# CHAPTER SIX

# BOOTSTRAPPING PRONUNCIATION MODELS

## 6.1 INTRODUCTION

In this chapter we apply the grapheme-to-phoneme rule extraction mechanisms developed earlier in order to bootstrap pronunciation models. We analyse the bootstrapping process by developing pronunciation models in Afrikaans, a Germanic language with a fairly regular grapheme-to-phoneme relationship, and describe a number of experiments conducted to evaluate specific aspects of the bootstrapping process. In Section 6.5.4 we analyse the efficiency of the bootstrapping process according to the framework defined in Chapter 3. The completed system has since been used for the development of dictionaries in a number of additional languages (isiZulu, Sepedi and Setswana[1] ) and these dictionaries integrated in speech technology systems, as described in Section 6.6.

## 6.2 BOOTSTRAPPING SYSTEM

Two bootstrapping systems were developed:

- System A: The bootstrapping approach as described in Section 3.4 was implemented in Perl, to run within a Web browser [72]. This prototype provided an experimental platform for the evaluation of the various algorithms described in Chapter 4 and allowed initial measurements with regard to developer efficiency and accuracy. The experiments described in Sections 6.3 and 6.4 utilised this system.

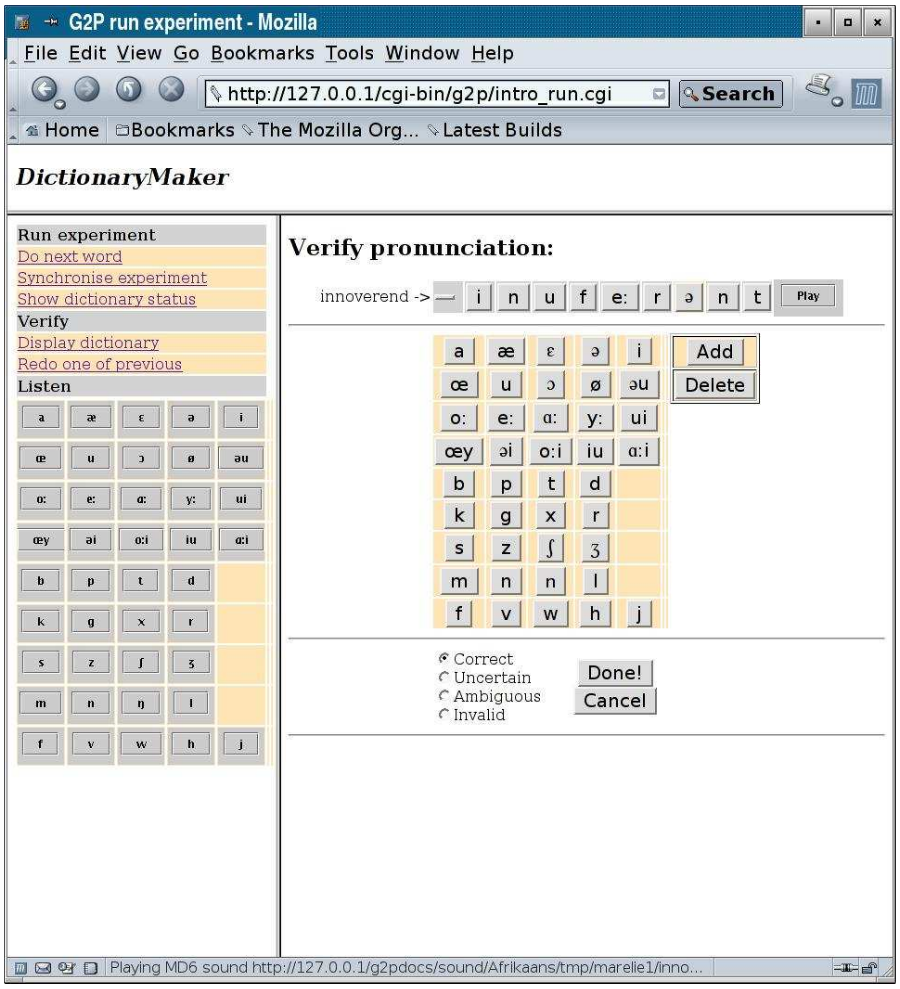
- System B: Components of System A were re-implemented in Java in order to provide more user-friendly interaction. The new system does not implement all the algorithms evaluated in

[1]Three more of South Africa's official languages, from the Bantu family.

this thesis, but provides a more robust platform for dictionary development[2]. System B was used in the experiment described in Section 6.5.

Both systems implement the bootstrapping approach described in Section 3.4, as described in more detail from both the user and system perspective in the next two sections.

### 6.2.1 USER PERSPECTIVE



[✓ Correct] Uncertain Ambiguous Invalid

Playing MD6 sound http://127.0.0.1/g2pdocs/sound/Afrikaans/tmp/marelie1/inno...

Figure 6.1: *Correcting the predicted pronunciations (System A).*

The dictionary development task as presented to the *verifier* is depicted in Fig. 6.1. The verifier is presented with each word/pronunciation pair in turn, and asked to provide a verdict of pronunciation accuracy. The verifier is required to verify all new predictions – none are assumed to be correct[3].

[2]This system will be released as Open Source Software in the near future – see http://www.csir.co.za/hlt for more information.

[3]In an alternative approach, Maskey *et al* [68] utilised a confidence metric to assume the correctness of some of the words. We preferred to verify all new predictions, given the unpredictability of some exceptions in pronunciation prediction tasks.

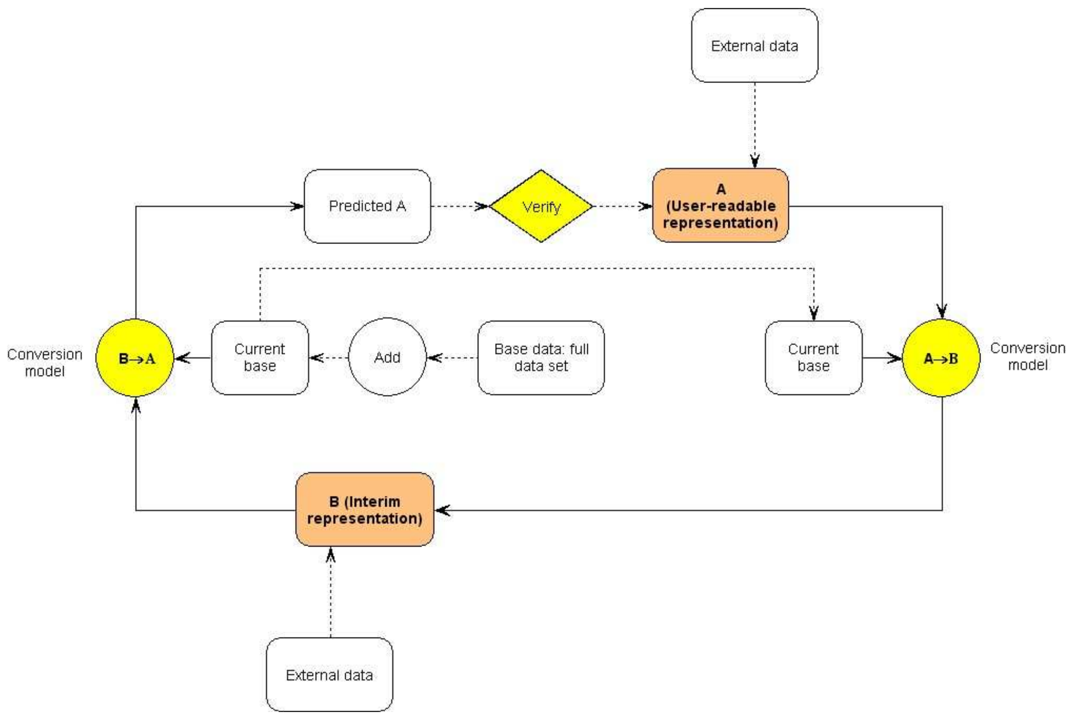


Figure 6.2: *The bootstrapping system concept.*

Once the word list and phoneme set have been loaded and the system prepared, no further expertise is required from the verifier apart from being able to differentiate between correct and incorrect pronunciations.

