Exploring Grapheme-to-Phoneme Induction with Machine Learning

Terrence Szymanski Kevin Wilson

1 INTRODUCTION

Text-to-speech (TTS) systems have increasingly found use in the modern world. One of the subproblems of TTS is determining the phonetic structure of words, i.e., their pronunciation, from their orthography, i.e., their spelling. This is known as the *grapheme-to-phoneme* (G2P) problem. In all languages this is a nontrivial task, but particularly in English, a language with rich historiolinguistics that has led to an irregular and inconsistent spelling system. A single letter, even appearing in similar contexts, can be pronounced several different ways. For example, see Table 1.

The most common solution to the obstacle of unpredictable pronunciations is using a phonetic dictionary with entries that look like the rows in Table 1. However, neologisms like *Google* in English, *Wikcionario* in Spanish, and *Klimakatastrophe* in German constantly creep into the lexicon. Keeping track of all these new words is impossible, yet native speakers can easily pronounce most neologisms at first sight. This suggests that their oththography carries enough information to determine their pronunciation.

The first solutions to the G2P problem involved hand-writing sets of pronunciation rules for various languages. With the onset of the machine learning (ML) paradigm, this method became extremely outmoded.[4] In the ML version, the basic idea is to develop a set of rules which map strings of graphemes to strings of phonemes based on their orthographic *context*, i.e., the letters surrounding the grapheme. These rules are learned from a phonetic dictionary like [2] which can then be applied to new words outside of the training set.

We focus on two kinds of phonetic dictionaries: *aligned* and *unaligned* as described in Section 2. For the aligned dictionaries, we describe algorithms based on k-Nearest Neighbors (kNN) and multi-class Adaboost [3]. These are detailed in Section 2. For unaligned dictionaries, we present two algorithms. The first is based on *Dynamically Expanding Context*, an algorithm that was developed by Kohonen and Torkkola [6, 10] which uses decision trees (cf. Section 3). The second is an original ensemble algorithm based on boosting (cf. Section 4).

While most of the motivation for our algorithms comes from English (as it is both authors' native language), the G2P is a very generic problem. Thus, we have tested our algorithms on German, Spanish, and English datasets. The results of our tests can be found in Section 5.

Finally, we conclude with some future research directions (Section 6) and some remarks (Section 7).

Throughout this paper, strings of

graphemes will be denoted by Latin characters contained in angle brackets ⟨⟩. Strings of phonemes will be denoted in the International Phonetic Alphabet (IPA) in slashes //. For a reference on the IPA, see [9, 5].

2 ALIGNED-DATA METHODS

Most supervised machine learning algorithms assume a set of data instances and corresponding labels. In the G2P problem, this criterion is satisfied if the graphemes in each word are somehow (manually or automatically) *aligned* with the phonemes in the pronunciation.

The organizers of the 2007 PASCAL Letter-to-Phoneme Conversion Challenge have made available several phoneme-aligned data sets, which were automatically generated from unaligned data sets using the EM algorithm.[8] In this data, null phonemes exist both in the orthographic and the phonological representations, represented by the underscore character. These null phonemes make it possible to model "silent" letters and many-to-one alignments. The example word ⟨Utah⟩ illustrates null alignment:

Latin	IPA
mean	min
memo	mɛmo
meme	mim
mneme	nimi
mercy	mərsi

Table 1: Five different words with similar orthographies but radically different pronunciations. Note that both vowels and consonants and even *number of syllables* are different in each word.

$$\begin{array}{rcccccc} g = \langle & \_ & \text{u} & \text{t} & \text{a} & \text{h} & \rangle \\ p = / & \text{j} & \text{u} & \text{t} & \text{ɔ} & \_ & / \end{array}$$

In this example, /j/ is pronounced at the beginning of the word, but is not written in the orthographic form. Conversely, the final ⟨h⟩ is written, but not pronounced.

