding the 3 previous phones to the pronunciation model actually allows to better model stress (they give the example of *strategy* vs. *strategic*). Therefore, an area of future work to correct stress errors, would be precisely to add phonemic context to our models.

To sum up, our models are being penalized by both the test set errors as well as the training set errors, and even though there are still several directions of future work that can be undertaken to reduce the test set errors (see Chapter 10), the serious problem of training set errors also needs to be addressed. There is little hope of manually correcting such errors, however it would be possible to use EM-like methods to throw away training data that has low likelihood, as is commonly done in training acoustic models. With such methods, we would get rid of some of the original errors in the data, or at least provide some confidence measure of the goodness of the phonetic transcriptions so that we could obtain a cleaner training set to start with.

Chapter 9
Conclusions and discussion

Language probability information definitely improves pronunciation accuracy of proper names. However, there are still many interesting issues that remain to be investigated regarding automatic ways to benefit from the pronunciation at training time, as well as exploring more data-driven approaches.

Ideally, we should have trained all our LLMs on just names, instead of text corpora, since that is the distribution of our training data. However, we did some experiments with LLMs trained on both text and just proper names for German, French and Spanish, and results have shown that the probability of the two LLM were very close, and it never happened that the LLM trained on text performed worse than the LLM trained on proper names.

Human language identification in isolation (without having any contextual information nor a prior, and without special characters) is very hard. Humans can confidently tag less than 50%. Therefore, a system that tries to mimic human behavior when pronouncing a proper name but that uses an automatic language identifier could actually perform better, and it will certainly be more consistent.

Getting untranscribed multilingual data for majority languages is not a problem, since most such countries have some kind of electronic telephone directory[33], the challenge is to confidently and automatically classify the transcribed names that we have at our disposal into language classes. A fully automatic language identifier, as the one implemented in this thesis, might not the answer, since it has very low accuracy.

[33] This is more problematic for minority languages which do not have a rich and technologically advanced country behind them; example of such languages are most African languages, Indian American languages, etc.

Experiments with a smaller and cleaner set show that our model 2 (cf. 5.3) achieves a significant improvement on pronunciation accuracy, but when trying to scale it up to all the training data with a fully automatic way of identifying the languages, results are not as promising. This led us into data driven classification methods, i.e. unsupervised clustering.

Models 6 and 7 are an attempt to find a way around the language identification problem, by not imposing any linguistic notion on the classification, but rather by letting the words and their pronunciation lead the clustering process. Unfortunately, preliminary results are not very promising.

Nevertheless, since the unsupervised clustering experiments described in this thesis are still preliminary, and there are many alternative paths that we need to further investigate (and more experiments need to be run before we can be confident about our results), we still believe that data-driven approaches will help improve pronunciation accuracy of proper names.

.

Chapter 10
Future work

This small, well-defined problem turned out to be more fructiferous than we expected at the beginning. While running the experiments for this thesis, several other ideas and techniques kept revealing themselves as appropriate, interesting and fun to try. Due to time constraints, we have not run all the experiments we would have liked. However, the experiments described in this thesis serve as a good starting point for all the future directions we are about to present.

10.1. Unsupervised Clustering

Basic data structure

The units we took as basic data structure are more than questionable. It remains to be seen whether they are the right ones or whether we could find a better representation that would allow us to maximize pronunciation differences more efficiently.

One of the questions that remain unanswered is whether to use letter and phone generalizations as features, would increase clustering accuracy (voice and unvoiced phones, CVC patterns, etc.).

Other machine learning techniques: Co-training

It would be interesting to apply co-training techniques [Blum and Mitchell 1998] to see if we can automatically come up with a partition of the classes that better differentiates the pronunciation, introducing as little bias as possible.

We could try clustering written words and pronunciations independently. For the first, using letter n-grams as features, and for the second with both phone n-grams and phones alone as features. Take the overlapping ones as suggested seeds for joint clustering.

For the joint clustering, we could take both the words and the pronunciation of that word and treat them as a compound item represented with three types of features: letter n-grams, phone n-grams and phones alone [3 different feature spaces], which is a modification of the clustering approach we have implemented.