The verifier is presented with two representations of the pronunciation, namely a visual transcription and an audio version. The audio version is created by concatenating pre-recorded samples of each phoneme (i.e. the word is 'sounded' rather than synthesised). The verifier specifies a verdict: whether the pronunciation is *correct* as predicted, whether the word itself is *invalid*, *ambiguous* depending on context, or whether the verifier is *uncertain* about the status of the word. If the pronunciation is wrong, the verifier specifies the correct pronunciation by removing, adding or replacing phonemes in the presented pronunciation. Once the verifier is certain of the accuracy of a specific pronunciation, he or she is encouraged to listen to the audio version of the final pronunciation, and so identify potential errors. At any stage the verifier can *Redo* a word, in order to correct a previous mistake. The verifier can also *List possible errors* which provides a list of exceptional pronunciations, as discussed in more detail in Section 6.4.

### 6.2.2 SYSTEM PERSPECTIVE

Fig. 6.2 illustrates the bootstrapping concept from a system perspective. The bootstrapping system is initialised with a grapheme and phoneme set, and a large word list (containing no pronunciation information). Each phoneme is associated with a pre-recorded audio sample. The system can be primed with an existing rule set or dictionary, if available. If neither is available, the system will predict empty pronunciations initially, which, when corrected, form the basis for further bootstrapping.

Bootstrapping occurs in two phases. During the initial phase, the grapheme-to-phoneme models are updated whenever a word is verified as correct. In the second phase, a complete update (referred to as a synchronisation event) only occurs after a set of words has been verified as correct. In between synchronisation events, learning can either be ceased, or continued using an incremental algorithm[4]. The dictionary developer chooses the number of words at which the system progresses from the first to the second phase, as well as the size of the set corrected before models are synchronised with the new training data during the second phase. Once initialised, the following steps are repeated:

1. The system analyses its current understanding of the task and generates the next word to consider, as described in Section 6.2.3.
2. For the chosen word, the system generates a new pronunciation using its current grapheme-to-phoneme rule set.
3. The system creates a 'sounded' version of each word using the predicted pronunciation and associated sound samples, and records the verifier's final response.
4. If a word has been verified as correct, the system increases its update synchronisation counter. If an update event is due, the system updates its grapheme-to-phoneme rule set based on the new set of pronunciations.

This process is repeated (with increasingly accurate predictions) until a pronunciation dictionary of sufficient size is obtained.

### 6.2.3 ALGORITHMIC CHOICES

In the experiments conducted here we either use *DEC-min* or *Default&Refine* for rule extraction, as stated per experiment. We also state whether incremental learning is utilised between synchronisation events or not. A further algorithmic choice concerns the mechanism whereby the next 'best' word to add to the knowledge base is selected, as this can influence the speed at which the system learns. We utilise three different techniques in our experiments, as referred to in the various experiment descriptions:

- *Evenly selected from corpus:*
  Here we order the available word list alphabetically, and select every $n^{th}$ word in order to obtain a subset of the required size.

[4]Such as incremental Default&Refine, described in Section 4.6.4

- *Systematic growth in context:*
  The system grows its understanding of pronunciations-in-context systematically. Contexts of varying sizes are ordered according to occurrence frequency in general text, creating a list of 'contexts in question'. A continuous process predicts the next best word to verify based on the current state of the system: the shortest word is chosen that contains the next context in question. If so required, the system will attempt to obtain certainty on as many contexts of size *n* as possible, before continuing to a context of size *n+1*.

- *Random:*
  A subset is chosen at random.

An alternative approach is suggested in [68], where words are ordered according to frequency in general text, and the most frequent words are processed first. This provides the advantage that more frequent words are automatically included in the dictionary but can also decrease learning performance if the more frequent words tend to have irregular pronunciations, as is possible, depending on the specific language being considered.

### 6.2.4 SYSTEM CONFIGURATION

Fig. 6.3 depicts the options presented to the user preparing the dictionary development process. Displaying the current status, as shown here, is one task within an experimental environment that allows a user to manipulate and generate the various resources involved (the rule set, word list and pronunciation dictionary) as required. For each experiment, the system logs the history of all activities and archives the intermediary data resources for further analysis.

## 6.3 EXPERIMENT A: VALIDATION OF CONCEPT

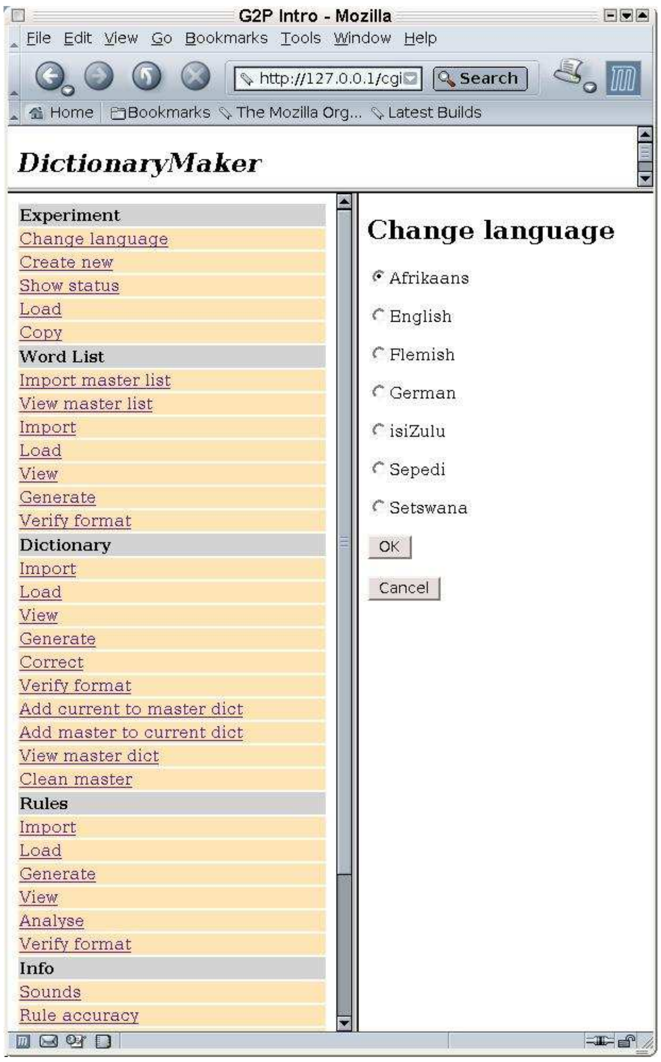
In this section we report on a series of experiments conducted in order to analyse the bootstrapping approach. The experiments are aimed at understanding a number of issues, including the following:

1. Can the bootstrapping approach be used to develop pronunciation dictionaries more quickly than conventional transcription?

2. How important is the linguistic background of the dictionary developer? Is it possible for a first language speaker without any phonetic training to develop an accurate pronunciation dictionary? (As mentioned in Section 1.1, this is highly significant in the developing world.)

3. How long does it take for a developer to become proficient with the bootstrapping system?

4. What are the practical issues that affect the speed and accuracy of dictionary development using the bootstrapping approach?