2.1 Data Representation

In this domain, each data point is a grapheme: each word in the dictionary contributes multiple data points. Each data point is represented as a feature vector, where each feature represents the grapheme and its local context (i.e. the graphemes in the same word up to l positions to the left and r positions to the right). Hash marks are used to indicate any position that exceeds the dimensions of the word.

$$x_i = [g_i \; g_{i-l} \; \cdots \; g_{i-1} \; g_{i+1} \; \cdots \; g_{i+r}]$$

For example, given the parameters $l = 2$ and $r = 2$, and the previous ⟨Utah⟩ data, the feature vector representation is:

$$\begin{array}{lllllll} x_1 = [\; \_ & \# & \# & \text{u} & \text{t} \;] & & y_1 = /\text{j}/ \\ x_2 = [\; \text{u} & \# & \_ & \text{t} & \text{a} \;] & & y_2 = /\text{u}/ \\ x_3 = [\; \text{t} & \_ & \text{u} & \text{a} & \text{h} \;] & & y_3 = /\text{t}/ \\ x_4 = [\; \text{a} & \text{u} & \text{t} & \text{h} & \_ \;] & & y_4 = /\text{a}/ \\ x_5 = [\; \text{h} & \text{t} & \text{a} & \# & \# \;] & & y_5 = /\_/ \end{array}$$

In the following sections, we discuss the application of two popular machine-learning algorithms, kNN and AdaBoost, to aligned data represented in this way.

2.2 kNN Classifier

A simple classifier that ends up performing rather well is the k-Nearest-Neighbors (kNN) classifier. To classify a data point x using kNN, its initial neighborhood N is selected as the set of all data points in the training data whose first feature value is the same (i.e. all other points representing the same grapheme).

$$N(x) = \{x_i : x_i^{(1)} = x^{(1)}\}$$

Next, we calculate the distance from x to every point x' in its neighborhood using a simple features-not-in-common metric for distance d:

$$d(x, x') = \sum_{j=1}^{d} 1_{\{x^{(j)} \neq x'^{(j)}\}}$$

The kNN set contains the k points in N having the smallest distance d from x. The predicted output label for point x is the most frequent label appearing in the kNN set. Since many points may be equidistant from the test point (due to the simple distance metric), we decided to allow the size of the kNN set to occasionally be greater than k in such cases, but never less than k. (In practice, the $d = 0$ or $d \leq 1$ neighborhoods often contain more than k members.)

To mimic the actual classification task, null graphemes in the test input were discarded prior to classification. A more improved classifier might include a step to predict when to insert null graphemes, but this one does not. The results of the kNN classifier are discussed in Section 5.3.

2.3 AdaBoost.M2

While trying to develop the unaligned ensemble classifier described in Section 4.2, we decided to investigate a well-known ensemble classifier, AdaBoost, utilizing a similar type of base classifier that we developed for the unaligned ensemble. Since the AdaBoost algorithm only applies to binary-class data, we implemented the multi-class AdaBoost.M2 algorithm developed by Yoav Freund and Robert Schapire.[3]

AdaBoost.M2 follows the general procedure of AdaBoost, but differs in interesting ways in order to incorporate multi-class data. Let X be the set of all data points (graphemes in context) and Y be the set of all outputs (phonemes). The weak learner selects a hypothesis $h_t : X \times Y \to [0, 1]$, which assigns a "plausibility" to each input-output pair.[3]

Specifically, if the training data consists of m examples $((x_1, y_1) \ldots (x_m, y_m))$, then define the set of all training-data indices and output labels which differ from the observed output label for that index:

$$B = \{(i, y) \in [1..m] \times Y : y \neq y_i\}$$

Also, define a distribution over B, with weights initially distributed equally:

$$D_1(i, y) = 1/|B| \, \text{for}(i, y) \in B$$

The boosting algorithm proceeds (roughly in the same way as the standard AdaBoost) through T iterations. The weak learner chooses h_t at each iteration that minimizes the the "pseudo-error" ϵ_t, defined as:

$$\epsilon_t = \frac{1}{2} \sum_{(i,y) \in B} D_t(i, y)(1 - h_t(x_i, y_i) + h_t(x_i, y))$$

At each iteration, D_{t+1} is updated accordingly:

$$D_{t+1}(i, y) = \frac{D_t(i, y)}{Z_t} \beta_t^{(1/2)(1 + h_t(x_i, y_i) - h_t(x_i, y))}$$

$\beta_t = \epsilon_t / (1 - \epsilon_t)$ and Z_t is a normalization constant. The final classifier is:

$$h_{M2}(x) = \underset{y \in Y}{\text{argmax}} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t} \right) h_t(x, y)$$

The performance of the AdaBoost.M2 classifier depends on choice of the weak classifiers. We experimented with several variations of a simple multi-class decision stump; each assigns its confidence somewhat differently, and this affects the performance of the classifier.