10.2. Probabilistic Model

We would also like to find a probabilistic model that deals with mappings instead of words or pronunciations alone. A first approximation could be a Bayesian approach:

$$P(phones \mid word) = \frac{\sum_L P(L)P(w \mid L)P(phones \mid wL)}{\sum_L P(L)P(w \mid L)}$$

where P(L) are the prior probabilities of all the languages in the corpus [need to gather statistics from training data: Sum over w P(L|w) / number of words], P(w|L) is the language model (trigram-based LLMs), and the P(phones|w L) is the pronunciation model, which is the piece of information missing. Since, we have impartial data, we would try using the EM algorithm to obtain these probabilities iteratively[34].

[34] For the probability distribution of words to phones we could probably use the list of allowables (letter to phones mappings).

10.3. Another model for the second, language family, approach

Another possibility to use language family information in the pronunciation model, would be to build a phone trigram database for each language and use the CART as a regression tree, which would output a density of pronunciations, instead of just one pronunciation.

In the cases where there is a distribution of phones, typically, we run the Viterbi algorithm to find the best path, i.e. the best sequence of phones [Chotimongkol and Black 2000].

Before running the Viterbi algorithm, though, what we would do is to use the phone trigram database as a language model, in the speech recognition sense, to weight the sequences of phones according to the output of the Language Identifier.

For example, if the name has probability 0.03 of being German and 0.02 of being Polish, once the CART has generated a phone distribution, we would look up the phone trigrams relative frequency in the German database, and in the Polish database for the phone trigrams that appear in the forest and weight them accordingly (frequency in German database · P(German)).

10.4. Behind the numbers

Perceived accuracy user studies

It would be useful to assess what is the actual perceived accuracy, as opposed to the objective number we get out of the testing script.

We would like to test perceived accuracy on the names that both the baseline and our models get “wrong”, i.e. did not exactly match the CMUDICT pronunciation in the test set. From those, we would select the ones for which the two models assigned a different pronunciation. We would then proceed to synthesize a randomly selected subset of those (about 20 or 30 names), using each one of the two models.

The users would be presented with 40 names and their pronunciations (in wave files), and would be asked to score each one of the names from 1 (clearly acceptable) to 3 (clearly unacceptable). This would allow us to compare the results from the baseline model with the ones obtained in the noise user studies (cf. 8.1).

End-to-end user studies

The ultimate evaluation of any pronunciation model is to apply it to real text, and see how much it improves overall perceived accuracy. To do this, we would need a way to detect proper names in text, so that proper name could be sent to the specific pronunciation model for proper names, and all other words would be sent to the general pronunciation model. We would need to set up user studies with two synthesized samples of a newspaper fragment, say, one that uses both models (the general plus the proper name specific one), and one that only uses the general pronunciation model.

Users would play both fragments (in random order) and would have to score their general pronunciation accuracy, and after having given their score, they would be asked which system they think pronounces proper names more accurately (letting them play the samples again).

This kind of experiment would allow us to make use of priors, and to do research on automatically setting the priors, as a function of the words in the context.

10.5. Use of (dynamic) priors

Another experiment that would be very interesting to try is adding prior probabilities. For each language, we would have a prior probability that would tell us how likely it is to find a name in that language, independently of the name. If our model were trained from newswires data instead of a name directory, it would be relatively easy to determine such priors. For example, if we had "*Yesterday in Barcelona, the mayor Joan Clos inaugurated the Forum of Cultures…*", then our prior *P(Catalan)* would go up to 0.6 say, and *P(Spanish)* would go up to 0.15, whereas the prior probabilities for all other languages would be smaller.

In newswires text, we would probably need to update our priors dynamically, since it is very common to find proper names from different nationalities in the same piece of news. A first approximation could be to use geographic cues, such as city and country names, as influence separators.

With our training data, however, it is very hard to determine such priors since we do not know exactly from which area in the United States the names come from. If we had all the names for CMU students, faculty and staff, say, then we could look up the country of origin statistics, and set our priors accordingly.