Change language

[✓ Afrikaans] English Flemish German isiZulu Sepedi Setswana

Figure 6.3: *Preparing the bootstrapping system (System A).*

In Section 6.3.1 we describe the experimental protocol followed. Utilising the framework defined in Chapter 3, we analyse the bootstrapping process from both a human factors perspective (Section 6.3.2) and a machine learning perspective (Section 6.3.3). In Section 6.3.4 we analyse the efficiency of the overall system, and compare expected and measured values.

### 6.3.1 EXPERIMENTAL PROTOCOL

The first set of experiments involved three dictionary developers who created pronunciation dictionaries for Afrikaans. All three developers are first-language Afrikaans speakers; and in informal interviews all three were found to employ a broadly similar dialect of "standard" Afrikaans. Two of the developers (whom we will refer to as A and B) have no formal linguistic training, whereas developer C has significant linguistic expertise, and has previous experience in the creation of pronunciation dictionaries.

The following protocol was used for all three developers:

1. A brief tutorial on the bootstrapping system, as well as the chosen phonetic representation, was presented by one of the experimenters.

2. A training set of 1000 words was drawn from a corpus of Afrikaans words, and the developers were given the opportunity to familiarise themselves with the system (and the phoneme set) by developing pronunciation rules for a subset of these words using the bootstrapping system. The process continued until the developers were satisfied that they were comfortable with the software and phoneme set.

3. A new set of 1000 words was selected, and the developers were asked to produce the most accurate rules they could, by listening to the sounded version produced by the system, correcting it if necessary, and repeating these two steps until satisfied with the pronunciation.

4. Further sets of 1000 words were used to experiment with various other factors, such as the effect of giving developers the option not to use audio assistance.

Each set of 1000 words was selected according to the 'systematic growth in context' word selection technique[5] from an independent 40,000-word subset of the full Afrikaans word list. The *DEC-min* algorithm was used for rule extraction, and all experiments were conducted in *phase 1* operation, that is, the rule set was updated after every corrected word. During these experiments we measured several relevant variables, including: the time taken to complete each verification; the number of phonemes changed per word verified; whether the developer chose to use the audio assistance; whether a developer returned to a word to re-correct it at a later stage; and the amount of idle (resting) time between sets of verifications.

### 6.3.2 HUMAN FACTORS

#### *6.3.2.1 USER LEARNING CURVE*

To measure a developer's facility in using the bootstrapping software, it is useful to obtain separate measurements of how long it takes (on average) to verify words in which no corrections are made, words where one correction is made, words where two corrections are made, etc. This eliminates the confounding effect of the system becoming more accurate as it learns more rules (thus accelerating apparent developer performance). By this measure, all three developers reached a satisfactory level of performance within approximately 400 words. For example, Fig. 6.4 depicts how the times for developer C to correct zero through four errors converge to their stable values; similar tendencies were seen for the other developers as well.

[5] as described in Section 6.2.3

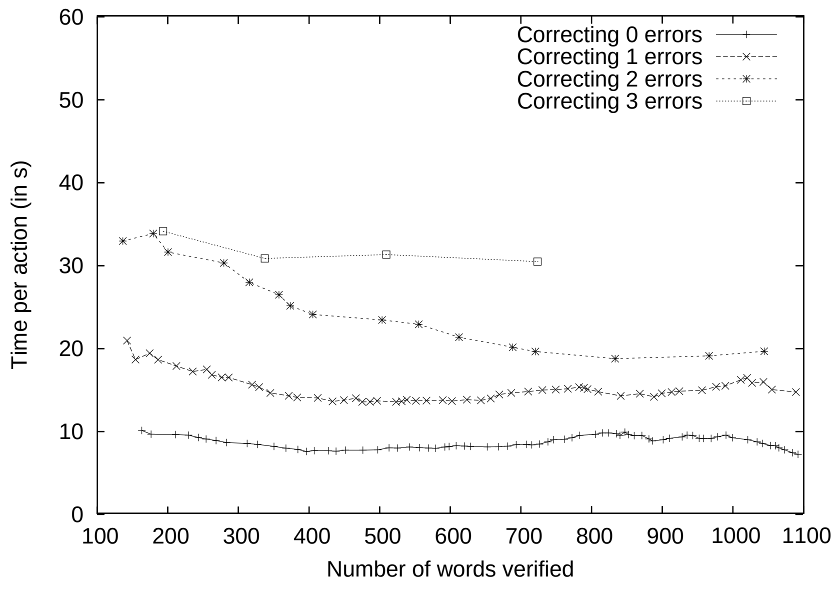


Figure 6.4: *Average time taken by developer C to verify words requiring zero, one, two or three corrections, as a function of the number of words verified. The averages were computed for blocks of 50 words each.*

This is highly encouraging, since the initial 400 words were completed in less than two hours in every case. Even linguistically untrained users can therefore become proficient at using bootstrapping within this length of time.

### *6.3.2.2 EFFECT OF LINGUISTIC EXPERTISE*

The ability of linguistically untrained users to become proficient at using the bootstrapping system does not necessarily imply that the users were using the system accurately. It is an interesting question whether it is at all possible for a first language speaker without any phonetic training to develop an accurate pronunciation dictionary.

In order to analyse the effect of linguistic sophistication, the performance of developers A and B (who have had no linguistic training) was compared with that of developer C along the dimensions of speed and accuracy. Because there is unavoidable ambiguity in defining “correct” pronunciations (even within a particular dialect), we measured accuracy by manually comparing all cases where any pair of developers chose different transcriptions for a word. In those cases, a transcription was flagged as erroneous if (in the opinion of the author) it did not represent an accurate transcription of the word.

Table 6.1: *Estimated transcription accuracies of three developers on a set of 1000 words.*

| Developer | Transcription experience | Word accuracy |
|---|---|---|
| A | None | 83.6% |
| B | None | 98.0% |
| C | Substantial | 99.0% |

Table 6.1 summarises the accuracies of the three developers, as estimated using this process. Only words marked as “valid” by a developer were included in the evaluation. As expected, developer C

was found to be highly accurate. Interestingly, developer B was only slightly less accurate, whereas developer A made significantly more errors than either of the others. During analysis it was revealed that developer A had not adhered to the protocol defined in Section 6.3.1: when confident of the accuracy of a pronunciation, developer A had accepted pronunciations without utilising the audio assistance provided by the system. Two conclusions are suggested by these measurements:

- It is possible for a linguistically inexperienced developer to use the bootstrapping system to attain levels of speed and accuracy comparable to those of a highly proficient dictionary developer.
- Developers with limited linguistic experience should be required to listen to every transcription, since it is easy to become over-confident about one's ability to read phonetic transcriptions.

#### 6.3.2.3 *THE COST OF USING AUDIO ASSISTANCE*

Since we found that the developer who did not sound words out made many more errors than those who did, it is important to investigate how much this sub-process delays the process of verification. To this end, we asked developer C to verify an additional set of 200 words, only choosing to sound out those words where she considered it useful. In Fig. 6.5 the time taken to verify words with various numbers of corrections is compared with the times when the use of audio assistance was compulsory.