Below are four of the classifier types we tried. Each base classifier specifies j, an index of the grapheme to match, g, the value of the grapheme to match, and y_0, the value

of the correct output phoneme. In this way, each weak classifier matches only those data points whose j^{th} member is g, and favors classifying those points as y_0. The values for j, g, and y_0 are extracted from the training data to generate the set of all weak classifiers from which h_t is drawn.
Weak Classifier 1:

$$h(x,y) = \begin{cases} 1 & \text{if } x^{(j)} = g \text{ and } y = y_0 \\ 0 & \text{otherwise} \end{cases}$$

Classifier 1 assigns all of its confidence to a single positive match output class. If the classifier does not match, then it makes no predictions at all.

Weak Classifier 2:

$$h(x,y) = \begin{cases} 1 & \text{if } x^{(j)} = g \text{ and } y = y_0 \\ 0 & \text{if } x^{(j)} = g \text{ and } y \neq y_0 \\ 1/\text{k} & \text{otherwise} \end{cases}$$

Classifier 2 is similar to Classifier 1, except that it assigns uniform nonzero confidence to all classes if the classifier does not match.

Weak Classifier 3:

$$h(x,y) = \begin{cases} 1 & \text{if } x^{(j)} = g \text{ and } y = y_0 \\ 0 & \text{if } x^{(j)} = g \text{ and } y \neq y_0 \\ 1 & \text{if } x^{(j)} \neq g \text{ and } y = y_l \\ 0 & \text{if } x^{(j)} \neq g \text{ and } y \neq y_l \end{cases}$$

Classifier 3 is similar to Classifier 1, except that it assigns a very high confidence to one arbitrarily-chosen output label, y_l, in the event that the classifier does not match the data point. We generate one such classifier for each value of y_l. It is not clear why, but this weak classifier outperforms all the others (see Table 5).
Weak Classifier 4:

$$h(x,y) = \begin{cases} \frac{\#\{(x_i,y_i): x_i^{(j)} = g, y_i = y\}}{\#\{(x_i,y_i): x_i^{(j)} = g\}} & \text{if } x^{(j)} = g \\ \frac{\#\{(x_i,y_i): x_i^{(j)} \neq g, y_i = y\}}{\#\{(x_i,y_i): x_i^{(j)} \neq g\}} & \text{if } x^{(j)} \neq g \end{cases}$$

Classifier 4 makes very weak predictions, based on the distribution of the data. If the classifier matches the data point, then the output is the relative frequency with which the label y appears in all training points also matching the classifier. If the classifier does not match, then the output is similar, but for all training points that also do not match.

The choice of which classifier set to use can impact performance by up to 20% (see Section 5.4).

3 UNALIGNED DATA: DECISION TREES

This section details our first algorithm based on unaligned phonetic dictionaries. As opposed to the aligned dictionaries described in Section 2, entries in an unaligned dictionary take the form

utah   jutɔ

Neither the fact that the ⟨h⟩ goes to the silent phoneme nor the fact that the ⟨u⟩ produces two phonemes is indicated. This makes the problem of unaligned G2P intrinsically harder than aligned G2P. On the other hand, it is very costly to produce accurate aligned dictionaries (though ML techniques have been applied to this problem as well [8]). Thus, there is a need for an unaligned algorithm.

The major hurdle to implementing an unaligned algorithm is the blow-up associated with adding context. One naïve approach to compiling a list of potential rules would be to write down every grapheme/phoneme pair and every possible context of length d or less and simply trying all of these rules. Unfortunately, this produces $np(1+n+n^2+\cdots+n^d)$ potential rules, where n is the number of graphemes and p is the number of phonemes. The CMU Pronouncing Dictionary [2] lists $n = 26$ and $p = 39$, and if we take $d = 5$, this is already $12529623834 \approx 10^{10}$.

3.1 Motivation

When English speakers see a letter, they know what it sounds like in most contexts. When confronted with ⟨z⟩ they think of ⟨zap⟩ and a ⟨t⟩ brings ⟨tap⟩ to mind. But English speakers also know many contextualized G2P rules. For example, a ⟨c⟩ might make them think of the /k/ sound in ⟨cap⟩, but then know that if ⟨c⟩ is followed by ⟨e⟩ or ⟨i⟩ that it is probably pronounced /s/ as in ⟨cerrated⟩.