An initial approximation to the prior probabilities for our list of 56K names would be to look at the information we have from the LI:

$$P(L) = \sum_{words} \frac{P(L \mid word)}{\# \, words}$$

10.6. Proper names web site

Following up on the idea that nobody knows how to pronounce their name better than themselves, we would like to set up as an automatic way as possible to collect proper names data from people.

We do not know of any initiative of this sort, with the exception of Spiegel's year and a half crusade of calling American homes asking people how they pronounced their names. His efforts resulted in a very accurate pronunciation dictionary, which belongs to Bellcore, and so the research community does not have access to it.

I would like to put up a web site where people could look up the pronunciation of their names as generated by our several pronunciation models, as well as the statistics of the pronunciation of their name given by previous users. We would ask them to pick which one is the one they use, or if it were none of the ones proposed by the system, they would be asked to enter it on the site.

This would allow us to gather statistics about how proper names are pronounced in the US, as well as a valuable proper name pronunciation database which, after making sure it is correct, could be added to the CMUDICT and thus benefit all the systems that use this pronunciation dictionary.

10.7. Handcrafted LTS rules

Finally, it might be worthwhile to hand tune rules specifically for foreign proper names, and compare time spent in building the rules and accuracy achieved with the unsupervised method presented here. Good candidates to do this experiment would be languages such as Spanish, Chinese or Japanese, which have easy LTS rules.

Chapter 11
References

Allen, J., Hunnicutt, M.S. and Klatt, D. 1987. From text to speech: The MITtalk system. Cambridge Studies in Speech Science and Communication. Cambridge University Press.

Association for Computational Linguistics. 1994. *European Corpus Initiative. Multilingual Corpus I.* Produced by HCRC, University of Edinburgh and ISSCO, University of Geneva.

Black, A. W. 2000. *Speech Synthesis in Festival. A practical course on making computers talk*. Edition 2.0, for Festival Version 1.4.1, Carnegie Mellon University, updated May 8th 2000. http://festvox.org/festtut/notes/festtut_toc.html

Black, A. W., Lenzo, K. and Pagel, V. 1998. *Issues in Building General Letter to Sound Rules*. 3rd ESCA Speech Synthesis Workshop, pp. 77-80, Jenolan Caves, Australia.

Blum, A. and Mitchell, T. 1998. *Combining labeled and unlabeled data with co-training*. In Proceedings of COLT '98.

Bouma, G. 2000. *A finite state and data-oriented method for grapheme to phoneme conversion.* In Proceedings of NAACL-2000, Seattle, WA.

Breiman, L., Friedman, J., Olshen, R., and Stone, C. 1984. *Classification and Regression Trees.* Wadsworth & Brooks, Pacific Grove, CA.

Brown P. F., Pietra, V.J.D., DeSouza, P.V., Lai J.C. and Mercer R.L. 1992. *Class-based n-gram models of natural language*. Computational Linguistics 18 4, 467-479.

Busser, B., Daelemans, W., and van den Bosh, 1999. A. *Machine Learning of word pronunciation: the case against abstraction*. In Proceedings of Eurospeech99, Budapest, 2123-2126.

Byrne, B. et al. 1997. *Pronunciation Modelling*. John Hopkins University Workshop 97, CLSP, Pronunciation Group.

Canvar, W.B., and Trenkle J.M. 1994. *N-Gram-Based Text Categorization*, in Proceedings of 3rd Annual Symposium on Document Analysis and Information Retrieval.

Charniak E. 1993. Statistical Language Learning. MIT Press.

Chotimongkol, A. and Black, A. 2000. *Statistically trained orthographic to sound models for Thai*. Beijing.

Church, K. 1986. *Morphological Decomposition and Stress Assignment for Speech Synthesis*. Association for Computational Linguistics.

Church, K. 2000 (1985). *Stress Assignment in Letter to Sound rules for Speech Synthesis* (Technical Memorandum). AT&T Labs –Research. November 27. This TM corresponds to the ACL 1985 paper.

