M.Phil. in Computer Speech, Text and Internet Technology

Statistical Grapheme to Phoneme Conversion using Language Origin

Paul J Moore

Hughes Hall
University of Cambridge

20 July 2006

This dissertation is submitted for the degree of Master of Philosophy

Declaration

I Paul J Moore of Hughes Hall, being a candidate for M.Phil. in Computer Speech, Text and Internet Technology, hereby declare that this dissertation and the work described in it are my own work, unaided except as may be specified below, and that the dissertation does not contain material that has already been used to any substantial extent for a comparable purpose.

The majority of scripts and software were written by the author but several short scripts, identified in Appendix 2, were re-used or adapted from originals written by the supervisor. The cleaned Unisyn lexicon created by the supervisor was reused for this project. The CELEX pronunciation lexicons and the GEONet names data were processed by the author.

Signed

P J Moore

Date 20 July 2006

Word count: 12534

Abstract

This report describes a method for grapheme to phoneme conversion using statistical models of pronunciation. The available techniques for this conversion are first described and examples of each are given. A baseline system which uses Hidden Markov Models to represent phonemes in English is described and evaluated. The results from the baseline system serve to replicate previous research and to function as a means of comparison for extensions created for this project.

The system is first extended to multiple languages using the CELEX English, German and Dutch lexicons, and results of 40.89%, 78.55% and 74.48% words correct respectively are obtained for a 4-gram phoneme model. The report then describes the classification of language origin using a letter n-gram model, similar to the language model in an automatic speech recognition system. An implementation of the language recogniser gives an overall language recognition accuracy of 90.48%.

The language identification and grapheme to phoneme stages are combined to create an integrated system. Two general methods of combining the stages are investigated. The first involves the language identification stage generating a result about the language origin of a word and the grapheme to phoneme converter appropriate to that language being used. The second approach combines the language identification probabilities with the values returned by the grapheme to phoneme models for all the languages. It is found that such a probabilistic treatment of language origin provides an enhanced result compared with a hard decision by the language classifier.

Table of Contents

List of Figures and Tables

1 Introduction

Grapheme to Phoneme conversion is usually accomplished by looking up a phoneme sequence from the orthography in a pronunciation lexicon. Over recent years, the coverage of dictionaries has increased as memory and disk space has become cheaper and more data is created and shared. This trend looks likely to continue and it is possible to consider the scenario when almost all names are contained within a lexicon. Despite this situation, some words will be missing and so there will be a need to convert from the orthography to a phonemic representation by automatic means. Some words are not included in the lexicon, even if they exist in the language at the time of compilation, because they are obscure or little-used. Other words are added to the language during the lifetime of the lexicon, such as neologisms or commercial terms that become generic (for example "Hoover" or "Google"). Loan words or names from other languages may become current as the political or news situation changes (as for example "Zidane"). In all these cases, the dynamic nature of language makes it hard to keep a lexicon up-to-date with open-class words.

There may be also engineering reasons for limiting the size of the lexicon. While some applications have access to large resources of memory, others, such as mobile or embedded programs, do not. There might also be reasons for relying on an automatic conversion in preference to a large lexicon if it is worthwhile to trade off memory footprint versus correct pronunciation. So the use of automatic conversion from the orthography to the phonetic sequence is likely to be necessary for the foreseeable future.

Since resource constraints were more severe in the past, the automatic conversion of graphemes to a phoneme sequence is a well established research area. However, it is not a solved problem, as the review in [1] demonstrates. The best system investigated in the review gives an error rate of 30% for English words. More recent work [6] using statistical methods gives a better result, but there is still scope for improvement.

A complication is the issue of loan words whose pronunciation depends on the language of origin. Since different languages have different mappings from letters to sounds, there is the problem of both identifying the language of an unknown word and of modelling the pronunciation for that language. A further subtlety is whether words of another language origin should be pronounced using the rules and phonemes of that language or in an "un-accented" manner. This is addressed in [2] where Llitjos and Black are interested in approximating the educated, Americanised pronunciation of foreign proper names. The example that they give is of the

American English pronunciation of "Van Gogh" as /V AE1 N . GOW 1/[1]. They justify this approach by claiming that the Americanised pronunciation is more recognisable to people who speak American English. There is some truth in this, but it will not always be the case for words that are unfamiliar. Additionally, the pronunciation itself, if accurately reproduced, also provide some clue as to the origin of the word.