This behavior can be modeled as a decision tree. Specifically, we may consider the nodes of the tree to be strings of phonemes and the edges labelled by graphemes. By traversing an edge, we are "adding context." Consider the tree in Figure 1. Then imagine trying to classify the ⟨a⟩ in ⟨gait⟩. Context expands to the right, then the left, and continues back-and-forth. Then our classifier would traverse the ⟨a⟩, ⟨i⟩ branch, and then the ⟨g⟩ branch. As it cannot traverse any farther, it will output the phoneme /eɪ/. Compare then what happens with ⟨cart⟩ and ⟨care⟩. In both cases, the classifier will traverse the tree down the ⟨a⟩, ⟨r⟩, and ⟨c⟩ branches. In the case of ⟨cart⟩, it would continue to traverse the ⟨t⟩ branch and would then output the phoneme /t/. In the case of ⟨care⟩, it cannot traverse any farther, and so will output /eɪ/.

Our first method is based on Dynamically Expanding Context (DEC), first introduced by Kohonen [6] and subsequently incorporated into the G2P literature by Torkkola [10]. The idea behind DEC is to induce all context-free rules that appear throughout an aligned dictionary. Many of these

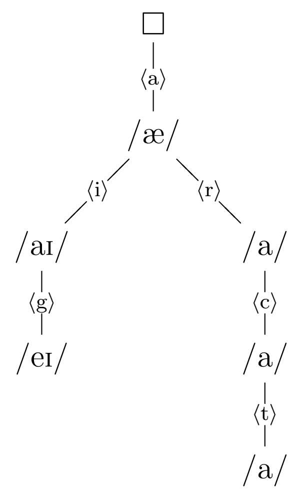

Figure 1: A toy example of a DEC decision tree. Note that the □ node is just a placeholder: all traversals start here.

rules will not be unique. For example

$$t \mapsto \begin{cases} /\mathrm{t}/ & \text{atTack} \\ // & \text{aTtack} \\ /\mathrm{t}/ & \text{Tip} \\ /\mathrm{ʃ}/ & \text{facTion} \\ /\mathrm{ð}/ & \text{The} \end{cases}$$

Once these rules have been estabilished, enough context is added to make the rules unique. For example, on this subset of five words, DEC would produce the following rules:

$$\begin{aligned} t\mathrm{t} &\mapsto /\mathrm{t}/ \\ t\mathrm{a} &\mapsto // \\ \#t\mathrm{i} &\mapsto /\mathrm{t}/ \\ \mathrm{c}t\mathrm{i} &\mapsto /\mathrm{ʃ}/ \\ t\mathrm{h} &\mapsto /\mathrm{ð}/ \end{aligned}$$

where # represents a word boundary.

If g_i is the ith grapheme in a word g, we call the *contextualization* of g_i in w $C(i, w) = g_{i+1}g_{i-1}\cdots$ (cf. Table 2).

Degrees of Context	Expansion
0	t
1	ti
2	cti
3	ctio
4	actio
5	action
C(3,w)	icoanf##

Table 2: Contextual expansion of the ⟨t⟩= g_3 in g =⟨faction⟩.

3.2 An Unaligned Algorithm

The original DEC algorithm assumes an aligned dictionary; we detail an unaligned version. We greedily grow a decision tree like Figure 1 adding one rule at a time, until the improvement from adding a single rule is sufficiently small for l consecutive rounds. Let $\mathcal{C}$ be our classifier, $E = E_{\mathcal{C}}$ be a fixed error metric based on $\mathcal{C}$, s and t fixed constants, D our dictionary, and $|w|$ the length of the word w:

```
Init Set $\mathcal{E}^i = \infty$ for all $i \leq 0$.
Loop In round $i$
  • Find words $w^1, \ldots, w^t \in D$ for which $E(w^j)$ is as large as possible. Let $p^1, \ldots, p^t$ be their phonetic representations.
  For every $j \in [1..t]$, $k \in [0..|w^j|)$, $m \in [0..|p^j|)$, $a \in [0..s)$
      – Traverse along the path defined by $C(k, w^j)$ in $\mathcal{C}$ until a leaf is reached.
      – Add a new node with label $p^j_m \cdots p^j_{m+a}$ connected to this leaf by an edge labelled $g^j_k$.
      – Calculate error
            $$e_{jkma} = \sum_{q=1}^{t} E(w^q)$$
      – Remove the node just added from $\mathcal{C}$.
  • Choose the $j, k, m, a$ that minimizes $e_{jkmaC}$ and add the triple $(w^j_k, p^j_m \cdots p^j_{m+a}, C(k, w^j))$ to $\mathcal{C}$ as in the loop
  • Set
            $$\mathcal{E}^i = \sum_{w \in D} E(w)$$
Repeat until $\max_{i-l \leq j \leq i} \mathcal{E}^j$ is sufficiently small.
```