CMU Speech group. 1998. Carnegie Mellon Pronunciation Dictionary.
http://www.speech.cs.cmu.edu/cgibin/CMUDICT

Coker C.H., Church, K.W. and Liberman, M.Y. 1990. *Morphology and Rhyming: Two Powerful Alternatives to Letter-to-Sound Rules for Speech Synthesis*. In Conference on Speech Synthesis, European Speech Communication Association.

Daelemans, W. and van den Bosch, A. 1997. *Language-Independent Data-Oriented Grapheme-to-Phoneme Conversion*. In Progress in Speech Synthesis. Editors: Van Santen, Sproat, Olive and Hirschberg. Springer.

Damper, R.I. and Luk, R.W.P. 1996. *Stochastic phonographic transduction for English*. In Computer Speech and Language, vol. 10, 133-153.

Dermatas, E. and Kokkinakis, G. 1999. *A language-independent probabilistic model for automatic conversion between graphemic and phonemic transcription of words*. In Proceedings of Eurospeech99, Budapest, 2071-2074.

Deshmukh, N., Ngan, J., Hamaker, J. and Picone, J. 1997. *An advanced system to generate pronunciation of proper noun*. ICASSP 97.

Diettreich, T., Hild, H., and Bakiri, G. 1995. *A comparison of ID3 and backpropagation for English text-to-speech mapping*. Machine Learning, 18:51-80.

Fiscus, J.G. 1997. *A Post-Processing System to Yield Reduced Word Error Rates: Recogniser Output Voting Error Reduction (ROVER)*. In Proceedings of IEEE ASRU Workshop, pp. 347-352, Santa Barbara.

Frederking, R., Nirenburg, S., Farwell, D., Helmreich, S., Hovy, E., Knight, K., Beale, S., Domashnev, C., Attardo, D., Grannes, D. and Brown, R. *Integrating Translations from Multiple Sources within the Pangloss Mark III Machine Translation*. In Proceedings of the First Conference of the Association for Machine Translation in the Americas (AMTA-94). Columbia, Maryland 1994.

Fukada, T., Yoshimura, T., and Sagisaka, Yoshinori. 1998. *Automatic generation of multiple pronunciations based on neural networks and language statistics*. In Proceedings of ESCA workshop on modeling pronunciation variation for automatic speech recognition, 41-46.

Ghani, R., Jones, R. and Mladenic, D. 2001 *Building Minority Language Corpora by Learning to Generate Web Search Queries*. Carnegie Mellon University Center for Automated Learning and Discovery Technical Report CMU-CALD-01-100.
http://www.cs.cmu.edu/~TextLearning/corpusbuilder/

Golding, A. R. and Rosenbloom, P.S. 1996. *A comparison of Anapron with seven other name-pronunciation systems*. Technical Report. Mitsubishi Electric Research Laboratories.

Grefenstette G. 1993. *SEXTANT: extracting semantics from row text: implementation details*. Heuristics: The Journal of knowledge Engineering.

Hain, H. 1999. *Automaton of the training procedures for neural networks performing multi-linual grapheme to phoneme conversion.* In Proceedings of Eurospeech99, Budapest, 2087-2090.

Jannedy, S. and Moebius, B. 1997. *Name pronunciation in German text-to-speech synthesis*. Proceedings of the Fifth Conference on Applied Natural Language Processing, Association for Computational Linguisitics, Washington, DC, 49-56.

Jiang, L., Hon, H. and Huang, X. 1997. *Improvements on a Trainable Letter-to-Sound Converter*. Eurospeech 97, Rhodes, Greece.

Knight, K. and Graehl, J, 1998. *Machine Transliteration*. Computational Linguistics, 24(4).

Lucassen, J. and Mercer, R. 1984. *An information theoretic approach to automatic detemination of phonemic forms*. In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 42.5.1-42.5.4.

Manning, C. and Schuetze, H. 1999. *Foundations of Statistical Natural Language Processing*. MIT.

Mitchell, Tom. 1997. Machine Learning. McGraw Hill.