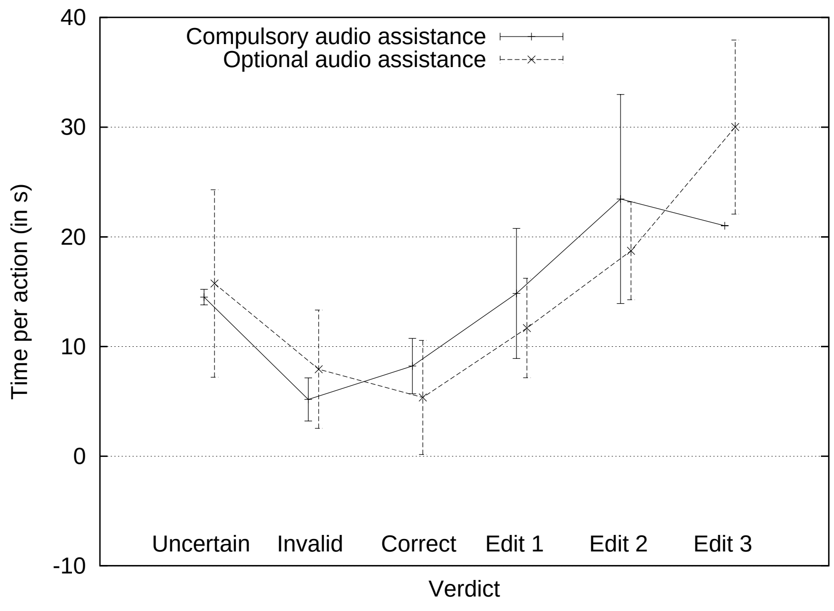


Figure 6.5: *Average time taken by developer C to verify words, with and without compulsory use of audio assistance.*

We found that this choice did not cause the developer to commit any errors; however, the reduction in verification time was also relatively small (3.6 seconds on average). This confirms the suggestion in Section 6.3.2.2 that it is generally better not to make the use of audio assistance optional.

### *6.3.2.4 THE COST OF PHONEME CORRECTIONS*

The number of phoneme corrections required is the dominant factor in determining verification time. For example, analysis shows that the length of the words to be verified correlates with the verification time if no corrections are required, but not if one correction is required, and that word length is the less important of these two factors. (Word length similarly does not predict verification time if two or more corrections are required.) Developers take comparable durations to perform their verifications, as shown in Fig. 6.6.

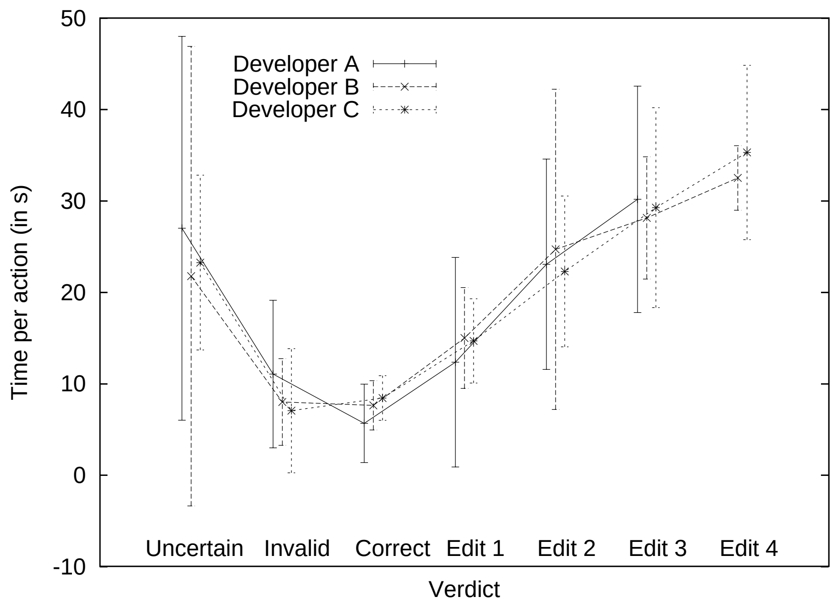


Figure 6.6: *Average time taken by three developers to verify words requiring different numbers of corrections (or to mark words as invalid or ambiguous/uncertain). The averages were computed for the same set of 1000 words as above.*

### *6.3.2.5 RELATED FACTORS*

Our experiments have underlined a number of practical factors that need to be taken into account when developing pronunciation dictionaries using bootstrapping:

- Relatively informal instruction of the developers is sufficient, if they are given the opportunity to learn by using the system.
- The appropriate definition and usage of the phoneme set requires some care. When a new language is being developed, it is advisable to do this in an iterative fashion: developers develop a small dictionary, and their comments as well as transcriptions are reviewed to determine whether any phonemes are absent from the set being used, and also to determine what conventions are required to ensure consistency of the dictionary.
- For a linguistically inexperienced dictionary developer, the audio samples used should ideally match the developer's regional accent.

- When developers have limited linguistic experience, they should be required to listen to every word prior to final acceptance of a transcription.

### 6.3.3 MACHINE LEARNING FACTORS

#### *6.3.3.1 SYSTEM CONTINUITY*

The faster the system learns, the fewer corrections are required of the human verifier, and the more efficient the bootstrapping process becomes. The most important aspect that influences the speed at which the system learns relates to the continuity with which the system updates its knowledge base. A continuous process was chosen, whereby the system regenerates its prediction models after every single word verified. This has a significant effect on system training responsiveness, especially during the initial stages of dictionary development when the system has access to very little information on which to base its predictions.

#### *6.3.3.2 PREDICTIVE ACCURACY*

The increasing likelihood that the system will correctly predict pronunciations as more words are verified is depicted in Fig. 6.7, which shows the average number of phoneme corrections required as a function of the number of words verified by developer B. The number of corrections decreases steadily as more words are verified, producing an increasingly accurate dictionary and enabling the developer to process subsequent words more rapidly.

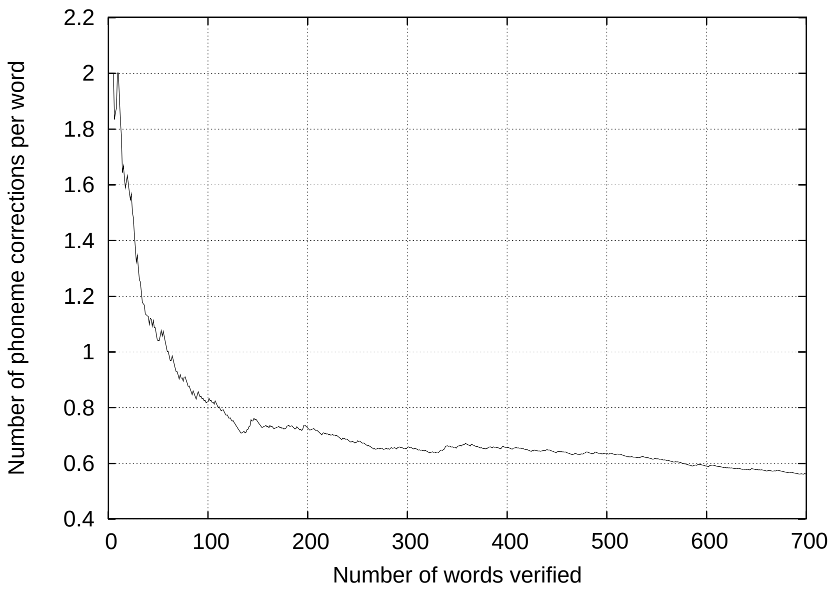


Figure 6.7: *Expected number of phonemes that required correction by developer B as a function of the number of words verified.*

#### *6.3.3.3 VALIDITY OF BASE DATA*

A final factor that influences the speed of dictionary development concerns the validity of the initial word lists. In this set of experiments word lists were obtained from Internet text and contained up to $15\%$ invalid words.