There are several parameters in this algorithm which serve to make it computationally feasible. First, rules are added to $\mathcal{C}$ based on how they improve the error on some small subset of the entire dictionary. In practice, we used $t = 5$ whereas $|D| \geq 4000$. Because of this, additional rules sometimes *increased* $\mathcal{E}^i$. Thus, we need not only a stopping tolerance, but also a stopping range. This is l, which for good results we took to be about 25.

Note also that s describes the maximum number of phonemes a single grapheme can map to. For our test languages, we took $s = 2$ (as in the ⟨U⟩ in the ⟨Utah⟩ example), though in other languages (Chinese, for example), this s could grow to 4 or 5. For results, see Section 5.

4 ENSEMBLE METHODS

4.1 Motivation

As mentioned, one hurdle to overcome when learning grapheme-to-phoneme rules is that the number of rules grows exponentially with the amount of context which is included in the rule. For instance, consider the class of rules of the form below, which includes l graphemes of context to the left of the target grapheme and r graphemes to its right (in this notation, g_0 is the target grapheme, and other g subscripts indicate a grapheme's position relative to the target).:

$$\{\langle g_0\rangle \rightarrow /\text{p}/ : \langle g_{-l}\cdots g_{-1}\rangle \_ \langle g_{+1}\cdots g_{+r}\rangle\ \}$$

In a situation where there are n graphemes and p phonemes, then there are pn^{l+r+1} potential G2P rules. However, using simple weak classifiers which only consider a single contextual grapheme can greatly reduce the size of the set of rules.

For instance, if we are trying to learn a rule to pronounce ⟨t⟩ as /ʃ/ before ⟨ion⟩ (as in words like ⟨caption⟩), then the intuition is that the effect of a G2P rule such as:

$$\{\langle \text{t}\rangle \rightarrow /\text{ʃ}/ : \_ \langle \text{ion}\rangle\ \}$$

could be approximated by the interaction of three simpler rules of the form:

$$\begin{array}{c}\{\langle \text{t}\rangle \rightarrow /\text{ʃ}/ : g_{+1} = \langle \text{i}\rangle\ \}\\ \{\langle \text{t}\rangle \rightarrow /\text{ʃ}/ : g_{+2} = \langle \text{o}\rangle\ \}\\ \{\langle \text{t}\rangle \rightarrow /\text{ʃ}/ : g_{+3} = \langle \text{n}\rangle\ \}\end{array}$$

In this way, each rule / classifier only includes the value of a single grapheme and its position relative to the target grapheme (e.g. 3 graphemes to the right). In a given language, there are $pn^2(l+r)$ such rules; the number of rules is linear in l and r.

Our original goal was to design an ensemble learning algorithm for unaligned data sets, and this is described in the following section. However, the AdaBoost.M2 classifier described in Section 2.3 is based on the same principles. The success of the AdaBoost classifier indicates that ensemble-based methods can work for this problem, although the current unaligned system is still unable to handle large data sets.

4.2 Unaligned Ensembles

Unaligned data are difficult to handle using off-the-shelf learning algorithms, since there is no straightforward correspondence between data points and output labels. Traditional performance evaluation measures (such as error rate) cannot be used, since the correct output labels are not known.

Therefore, some word-level metric of error must be used as the objective function which we want to minimize. Below is a description of an unweighted ensemble classifier for unlabeled data. At each iteration, the learner greedily searches for the weak classifier that minimizes the Mean Phoneme Error Rate (MER, discussed in section 5.1), which is related to the edit distance between the true and predicted pronunciations..