Pagel, V., Lenzo, K. and Black, A. 1998. *Letter to sound rules for accented lexicon compression*. ICSLP98, vol 5, Syndey, Australia, 2015-2020.

Pereira F. and Tishby N. 1992. *Distributional Similarity, phase transition and hierarchical clustering*. In Working Notes, Fall Symposium series. AAAI, 54-64.

Sejnowski, T. and Rosenberg, 1987. C. *Parallel networks that learn to pronounce English text.* In Complex Systems, vol. 1, 145-168.

Spiegel, M. F. 1985. *Pronouncing Surnames Automatically*. Bell Communications Research. September 10.

Sproat, R., Black, A., Chen, S., Kumar, S., Ostendorf, M. and Richards, C. 1999. *Normalization of Non-standard Words WS '99 Final Report*. John Hopkins University Workshop 99, CLSP, Pronunciation Group.

Tanimoto, T.T. 1958. *An elementary mathematical theory of classification*. IBM Technical report.

Taylor, P., Black, A. and Caley, R. 1998. *The architecture of the Festival Speech Synthesis System*. 3rd ESCA Workshop on Speech Synthesis, pp. 147-151, Jenolan Caves, Australia.

Tomokiyo, T. 2000. *Applying Maximum Entropy to English Grapheme-to-Phoneme Conversion*. LTI, CMU. Project for 11-744, unpublished.

Torkkola, K. 1993. *An efficient way to learn English grapheme-to-phoneme rules automatically*. In Proceedings ICASSP 93, vol. 2, 199-202.

Van den Bosch, A., Weijters, T., and Daelemans, W. 1998. *Modularity in inductive-learned word pronunciation systems.* Proceedings NeMLaP3/CoNNL98, Sydney, 185-194.

Vitale, T. 1991. *An Algorithm for High Accuracy Name pronunciation by Parametric Speech Synthesizer*. Association for Computational Linguistics.

Weintraub, M., Yu-Hung, K., Khudanpur, S., Galles, C., Fosler, E., Saraclar, M. 1996. *Automatic Learning of Word Pronunciation from Data*. John Hopkins University Workshop 96, CLSP, Pronunciation Group.

Yvon, F. 1994. *Self-learning techniques for grapheme-to-phoneme conversion.* Onomastica Research Colloquium, London.

Zipf, G. K. 1929. *Relative frequency as a determinant of phonetic change*. (PhD) Harvard Studies in Classical Philology 40:1-95.

Appendix A: CMUDICT Phoneme Set

This appendix shows the US Phoneme Set used in the CMUDICT. The current phoneme set has 39 phonemes, not counting variations for lexical stress.

Phoneme	Example	Translation
AA	odd	AA D
AE	at	AE T
AH	hut	HH AH T
AO	ought	AO T
AW	cow	K AW
AY	hide	HH AY D
B	be	B IY
CH	cheese	CH IY Z
D	dee	D IY
DH	thee	DH IY
EH	Ed	EH D
ER	hurt	HH ER T
EY	ate	EY T
F	fee	F IY
G	green	G R IY N
HH	he	HH IY
IH	it	IH T
IY	eat	IY T
JH	gee	JH IY
K	key	K IY
L	lee	L IY
M	me	M IY
N	knee	N IY

NG	ping	P IH NG
OW	oat	OW T
OY	toy	T OY
P	pee	P IY
R	read	R IY D
S	sea	S IY
SH	she	SH IY
T	tea	T IY
TH	theta	TH EY T AH
UH	hood	HH UH D
UW	two	T UW
V	vee	V IY
W	we	W IY
Y	yield	Y IY L D
Z	zee	Z IY
ZH	seizure	S IY ZH ER

When phonemes are followed by a 0, this indicates they are unstressed; when followed by a '1', this indicates primary stress.

Appendix B: Training data sample

This appendix shows a sample from the training examples used to build all the models described in this thesis. The training data is a list of 56,000 proper names from directory listings originally developed at Bell Labs (AT&T) by Schultz 20 years ago. The pronunciations were initially automatically generated with LTS rules, and then they were hand corrected by Burnstein.