### 6.3.4 SYSTEM ANALYSIS

We can combine the information in Figs. 6.7 and 6.6 to derive a model of how long it will take system users such as developers B and C to create pronunciation dictionaries of various sizes. To do this, we fit an exponential curve through the smooth part of the graph in Fig. 6.7 (i.e., for 100 or more words verified), and estimate a linear model for the expected verification time as a function of the required number of corrections. Fig. 6.8 shows how machine learning produces slower-than-linear growth in development time, and that a fairly sizeable dictionary can be created in fewer than 20 hours of developer time. The bootstrapping approach is compared to manual verification at 19.2s and 30s per word. (19.2s was the fastest average time observed in our laboratory using a proficient phonetic transcriber, and represents an optimistic time estimate.)

Also note that the model of expected development time, which was based on measurements of the time taken by Developer B, predicts Developer C's measurements with reasonable accuracy.

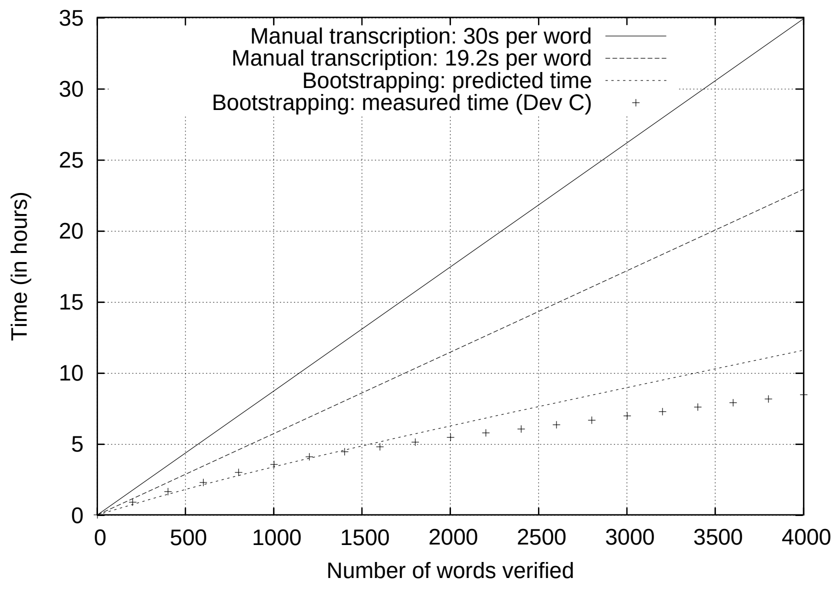


Figure 6.8: *Expected time (in hours) required to compile an Afrikaans pronunciation dictionary, as a function of dictionary size.*

From this set of experiments we conclude that a bootstrapping approach can be used to generate pronunciation dictionaries efficiently. Encouragingly, similar estimates are found for an experienced creator of pronunciation dictionaries (with significant linguistic training), and a developer with no prior exposure to formal linguistics.

## 6.4 EXPERIMENT B: SEMI-AUTOMATIC DETECTION OF VERIFIER ERRORS

Dictionary developers are typically required to enter phonemic predictions for several thousand words in order to develop dictionaries of sufficient accuracy. Although our interface attempts to assist developers in this task (e.g. by audibly sounding out the chosen pronunciations and by providing automatic predictions for every word), it is inevitable that errors will arise from time to time.

Fortunately, the *Default&Refine* approach is well suited to assist in the detection of such errors. Since every rule in the hierarchy is selected to describe a particular set of words, and errors are likely to result in rules that are applicable to few words besides the erroneous one, one expects that erroneous transcriptions will tend to show up as rules which support few words. Of course, there may also be valid pronunciation rules which are not supported by many examples; it therefore is an experimental issue to determine how useful this guideline is in practically detecting transcription errors. Different languages will differ in this regard – a highly "regular" language such as Spanish [6] will generally have many examples of each valid rule, whereas the idiosyncrasies of English pronunciation will produce a large number of valid special cases. As a consequence, our approach is expected to be more successful for languages such as Spanish.

To investigate the utility of the proposed method for detecting transcription errors, we conduct a number of simulation experiments with Afrikaans. Heuristically, we expect Afrikaans to lie somewhere in the middle of the continuum between regular and irregular languages. Our experiments use a verified dictionary with 4 923 valid words (*Afrikaans A*). Based on earlier experience with dictionary developers who are error prone (see Section 6.3.2.2), we artificially corrupt a fraction of these transcriptions and then measure the efficiency of the number-of-words guideline to indicate the words with corrupted transcriptions. This is the similar to the process followed in Section 4.7.4 where we evaluated the effect of noise on the predictive ability of the *Default&Refine* and *DEC-grow* algorithms.

As in Section 4.7.4 we introduce two types of corruptions into the transcriptions:

- *Systematic corruptions* reflect the fact that users are prone to making certain transcription errors - for example, in the DARPA phone set, *ay* is often used where *ey* is intended. We allow a number of such substitutions, to reflect observed confusions by Afrikaans transcribers.
- *Random corruptions* simulate the less systematic errors that also occur in practice; in our simulations, random insertions, substitutions and deletions of phonemes are introduced.

We generate four corrupted data sets (systematic substitutions; random insertions, substitutions and deletions), where 1% of the words are randomly selected for corruption. *Default&Refine* rule sets are then generated for each case, and the percentage of erroneous words that are matched by the most specific rules are determined[7]. In Fig. 6.9 we show the fraction of errors that remain undetected

[6] That is, a language with a very regular mapping between phonemes and graphemes.

[7] Since *Default&Refine* always applies rules in the order most to least specific, the rule ordering used for prediction was

against the fraction of words examined, as this threshold of specificity is adjusted. Note that this depiction is closely related, but not identical, to that in the well known Detection Error Tradeoff (DET) curves [73].

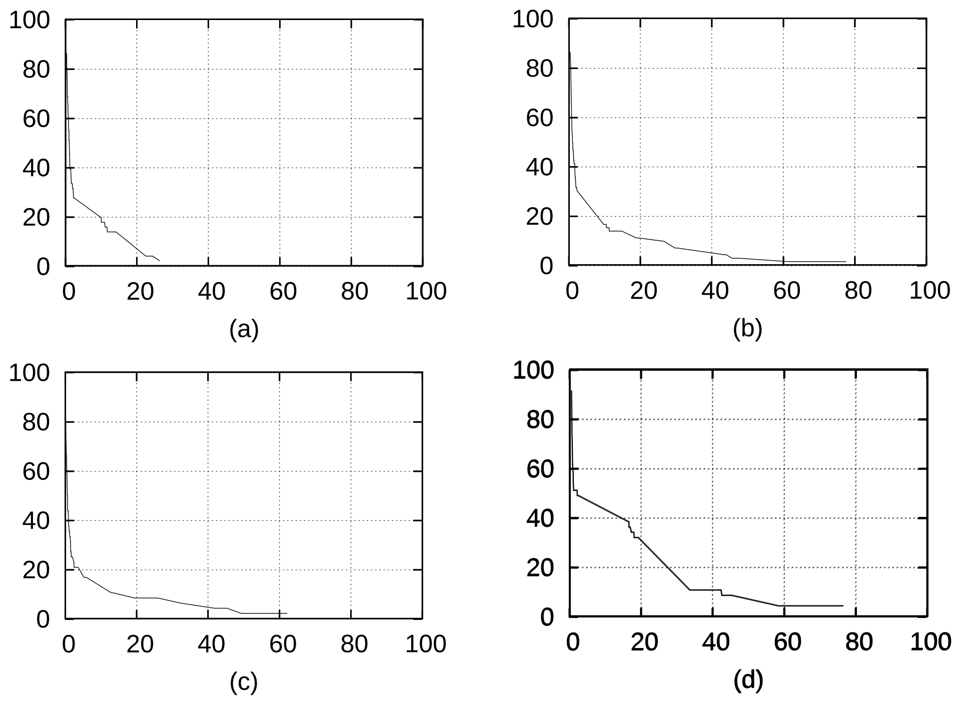


Figure 6.9: *Fraction of erroneous words that are not detected as a function of the fraction of all words examined, when words are examined in the order of their most specific rules, for various types of corruptions: (a) random substitutions (b) random insertions (c) random deletions and (d) systematic substitutions.*

These results suggest that this method has significant use in accelerating the process of error detection. For all three types of random errors, more than 90% of the errors can be identified after inspecting fewer than 20% of the transcriptions. As far as the systematic errors are concerned, about half the errors occur in the first 5% of the words inspected; by that time, the systematic patterns are obvious, and can be used to select other candidate words where these same errors may have occurred.