Assume a set of base classifiers H, and as input to the learning algorithm a set of word-pronunciation pairs $((w_1, p_1)\ldots(w_m, p_m))$.

```
Initialize $H$ = the set of all base learners
Initialize $D_i = 1/n$ for all $i$
Do for t = 1, 2, ... T
    1. $h_t \leftarrow \text{WeakLearn}(H, D)$
    2. $r_t = \sum_{i=1}^{n} D_i \cdot \text{mer}(D, w, p)$
    3. $\alpha_t = \frac{1}{2} \cdot \log\left(\frac{(1-r_t)}{r_t}\right)$
```

The final classifier is:

$$F(x) = \operatorname*{argmax}_{y\in Y} \sum_{t=1}^{T} \alpha_t f_t(x)$$

H includes a classifier for each combination of data point (grapheme) and output (phoneme) within a word. Without alignments, we assume that any grapheme in the word could be the origin of any grapheme in the output. The weak learner finds h_t from H that minimizes the error r_t defined above.

This algorithm works, but can become extremely slow as m and H increase. Calculating MER requires calculating the edit distance between each true word and each predicted word. If there are m words of average length l, then this is an $O(ml^2)$ procedure. Since l is typically small, this is not a terrible problem, but it certainly slower than other types of error metrics that only involve counting.

The set H is perhaps the biggest problem. In the aligned problem, each grapheme will only be observed to align with some subset of the total phoneme inventory. However, in the unaligned case, base classifiers are extracted based on the co-occurrence within a word of a grapheme and a phoneme. Inevitably, every grapheme will eventually be seen to co-occur with every phoneme in a large enough dictionary.

Alignment is the obvious solution for cutting down the size of H, but there could be alternatives, such as iteratively recalculating H from a subset of the data and training on that data. This remains for future work. Evalutation of this algorithm on small data sets with reduced phoneme and grapheme inventories shows that it works in principle.

5 RESULTS

In this section we describe the data sets and performance measures used to evaluate the classifiers, and present the results of the G2P classifiers described in this paper.

5.1 Evaluation

One possible evaluation metric for the aligned classifiers would be to simply measure the error rate on phonemes; however, this metric cannot be applied to unaligned data. Therefore we use a more general evaluation metric inspired by Word Error Rate (WER), which is commonly used for measuring the performance of speech recognition systems.[7] This measure counts the number of operations needed to transform the predicted form to the actual form. We define the phoneme error rate (PER) as:

$$\frac{\text{Insertions} + \text{Substitutions} + \text{Deletions}}{\text{Total Phonemes in Correct Transcription}}$$

The mean phoneme error rate (MER) is simply the mean of the PER for all words in the data set. In the tables, we present 1-MER as an analog to accuracy: high values represent good performance, low values represent poor performance.

5.2 Data Sets

In this paper we use three sources of data. The English dictionary we use is the CMU Pronouncing Dictionary, containing 112,102 word-pronunciation pairs.[2] We also use a CELEX dictionary of German, with 49,421 entries and a CELEX dictionary of Spanish with 31,391 entries. For the aligned experiments, we used automatically-aligned versions of these three dictionaries made available by the organizers of the PASCAL Letter-to-Phoneme Challenge.[8]

The CMU dictionary is much larger than the other two dictionaries, and it includes rare words and proper names, as well as some abbreviations and acronyms. The CELEX dictionaries, by comparison, contain more "core" words of the languages. This should be kept in mind when evaluating the performance on these different dictionaries.

5.3 kNN Results

There are some parameters which need to be set for this algorithm, and we do not thoroughly investigate how to set those parameters here. However, Table 3 gives some partial results that indicate that varying the context parameters does not significantly affect the result. Neither MER nor the number of words perfectly predicted, changes much either by varying the total amount of context (i.e. rows 1 and 2 compared to row 3) or by varying the direction of context (row 1 compared to row 2).

Table 4 shows the average results and standard deviation of ten-fold cross validation, using $l = 2$, $r = 2$, and $k = 5$, on the three data sets.

The different accuracy levels achieved by the classifier hints at the relative complexity of English, German, and Spanish orthography; English orthography is very irregular compared to the straightforward orthography of Spanish. Still, it is important to keep in mind that the better results on the Spanish and German data sets may be related to the type of words contained in the CELEX dictionaries.