…

```
("adaire" nil (aa d eh1 r))
("adamcik" nil (aa1 d ax m ch ih k))
("adamczak" nil (aa1 d ax m ch ae k))
("adamczyk" nil (aa1 d ax m ch ih k))
("adame" nil (aa d aa1 m iy))
("adamec" nil (ax d aa1 m ih k))
("adames" nil (ax d ey1 m z))
("adami" nil (aa d aa1 m iy))
("adamik" nil (ax d aa1 m ih k))
("adamina" nil (aa d aa m iy1 n ax))
("adamo" nil (aa d aa1 m ow))
("adamowicz" nil (ax d aa1 m ax v ih ch))
("adams" nil (ae1 d ax m z))
("adamski" nil (ax d ae1 m s k iy1))
("adamson" nil (ae1 d ax m s ax n))
("adar" nil (ax d aa1 r))
("aday" nil (ax d ey1))
("adcock" nil (ax d k aa1 k))
("adcox" nil (ax d k aa1 k s))
("addeo" nil (aa1 d iy ow))
("adderley" nil (ax d er1 l iy))
("addie" nil (ae1 d iy))
("addington" nil (ae1 d ih ng t ax n))
("addis" nil (aa1 d ih s))
```

```
("addleman" nil (ae1 d ax l m ax n))
("adduci" nil (aa d uw1 ch iy))
("addy" nil (ae1 d iy))
("ade" nil (ey1 d))
("adee" nil (ax d iy1))
("adel" nil (ax d eh1 l))
("adela" nil (ax d eh1 l ax))
("adelaide" nil (ae1 d ax l ey1 d))
("adelbert" nil (ax d eh1 l b er t))
("adeline" nil (ae1 d ax l ay1 n))
("adell" nil (ax d eh1 l))
("adelle" nil (ax d eh1 l))
("adelman" nil (ae1 d ax l m ax n))
("adelmann" nil (ae1 d ax l m ax n))
("adelpha" nil (ax d eh1 l f ax))
("adelsberger" nil (ae1 d ih l s b er g er))
("adelson" nil (ae1 d ax l s ax n))
("adelstein" nil (ae1 d ax l s t ay n))
("adena" nil (ae1 d ih n ax))
("ader" nil (ey1 d er))
("aderhold" nil (ae1 d er hh ow l d))
("aderholt" nil (ae1 d er hh ow l t))
("aderman" nil (ae1 d er m ax n))
("ades" nil (ey1 d z))
("adey" nil (ey1 d iy))
("adger" nil (ae1 jh er))
("adin" nil (ax d ih1 n))
…
```