In practice, the error-detection process can be combined with the synchronisation event, with possible errors flagged by the bootstrapping system and corrected where necessary by a human verifier, prior to continuing with the next session. This then becomes a simple and efficient way of identifying errors during bootstrapping. Alternatively, the error-detection process can be used as a stand-alone technique, in order to identify possible errors in a pronunciation dictionary developed via different means.

used as measure of specificity. The specificity of a word is taken as the specificity of its most specific grapheme, since a transcription error may result in one or more rules becoming highly specific to that word.

## 6.5 EXPERIMENT C: BUILDING A MEDIUM-SIZED DICTIONARY

In the final controlled experiment we build a medium-sized Afrikaans dictionary utilising the new techniques developed in this thesis. In section 6.5.1 we define our experimental protocol and in the remainder of this section we analyse the efficiency of the process according to the framework defined in Section 3.

### 6.5.1 EXPERIMENTAL PROTOCOL

Up to this point, the various dictionaries developed during experimentation were fairly small (approximately 1000 to 2000 words). In this experiment, we verify the effectiveness of the various techniques when building a medium-sized dictionary in a continuous process. Since we are growing the dictionary from a previous baseline we are specifically interested in the extent to which the bootstrapping process supports the extension of an existing dictionary.

We utilise one of the developers (Developer C) who has previous experience in using the bootstrapping system. We perform bootstrapping using System B, and initialise the bootstrapping system using the dictionary *Afrikaans A*[8]. We use incremental *Default&Refine* for active learning in between synchronisation sessions, and standard *Default&Refine* during synchronisation. We set the update interval (number of words modified in between synchronisations) to 50, and order words randomly (in the list of new words to be predicted).

At the end of the bootstrapping session we perform error detection. (No additional error detection is performed during bootstrapping.) We first extract the list of graphemic nulls, and identify possible word errors from the graphemic null generators. We then extract *Default&Refine* rules from the full dictionary with the purpose of utilising these rules to identify errors, similar to the process described in Section 6.4. We list all words from word sets that result in a new rule and contain fewer than five words as possible errors, and verify these words manually[9].

### 6.5.2 HUMAN FACTORS ANALYSIS

We measure the time taken by the verifier (developer C) to perform each verification action, and analyse the effectiveness of the verification process from a human factors perspective. Fig. 6.10 illustrates the verification process as the dictionary grows from 5500 to 7000 words. We plot the time taken to verify each valid word, indicating whether 0,1,2, or 3 corrections are required, for each word as it is added to the dictinary. (The number of training words on the x-axis includes both valid and invalid words.)

We note the following:

[8]As described in Section 4.3, we create the *Afrikaans A* dictionary by cross-analysing the dictionaries from the various experiments run to date and manually verify discrepancies.

[9]A word set associated with a rule tends to have either only one or two words associated with it, or a large set of words: within an acceptable range, the error detection process is not sensitive with regard to the exact cut-off point selected.

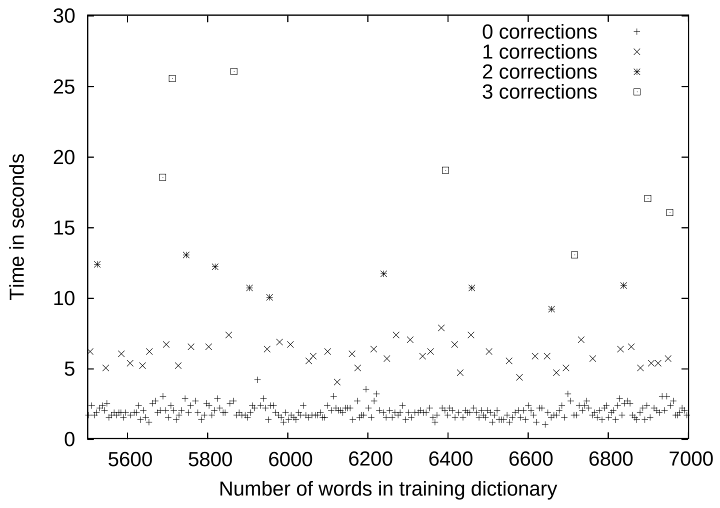


Figure 6.10: *Time taken to verify words requiring zero, one, two or three corrections, as a function of the number of words verified. For the first three measures, the averages were computed for blocks of 5 words each.*

- *User learning curve:* Developer C was proficient in using the system prior to the current bootstrapping session, and further training was not required[10].

- *Cost of intervention:* In this experiment we utilised two intervention mechanisms: verifying predictions, and verifying the list of possible errors. Table 6.2 provides the average verification times observed for Developer C where the intervention mechanism is a single verification of a prediction ($t_{verify(single,s)}$) for words that are in different states $s$ prior to verification. Verification of the list of possible errors took approximately 27 minutes (for approximately 3000 words).

Table 6.2: *Statistics of the time taken to verify words requiring 0,1,2 or 3 errors, or to identify a word as invalid or ambiguous ($\mu$ is the mean, and $\sigma$ the standard deviation.).*

| Verdict | Time in seconds | |
|---|---|---|
| | $\mu$ | $\sigma$ |
| correct | 1.95 | 1.35 |
| 1 error | 5.79 | 2.30 |
| 2 errors | 10.74 | 3.19 |
| 3 errors | 17.91 | 6.12 |
| invalid | 3.39 | 4.71 |
| ambiguous | 8.92 | 5.08 |

- *Task difficulty:* During the bootstrapping process, 3019 words were added to the dictionary, of which 181 were invalid or ambiguous. During error detection, 9 errors were found in the remaining 2838 valid words. Given our analysis in Section 6.4 we estimate that this represents

[10]The value of $t_{train}$ during the initial session was $< 120\ min$.

at least 50% of the errors, and therefore estimate the actual error rate to be $0.6\%$[11]. It is interesting to note that, while our error detection protocol resulted in a re-verification of $3.3\%$ of the full dictionary (1832 grapheme-specific patterns, or about 300 words), the average position of each error in the ordered error prediction list was at $0.67\%$ of the full training dictionary, with the majority of errors found in the first $0.1\%$ of words, i.e. the first or second pattern on the per-grapheme list of potential errors.

- *Difficulty of manual task:* $error\_rate_{manual}$ is assumed to be $< 0.5\%$, which is an optimistic estimate for the range of manual development speeds evaluated.
- *Manual development speed:* Different values of $t_{develop}$ are used for comparison, ranging from $19.2s$, again an optimistic estimate.
- *Initial set-up cost:* As this is an extension of an existing system, no further set-up cost was incurred[12].