In this study, therefore, we have decided to generate the pronunciation according to the rules of the original language. This makes necessary the construction of models for each language and some means of choosing the model. The approach taken to solving this and the more general problem of grapheme to phoneme conversion is provided in the next section.

1.1 Project Structure

Review

We begin by examining the current methods of grapheme to phoneme conversion using the review in [1] as a guide. The most popular means of translation has been to use context sensitive rules, written by an expert. This approach is now giving way to data driven, and in particular statistical approaches which derive the model automatically from training data. Examples of each of the approaches are given, and the use of Hidden Markov Models is examined in detail, since this kind of model is used in the project.

Baseline System

An aim of this project is to extend a baseline grapheme to phoneme system to work with non-English names and to evaluate the techniques used. The baseline system, based on a statistical approach to pronunciation modelling is implemented. It uses an existing technique [3] of a Hidden Markov Model whose hidden states represent phonemes and which generate graphemes as observations. In this system, the most likely sequence of phonemes for a given orthography is decoded using a Viterbi recogniser. The baseline system is a re-implementation of the system used in [3] and its results are compared with the original version.

1 This sounds like "van go". The British English pronunciation is different, but no more faithful to the original language.

Extension to different languages

The pronunciation model for the baseline system is trained on a lexicon for a specific language (the Unisyn English Lexicon). Words from a different language which are translated using this model will not be pronounced correctly, so the purpose of the first extension is to train models for different languages. To this end, the CELEX lexicons [4] for English, Dutch and German are used to create additional models. The English lexicon is used to provide some comparison with the Unisyn-generated model and also serves as a reference for the German and Dutch models.

Language Identification

Having created pronunciation models for different languages, we model the language of the input word to determine which pronunciation model to use for the grapheme to phoneme conversion. The method of language identification used is based on n-grams of letters. This is the same concept as used for the language[2] model in automatic speech recognition systems, where the output grammar is modelled by short histories of word sequences. For language identification, we examine the letter sequences in each word of the training lexicon and create a model that characterises each language.

Language Identification combined with Grapheme to Phoneme Conversion

The final section examines how to use the output of the language stage with the grapheme to phoneme conversion. A simple approach is to choose the most likely language for a word and use the appropriate pronunciation model to convert the word's graphemes to a phoneme sequence. Other, more elaborate approaches, which combine the results of the two models, are possible and these are investigated.

2 The language model in an ASR system characterises a single language by its word sequences. N-grams are used here to identify one language among many by means of letter sequences.

2 Grapheme to Phoneme Conversion Techniques

It is as well to look at existing and past techniques for grapheme to phoneme conversion so that the method chosen for this project may be placed in context. It has already been mentioned that statistical means of model generation are the most current techniques, but it is helpful to examine how these have developed and provide some explanation of other methods. Damper[1] provides a comprehensive review of letter to sound conversion techniques and this section is based broadly on his explanations. Taylor[3] further analyses these into three broad categories: phonological rules, data driven approaches, and statistical approaches.

Phonological Rules

The first technique is based on the assumption that the pronunciation of a letter or letter substring can be found if enough is known about its context. It uses hand-written phonological rules of the form

$$A[B]C \rightarrow d$$

This states that the letter substring B, in the context of A and C, rewrites as phoneme d. Since the mapping from orthography to sound is complex, especially for English, more than one rule is typically needed for the transformation. As described in [1], the conflicts that occur between rules as they are applied are resolved by keeping them in a set of sublists grouped by initial letter and ordered by the specificity of the rule. The most specific rule is placed at the top with more general rules towards the bottom. During transformation, the sublists are searched in a left to right process; for each letter, the appropriate sublist is searched from top to bottom until a match is found. Damper reports that the phonological rules perform badly compared with more recent methods, citing a figure of 25.7% of words with correct pronunciations. In particular he found insertion errors, where the generated phoneme string is longer than the reference in the lexicon, confusions of the phoneme labels /g/ and /j/, and confusions between vowels.