Appendix C: List of proper names allowables

This appendix shows the list of letter-to-phone mappings, irrespective of context, allowed for the hand-seeded alignment algorithm described in section 4.3. We started with the basic 50 US phones (including stressed vowels), and progressively allowed phone-combinations according to the examples in the training data (most of which are commented with an example). The final number of phone combinations is 101.

```
(set! allowables
    '((a _epsilon_ aa aa1
        eh eh1
        ah1
        ae ae1
        ey ey1
        ay ay1
        er er1
        ax ax1
        y-ax y-ah1
        aw aw1
        ao ao1
        ih ih1
        w-ax w-ah1
        ow ow1
        w-ey w-ey1
        iy iy1
            ey-ae1)      ;;(# c a s a l #) (# k aa s ey ae1 l #)
     (b _epsilon_ b p )
     (c _epsilon_ k ch s sh t-s
            s-iy       ;;(# c d e b a c a #) (# s iy d ih b aa1 k ax #)
            g          ;;(# m c i n n e s #) (# m ax g ih1 n ax s #)
            eh1-s)     ;; (# v l c e k #) (# v l eh1 s ih k #)
     (d _epsilon_ d t jh)
```

```
(e _epsilon_ ih ih1
  iy iy1
  er er1
  ax ah1
  eh eh1
  ey ey1
  uw uw1
  ay ay1
  ow ow1
  y-uw y-uw1
  oy oy1
  aa aa1
       ao1     ;;(# e r l e #) (# ao1 r ax l #)
       ae1)    ;;(# m c a n e l l y #) (# m ae1 k ax n ae1 l iy)
(f _epsilon_ f)
(g _epsilon_ g jh zh k f
       g-ax)   ;;(# g f e l l e r #) (# g ax f eh1 l er #)
(h _epsilon_ hh
       hh-ax)  ;;(# h j e l m #) (# hh ax jh eh1 l m #)
(i _epsilon_ iy iy1
  ih ih1
  ax ah1
  ay ay1
  y
  aa aa1
  ae ae1
  w-ax w-ah1
  eh eh1
       ey     ;;(# b a u d o i n #) (# b ow d w aa1 ey n #)
       ao1)   ;;(# d i r i c k s o n #) (# d ao1 r ih k s ax n)
(j _epsilon_ jh y hh zh
       ch)    ;;(# w o j i c k #) (# w oy1 ch ih k #)
(k _epsilon_ k
       k-ax)  ;;(# k j a r #) (# k ax jh aa1 r #)
```

```
(l _epsilon_ l ax-l y)
(m _epsilon_ m ax-m m-ax
  m-ae m-ae1
  m-ih
       m-iy1)    ;;(# m c c a r y #) (# m iy1 k eh r iy #)
(n _epsilon_ n ng n-y
       ax-n)     ;; (# b a r f k n e c h t #)  (# b aa1 r f k ax n eh k t #)
(o _epsilon_ ax ah1
  ao ao1
  ow ow1
  uw uw1
  er er1
  aa aa1
  aw aw1
  oy oy1
  uh uh1
  w
  w-ax w-ah1
       w-aa1     ;;(# b a u d o i n #) (# b ow d w aa1 ey n #)
  aa aa1
  ih ih1
  ae ae1
       iy iy1)   ;;(# p o u l o s #) (# p ax l iy1 s #)
(p _epsilon_ p f
       v)        ;;(# s t e p h e n #) (# s t iy1 v ax n #)
(q _epsilon_ k )
(r _epsilon_ r er1 er )
(s _epsilon_ s sh z zh ch
       ih-z      ;;(# b u r r i s s #) (# b er1 ih s ih z #)
       ih-s      ;;(# c h a r e s t #) (# ch aa r ey1 ih s t #)
       ax-s      ;;(# c s a s z a r #) (# k ax s aa1 sh er #)
       s-sh)     ;;(# n e t t e s h e i m #) (# n eh1 t ih s sh ay m #)
(t _epsilon_ t th sh ch dh d s zh)
(u _epsilon_
  ax ah1
  uw uw1
  er er1
  uh uh1
  y-uw y-uw1
  ax-w ah1-w
  y-er y-er1
  y-ax y-ah1
  w
  ih ih1
  ao ao1
  eh eh1
  y-uh y-uh1
        ay      ;;(# k u j a l a #) (# k ay aa1 l ax #)
        iy1     ;;(# n u s b a u m #) (# n iy1 ih s b aw m #)
        uw-w)  ;;(# r u i z #) (# r uw w iy1 z #)
(v _epsilon_ v f)
(w _epsilon_ w v f)
(x _epsilon_ k-s g-z ng-z k-sh z g-zh z
        hh      ;;(# q u i x o t e #) (# k iy hh ow1 t iy #)
        zh)     ;;(# x i o n g #) (# zh ao1 ng #)
(y _epsilon_
  iy iy1
  ih ih1
  ay ay1
  y
  ax ah1)
(z _epsilon_ z t-s zh s
        ax-zh    ;;(d o b r z y n s k i) (d ow1 b r ax zh ih1 n s k iy)
        ax-z     ;;(# w r z e s i n s k i #)  (# r ax z ih s ih1 n s k iy)
        eh1-z)   ;;(s t r z e l e c k i) (s t r eh1 z ih l eh1 t s k iy)
(# #)))
```