5.4 AdaBoost.M2 Results

Due to time constraints, the AdaBoost.M2 classifier was not evaluated on the full data sets available to us. Instead, the classifier was tested on one fold of the Spanish data set, using the other folds as training data. (The small variance observed in the kNN cross-validated results suggests that we can consider single-fold results to be representative, but of course if we had more time we would like to cross-validate these results as well.) The algorithm was run for a maximum of 20 iterations while classifying each grapheme.

Table 5 shows the results, which illustrate how the definition of the set of weak classifiers described in Section 2.3 affects the overall performance. The performance of classifiers 1, 3, and 4 is roughly comparable, but classifier 2 (which doesn't look that different on paper) was significantly poorer. Also, note that if the set of weak classifiers is taken to be the union of two of the previously defined sets (1 and 4), the performance is better than either of the sets individually.

The highest accuracy achieved by the AdaBoost.M2 classifier (.978) is lower than the accuracy of the kNN classifier on the same data (.986). However, it is possible that better results can be achieved with continued engineering of the weak classifiers.

5.5 DEC Results

The first few runs of the DEC classifier were slightly disasterous. This came from the fact that eventually the classifier got stuck with a particular set of worst words and consequently hit a local minimum of the error. The result was that it would keep adding the same rule to the classifier ad infinitum. This is when we added the l parameter to the stopping condition and also introduced some randomness to the algorithm. Specifically, with some small probability instead of choosing the t words with highest error, the algorithm will choose a *random* set of t words. Tended to force the algorithm to climb out of its local minimum and start back downward.

The results for this modified version of

l	r	1-MER	% Perfect
2	2	.850	43.3
1	3	.852	42.8
3	5	.847	43.7

Table 3: Results of running the kNN classifier with different values of l and r on fold 0 (training on folds 1-9) of the English CMU aligned data.

Language	1-MER	% Perfect
English	.845 (.0014)	43.0 (.37)
German	.941 (.0015)	66.4 (.58)
Spanish	.986 (.0005)	92.1 (.33)

Table 4: Average cross-validated results of the kNN classifier on English, German, and Spanish aligned dictionaries. Standard deviations are given in parentheses. The fourth column shows the number of words in each dictionary.

DEC are contained in Tables 6 and 7. For the values $s = 1$ and $s = 2$ (where s is the maximum number of phonemes a single grapheme can map to). We note that for Spanish, allowing $s = 2$ actually *increased* our accuracy, but both measures of accuracy *decreased* for English and German. From looking at training output, we believe that this is caused by getting trapped in local minima and having to escape with extremely specific rules. This is especially evident in the German data where the percent of perfect words is 83% for the training data set as opposed to 32.5% for the test data set. One way to improve this performance would be to increase the values of n and decrease the value of l. Increasing the value of n has the unfortunate effect of causing both our decision trees to become more complex, but hopefully decreasing l would offset this. Tuning these parameters is the subject of future work.

We also note that due to its high regularity, Spanish was unaffected by the problem of overfitting as there are always very few plausible rules a grapheme could map to, even when viewed from an ML perspective. What is most surprising is that the DEC classifier actually beat the aligned classifiers in Spanish.

Classifier:	1-MER	% Perfect
1	.933	61.3
2	.786	13.4
3	.978	86.5
4	.941	67.1
1 + 3	.956	75.5

Table 5: Performance of the AdaBoost.M5 classifier using different sets of base classifiers. Evaluation is on fold 0 of the Spanish CELEX data, using folds 1-9 for training.

Language	1-MER	% Perfect
English	.729 (.0060)	15.7 (.004)
German	.902 (.0138)	51.2 (2.70)
Spanish	.986 (.0003)	92.1 (.260)

Table 6: Average cross-validated results of the DEC classifier in the three languages. Standard deviations are in parantheses. For this table $s = 1$.

Language	1-MER	% Perfect
English	.695 (.0308)	15.6 (1.4)
German	.839 (.0473)	32.5 (3.9)
Spanish	.994 (.0006)	97.0 (.36)

Table 7: Same as above, $s = 2$.

5.6 Unaligned Ensemble Results

The unaligned ensemble classifier is very slow, since it requires calculating the full edit-distance error rate for each classifier at each iteration of the algorithm, and it is not possible to classify each grapheme separately (as it is for the aligned AdaBoost classifier). Therefore, we have no large-scale data results for this classifier.