Data Driven Approaches

The second approach uses a data driven model whereby the mapping from grapheme to phoneme is constructed automatically by processing a training set. This, along with statistical approaches falls into the general category of machine learning techniques. Taylor points out that the operation of many data driven approaches is no different from the phonological rules; the questions stored in a decision tree could be reformulated as context-sensitive rules.

Damper describes a data driven technique called "Pronunciation by Analogy". In this method, a directed graph is built from substrings in the input word and their partial pronunciations.

A letter to phoneme alignment is created automatically from the lexical database. The input word is then scanned for substrings that match the alignment database, and a *pronunciation lattice* is created. The decision function finds a possible pronunciation for the word by traversing the lattice and concatenating the phoneme labels on the arcs. The path chosen is shortest, and in the case of a tie, a heuristic is used.

Damper quotes a best figure of 71.8% words correct on an implementation which featured several enhancements compared with the original PRONOUNCE system. This result was based on a different phoneme set than for the rules-based system described in the preceding section, so it cannot be compared directly, but the difference from 25.7% for the rules-based approach is marked.

Statistical Approaches

A final category of technique is also data-driven, but uses a statistical method to train a model from the data. One such method is described by Chen[6] who uses a model which is similar to that used in machine translation. The problem is framed as follows: given a letter[3] sequence G, find the phoneme sequence S that maximises $P(S \mid G)$

$$\hat{P} = \arg\max_{s} P(S \mid G) = \arg\max_{s} P(G, S) \qquad (1)$$

The model uses vocabulary of "chunks" c_i to estimate the joint distribution $P(G, S)$, where a chunk is a letter, a phone or a letter-phone pair. Then,

$$P(G, S) = \sum_{C:\ letters(C) = G,\ phones(C) = S} P(C) \qquad (2)$$

$$P(C = c_1 \ldots c_m) = \prod_{i=1}^{m} P (c_1 \ldots c_{i-1}) \qquad (3)$$

The term on the right hand side of equation (3) is estimated using an n-gram model, which is trained using an Expectation-Maximisation algorithm. Chen claims that

3 Chen uses the term 'letter'. Throughout this report, we use the more general 'grapheme'

word error rates are significantly lower using this model compared with any of the rule-based approaches.

Another kind of statistical approach uses Hidden Markov Models to determine the phoneme sequence from the grapheme sequence. Jelinek[7] uses HMMs to align the grapheme and phoneme sequences, but he uses a decision tree as the classifier. Taylor's work [3] uses HMMs for both alignment and decoding of the input sequence. The method is described in more detail in the next section.

3 Using Hidden Markov Models for Grapheme to Phoneme Conversion

For automatic speech recognition, HMMs are used to model units (words or phones), and the process of recognition involves finding the most likely sequence of units for a given sequence of observation features. A model prototype is shown in Figure 1. The prototype is used as a template for creating models for each of the individual units to be recognised by the system.