### 6.5.3 ANALYSIS OF MACHINE LEARNING FACTORS

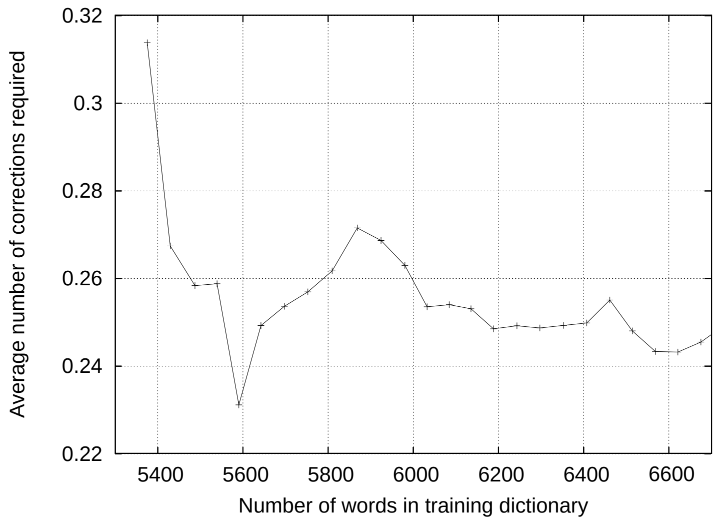


Figure 6.11: *The average number of corrections required as a function of the number of words verified. Averages were computed for blocks of 50 words each.*

From a machine learning perspective, the following is observed:

- *Predictive accuracy of current base:* Measured directly during experimentation, the number of corrections required per word added to the dictionary $(inc\_n(s, n))$ is depicted in Fig. 6.11. We plot the running average (per blocks of 50 words) of the number of corrections as a function of the number of words verified.

[11] 18 errors in 2838 valid words.

[12] In the previous experiment $t_{setup\_bootstrap}$ - $t_{setup\_manual}$ $< 60\ min$.

- *On-line conversion speed:* The average time taken for a synchronisation event was $50.15$ seconds ($\sigma = 7.72s$). This value increased gradually from $35s$ during the initial cycle, to $56s$ in the final cycle.
- *Quality and cost of verification mechanisms:* The computational times required for both verification mechanisms are included in the verification times. No additional processing is required.
- *Validity of base data:* $valid\_ratio = 94\%$.

### 6.5.4 SYSTEM ANALYSIS

Based on our observations during this experiment, we can assign approximate values to the different costs and efficiencies involved during bootstrapping of an Afrikaans dictionary up to 10,000 words. We list these values in Table 6.3.

Table 6.3: *Typical observed values for various bootstrapping parameters.*

| Bootstrapping parameter | | Estimated value |
|---|---|---|
| Training cost | $t_{train}$ | $< 120$ min |
| Verification cost for single words, with x corrections required for a word in state s: | $t_{verify(single,s)}$ | $(2 + 4.5x)$ sec |
| Verification cost during error detection (per 1000 words): | $t_{verify(error-det)}$ | $< 10$ min |
| Verification cost during error detection (per 400 words): | $t_{verify(error-det)}$ | $< 3$ min |
| Task difficulty - bootstrapping, no error detection | $error\_rate_{bootstrap}$ | $0\% - 1\%$ |
| Task difficulty - bootstrapping, error detection | $error\_rate_{bootstrap}$ | $0\% - 0.5\%$ |
| Task difficulty - manual | $error\_rate_{manual}$ | $0 - 0.5\%$ |
| Manual development speed | $t_{develop}$ | $19.2 - 30$ sec |
| Initial set-up cost | $t_{setup\_bootstrap}$ - $t_{setup\_manual}$ | $< 60$ min |

We use eq. 3.4 to analyse our results, and for the single word verifier we combine the values of $t_{auto(s,single)}$ with $t_{verify(s,single)}$ as a single measurement, as discussed in the previous section. We also combine the value of $t_{idle}$ with $t_{verify(error-det)}$, as these two events both occur during synchronisation. We then obtain the following expected cost of $N$ cycles of bootstrapping:

$$E[t_{bootstrap}(N)] = E[t_{setup\_bootstrap}] + E[t_{train}] + E[t_{iterate}(N)] \tag{6.1}$$

$$E[t_{iterate}(N)] = \sum_{x=1}^{N-1}\left(\sum_{s\in status}(E(t_{verify}(s, single)).E(inc\_n(s, x))\right)$$
$$+\sum_{x=1}^{N-1}\left(t_{idle}(inc\_n(valid, x+1)) + t_{verify(error-det)}(inc\_n(valid, x+1))\right) \tag{6.2}$$

We assume an update event after every 100 errors (approximately 400 words verified.) As $t_{idle}$ is dominated by $t_{verify(error-det)}$ during the initial 10,000 words, we keep this value constant as the number of words in the training dictionary increases[13], and estimate it at:

$$t_{verify(error-det)}(400) + t_{idle}(400) = 180 \text{ seconds} \tag{6.3}$$

From Table 6.3 we estimate $E(t_{verify}(s, single))$ as $t_0 + t_e x$ seconds, where $x$ is an indication of the number of corrections required, $t_0 = 2$ and $t_e = 4.5$. In order to estimate $\sum_{x=1}^{N-1} E(t_{verify}(s, single))E(inc\_n(s,x))$ for different states (different numbers of corrections per word) we smooth the number of errors across the training data – as if a word could only have one error – and fit an exponential curve through the accuracy measurements depicted in Fig. 6.11. That is, we assume the probability that the system will predict an error when the training dictionary is of size $d$ is given by $p_e(d)$, where:

$$\begin{aligned} p_e(d) &= P_0 e^{-\frac{d}{k}} \\ \text{i.e. } \log p_e(d) &= \log P_0 - \frac{d}{k} \end{aligned} \tag{6.4}$$

and $P_0$ and $k$ are parameters to be estimated. The time required for $d$ corrections $T(d)$ (excluding synchronisation events) is then given by:

$$\begin{aligned} T(d) &= \sum_{i=0}^{d-1}(t_0 + t_e P_0) \\ &= dt_0 + t_e P_0 \sum_{i=0}^{d-1} e^{-\frac{d}{k}} \\ &= dt_0 + t_e P_0 \frac{1-e^{-\frac{d}{k}}}{1-e^{-\frac{1}{k}}} \end{aligned} \tag{6.5}$$

For the specific data depicted in Fig. 6.11 we obtain the estimates:

$$\begin{aligned} logP_0 &= -1.274 \\ -\frac{1}{k} &= -3.49 * 10^{-5} \end{aligned} \tag{6.6}$$

We can combine eq. 6.2 and eq. 6.5 in order to estimate the value of $E[t_{iterate}(d/400)]$ for various values of total dictionary size $d$:

$$\begin{aligned} E[t_{iterate}(d/400)] &= dt_0 + t_e P_0 \frac{1-e^{-\frac{d}{k}}}{1-e^{-\frac{1}{d}}} \\ &\quad + \frac{d}{400} * (t_{verify(error-det)}(400) + t_{idle}(400)) \end{aligned} \tag{6.7}$$

[13]This value is influenced by the number of words corrected per cycle – a number that remains constant per cycle.