On a small-scale data set, consisting of 80 English words using a inventory of 5 graphemes and 9 phonemes, the classifier achieved 1-MER = .885, with 66.7% of words perfectly predicted. It can be assumed that these numbers would drop on larger and more complex data sets and, as mentioned, the processing time increases dramatically.

6 FUTURE WORK

There is much work to be done on the unaligned approach to ensemble classification. Determining a way to link the error to classifier confidence is a priority, as is finding an efficient way of selecting new classifiers to add to the ensemble.

With the DEC algorithm, the number of parameters is unwieldy. Simplifying these to make them more intuitive should be a priority. Moreover, the decision tree itself can become rather unwieldy and contains much redundancy. For example, there is a general rule in English that a vowel which is followed by a consonant and an ⟨e⟩ is lengthened. This is captured in the decision tree by about 200 rules, as to even see this construction, the tree must also see the grapheme to the *left* of the vowel in question. One way to improve this would be to introduce *wildcards* to the edge graphemes. This would allow pruning to occur as many rules could be collapsed into one.

Work on the AdaBoost.M2 classifier could progress by inventing new sets of base classifiers, or combining old sets, to improve performance. Also, much as in initial DEC runs, the same classifier is often added for several iterations with no immediate improvement in training error. Approaching this problem is an important step to improve this algorithm.

An interesting experiment with the kNN classifier would be to examine in detail how much context is necessary and sufficient. Our small results suggest that $l = 2$, $r = 2$ is adequate, but could they be reduced? Also, various distance metrics could improve performance–perhaps one that weights near context more heavily than distant context. This would also alleviate the problem of large $d = 1$ neighborhoods.

7 CONCLUSION

Our investigation shows that ML techniques applied to automatically aligned dictionaries can achieve good results. Surprisingly, they also show that similar techniques can perform nearly as well on unaligned data (at least in some languages).

From a linguistic point of view, our results also show that G2P is a much more difficult problem in English than in Spanish or even German. While this is not a new discovery, the magnitude of the difference was a surprise.

We think it is worthwhile to continue exploring unaligned methods for G2P classification, since intermediate alignment by humans is extremely costly and ML techniques can introduce more noise. Given our results, future work may see unaligned G2P classification outperform aligned classification.

REFERENCES

[1] A.W. Black, K. Lenzo, V. Pagel. Issues in Building General Letter to Sound Rules. ESCA Workshop on Speech Synthesis '98. pp. 77-80. `http://www.cs.cmu.edu/~awb/papers/ESCA98_lts.pdf`

[2] Carnegie Mellon Pronouncing Dictionary. `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`

[3] Y. Freund, R. E. Schapire. Experiments with a New Boosting Algorithm. ICML '96. pp. 148–156

[4] S. Hanov. Automatic Letter-To-Sound Rules for Speech Synthesis. Lecture notes at University of Waterloo. `http://gandolf.homelinux.org/~smhanov/cs886_hanov_2007.pdf`

[5] International Phonetic Association. *International Phonetic Alphabet*. 2005. `http://www.arts.gla.ac.uk/IPA/IPA_chart_(C)2005.pdf`

[6] T. Kohonen. Dynamically Expanding Context, with application to the correction of symbol strings in the recogni-

tion of continuous speech. ICPR '86. pp. 1148–1151.

[7] D. Jurafsky and J. Martin. *Speech and Language Processing*. Prentice Hall, Upper Saddle River, NJ: 2000.

[8] The PASCAL Letter-to-Phoneme Conversion Challenge 2007 (PRONALSYL). `http://pascal-network.org/Challenges/PRONALSYL/`

[9] P. Roach. *A little encyclopedia of phonetics.* `http://www.personal.reading.ac.uk/~llsroach/peter/`

[10] K. Torkkola. An efficient way to learn English grapheme-to-phoneme rules automatically. ICASSP '93. Vol. 2. pp. 199–202.

[11] Z.-R. Zhang, M. Chu, E. Chang. An Efficient Way to Learn Rules for Grapheme-to-Phoneme Conversion in Chinese. ISCSLP '02.

WORK DIVISION

Throughout this project, Terry focused on the aligned algorithms whereas Kevin focused on the unaligned algorithms. Ideas were developed in joint brainstorming sessions and subsequently divided for implementation and improvement.