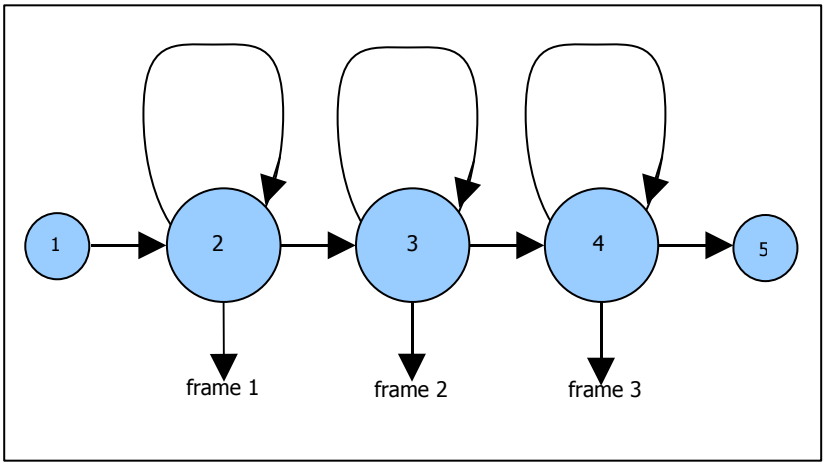

Figure 1. HMM Topology for Automatic Speech Recognition

It is helpful to briefly review why HMMs are appropriate for automatic speech recognition before considering their applicability to grapheme to phoneme conversion. Speech can be segmented into smaller units, phones, which describe the original signal and from which words can be derived. The signal can also be divided into frames and processed into quasi-stationary segments which can be characterised by a series of feature vectors.

The feature vectors form an input stream of *observations*, and to find the most likely sequence of words represented by the stream, we use the standard equation:

$$\hat{W} = \arg\max_{w} p(\mathbf{O}|W)\, P(W) \qquad (6)$$

where $\hat{W}$ is the most likely sentence and $\mathbf{O}$ is the stream of observations. The term $p(\mathbf{O}|W)$ is called the acoustic model and $P(W)$ is the language model. The acoustic model is trained using the Baum-Welch algorithm and the language model trained by analysis of a sample corpus of text[4]. The model makes two assumptions: the

4 More detail on n-gram models is given later in the report, where they are used for language identification.

Markov assumption is that a state depends only on its predecessor and the *independence assumption* is that the observations are independent of one another. For speech, this latter assumption is clearly broken since it is hard to create streams of observation vectors that are both independent and which faithfully represent the input signal.

Model Design for Grapheme to Phoneme Conversion

The purpose of automatic speech recognition is usually to convert a speech signal to a sequence of words. For the grapheme to phoneme problem, the problem is to convert a sequence of graphemes to a sequence of phonemes and the solution is analogous. The HMM states in this case represent the hidden sequence of phonemes S,

$$S = < s_1, s_2, ..., s_M > \qquad (7)$$

while the observations are now graphemes, the analogue of speech vectors.

$$G = < g_1, g_2, .. g_N > \qquad (8)$$

The question of why we choose this formulation arises. Taylor[3] describes the phoneme sequence as having the underlying property of the word, while the grapheme sequence is seen as being generated by the phoneme sequence. For example, the phonetic sequence /p l au z/ can be thought of as generating the word "ploughs". It can be seen that some phonemes map one-to-one with graphemes, such as /p/→p while others map one-to-many, like /au/ → ough. In English, there are very few examples of many-to-one mappings where more than one phoneme generates a single letter; one such case is /k s/ which generates x.

So the phonemes are represented by the hidden states because they generate one or more graphemes. It would be more difficult to use a model in which the graphemes are the states and the phonemes the observations, because the states would frequently generate no observation. In the formulation as given, with some correction for the many-to-one mappings like /k s/ → x, the phonemes always generate graphemes.

Model Topology

Having chosen the hidden states and the observation vectors, it is necessary to design an appropriate topology for the model prototype. The design parameters include the number of states and their connectivity. In automatic speech recognition, a common topology is illustrated in Figure 1 above. There are three emitting states, each of which has a loop back to itself. This allows the modelling of time-dependent characteristics of the speech signal by allowing a given state to repeatedly generate frames. Is this an appropriate configuration for the grapheme to phoneme prototype

model? There is no equivalent to time-compressibility with graphemes because graphemes are not themselves dynamic time-dependent, unlike speech, so this would suggest not.

Considering the example mapping given earlier /p l au z/ → "ploughs", we can identify other possible outputs from the phoneme labelled /au/ as follows

/au/ → ough
/au/ → ou
/au/ → ow
/au/ → au

From this is can be seen that the first state of the model is likely to generate a letter "o", and the second, the letter "u". There is the question of how to generate sequences of varying length; in the example shown, the phoneme label can generate up to four letters. A solution is shown in Figure 2, where there is an arc from each state to the final, non-emitting state. The observation sequence for the substring "ough" is shown.