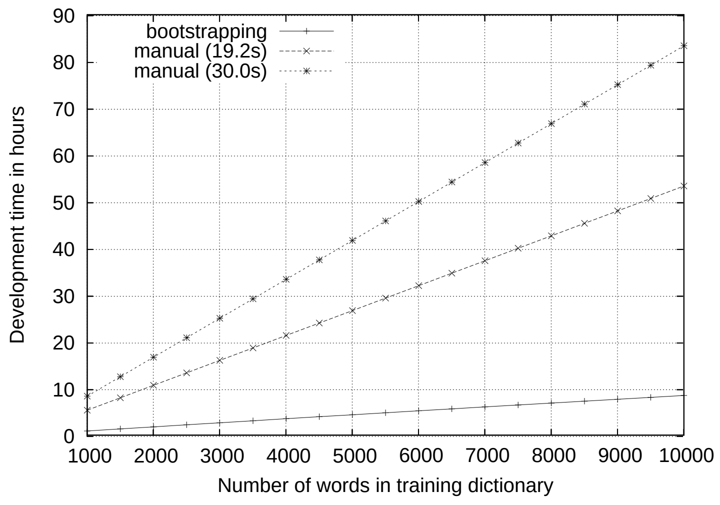


Figure 6.12: *Time estimates for creating different sized dictionaries. Manual development is illustrated for values of* $t_{develop}(1)$ *of* $19.2$ *and* $30$ *seconds, respectively.*

In Fig. 6.12 we plot eq. 6.7 for different values of $d$, using the estimates from eq. 6.3 and eq. 6.6. On the same graph we plot the cost of manual dictionary development (again excluding setup cost) using eq. 3.5 and estimates for $t_{develop}(d)$ of $19.2$ and $30$ seconds, both optimistic estimates. For these estimates we assume that the same base data (or at least data with a similiar validity ratio) is used for both approaches. We also assume that the error rates for the bootstrapping system with error detection and the manual process are approximately equal. In Fig. 6.13 we plot the efficiency estimates of the bootstrapping process as compared to a manual dictionary development process for the same values as Fig. 6.12.

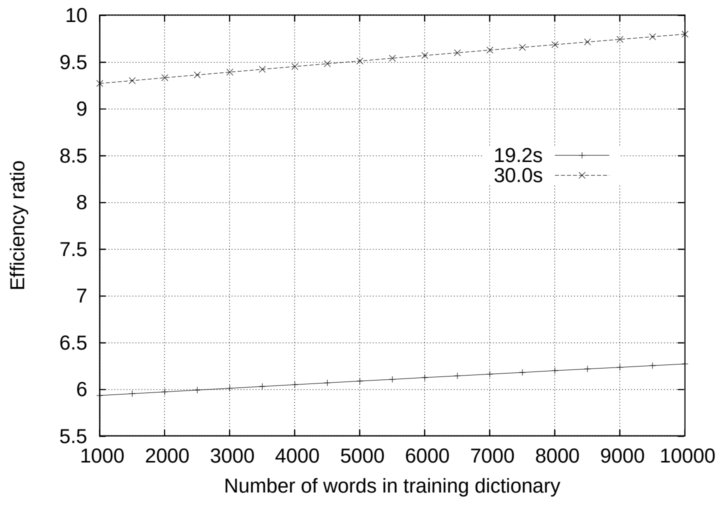


Figure 6.13: *Estimates of the efficiency of bootstrapping, as compared with manual development for values of* $t_{develop}(1)$ *of* $19.2$ *and* $30$ *seconds, respectively.*

## 6.6 BUILDING SYSTEMS THAT UTILISE BOOTSTRAPPED DICTIONARIES

In the work up to this point we have verified the bootstrapping process through (1) simulated experiments in which an actual pronunciation dictionary existed, and was utilised as a pseudo-verifier, and (2) by creating multiple dictionaries using different human verifiers and comparing the results. In this section we describe a number of speech technology systems that were developed using the bootstrapped dictionaries.

### 6.6.1 ISIZULU TEXT-TO-SPEECH

The first system developed using a bootstrapped dictionary was a general purpose text-to-speech (TTS) system developed in the Festival [74] framework as part of the Local Language Speech Technology Initiative (LLSTI) [75], a collaborative project that aims to support the development of speech technology systems in local languages. A small grapheme-to-phoneme rule set was generated using the bootstrapping system and converted to the Festival letter-to-sound format. (The *DictionaryMaker* prototype can automatically export a developed dictionary as either a Festival-formatted lexicon or Festival-formatted letter-to-sound rules.)

The TTS system used the *Multisyn* approach to synthesis and is described in more detail in [76] and [77]. The completed system was evaluated for intelligibility and naturalness by both technologically sophisticated and technologically unsophisticated users, as described in [78].

Table 6.4: *Parameters of the isiZulu text-to-speech dictionary*

| | |
|---|---|
| Number of graphemes in orthography | 26 |
| Number of phonemes in phoneme set | 50 |
| Number of words in dictionary | 855 |
| Number of derived rules (*DEC-min*) | 84 |

### 6.6.2 SEPEDI SPEECH RECOGNITION

During 2004, the University of Limpopo collected a first corpus of Sepedi (Northern Sotho) speech with the purpose of creating an automatic speech recognition (ASR) system, and required a pronunciation dictionary in order to proceed with further development. In collaboration with partners from the University of Limpopo, a bootstrapped dictionary was created. Again a fairly small number of words were bootstrapped in order to develop a concise set of letter-to-sound rules. These were then used to develop a speech recognition system using the HTK [79] framework, as described in [80].

### 6.6.3 AFRIKAANS TEXT-TO-SPEECH

Much of the initial experimentation with the bootstrapping approach was performed for Afrikaans, as described in previous sections of this thesis. The Afrikaans dictionary was used to develop a

Table 6.5: *Parameters of the Sepedi speech recognition dictionary*

| | |
|---|---|
| Number of graphemes in orthography | 27 |
| Number of phonemes in phoneme set | 41 |
| Number of words in dictionary | 2827 |
| Number of derived rules (*DEC-min*) | 90 |

Afrikaans TTS system for the South African Centre for Public Service Innovation (CPSI), who are using the voice to pilot a system that will allow citizens to interact with a governmental service that deals with passport applications via a number of interaction mechanisms not previously available. One of the mechanisms tested includes the use of cellphone based Short Message Service (SMS) to communicate, and converting such SMSs to voice when a user prefers a voice-based service – mainly in order to ensure accessibility to all citizens, including illiterate system users, and system users with specific disabilities. This system is currently being piloted.

Table 6.6: *Parameters of the Afrikaans text-to-speech dictionary*

| | |
|---|---|
| Number of graphemes in orthography | 40 |
| Number of phonemes in phoneme set | 43 |
| Number of words in dictionary | 7782 |
| Number of derived rules (*Default&Refine*) | 1471 |

### 6.6.4 OTHER SYSTEMS

The CPSI pilot project described above aims to provide services in four languages: English, Afrikaans, isiZulu and Sepedi; a Sepedi voice similar to those described in Sections 6.6.1 and 6.6.3 was therefore developed, using the dictionary built as described in Section 6.6.2. Further development on the Sepedi voice is currently under way, specifically aimed at improving the intonation contours of the current voice.

Furthermore, an initial isiZulu ASR system and an Afrikaans ASR system were developed, with further optimisation currently in progress. A first Setswana dictionary was developed, and will be refined and integrated in similar systems as part of the OpenPhone [81] project, a project sponsored by the International Development Research Centre (IDRC) and the Open Society Initiative (OSI), which aims to make telephony services more accessible to information service providers in the developing world.

## 6.7 CONCLUSION

In this section we demonstrated the practical application of the bootstrapping system, evaluating the efficiency of the approach from both a human factors and a machine learning perspective. We found that, even with optimistic estimates for the time required to develop a single instance of a pronunciation dictionary manually, the bootstrapping process provides a significant cost saving, as illustrated in Fig. 6.12. We also described a number of speech technology systems developed using newly bootstrapped dictionaries. In the next chapter (Chapter 7) we discuss the implications of our results.