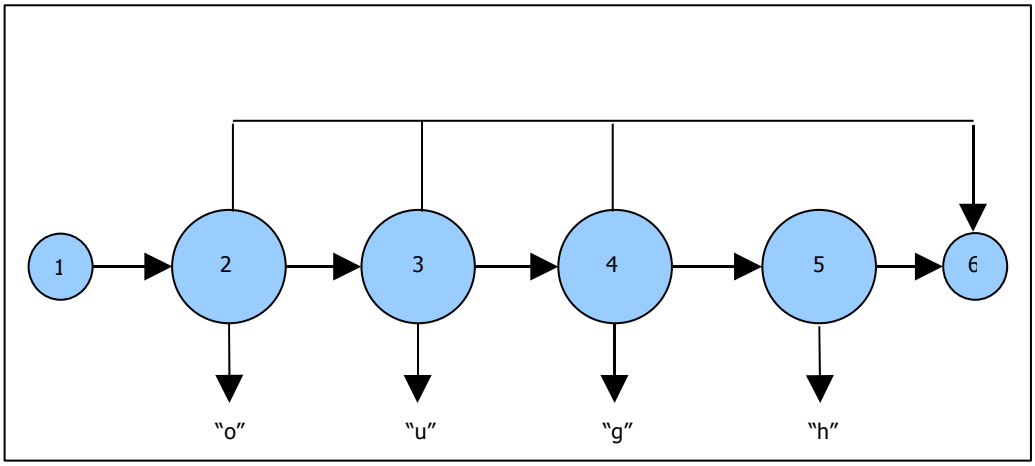

Figure 2. HMM Topology for Grapheme to Phoneme Conversion

It can be seen that this model, with suitably defined observation probabilities, can generate all the letter sequences for the phoneme /au/. For sequences of two letters the path taken is *1236*, while for a four letter sequence, the entire graph is traversed. The model also shows that, while the output probabilities for automatic speech recognition are usually continuous density distributions, for this application, they are discrete quantities. It is instructive to define the HMM for the phoneme /au/, assuming that the mapping is as shown in the example given above.

/au/	***State***				
	2	**3**	**4**	**5**	**6**
1	1				
2		1			
3			¼		¾
4				1	
5					1

Figure 3. Transition Matrix for Phoneme /au/

Figure 3 shows the transition matrix for the example. Since there are always at least two letters generated by the phoneme, the probability of traversing the arc from state 2 to state 3 is unity. The probability for moving from state 3 to state 4 is ¼, while it is ¾ from state 3 to state 6, since there are three times as many two letter sequences as four letter sequences. Figure 4 shows the observation probabilities for each letter.

/au/	***State***				
Letter	**2**	**3**	**4**	**5**	**6**
a	¼				
g			1		
h				1	
o	¾				
u		¾			
w		¼			

Figure 4. Observation Probabilities for Phoneme /au/

Since three of the four letter sequences start with the letter “o”, this has the probability of ¾ in the first emitting state (2) while the letter “a” has the probability ¼. Similarly, “u” and “w” are in the proportions 3:1 for state 3. States 4 and 5 always generate a single letter as output and so each letter in these states has a probability of unity.

Model Training and Decoding

The example just given is a simple explanation of the hand derivation of HMM transition and observation probabilities. To be useful as a model, these parameters have to be able to be derived from a training corpus, and to do this, the Baum-Welch algorithm is used. This results in a probability for each letter being generated by a given state, and in the transition probabilities. Using Baum-Welch training, the graphemes are *aligned* with the phonemes; that is the graphemes that are more likely to be generated by a given state are given higher probabilities than those that are not likely to be generated. Clearly, since only a few letters tend to be associated with each phoneme, most graphemes will have almost zero probabilities for most states.

To derive the phoneme sequence from the graphemes, we use the standard equation:

$$\hat{S} = \underset{S}{arg\,max}\ P(G|S)\,P(S) \qquad (9)$$

where $\hat{S}$ is the optimal sequence of phonemes, *P(S)* is the *prior* probability of a sequence of phonemes and *P(G|S)* is the *likelihood* of a grapheme sequence given the phoneme sequence.

The first element of equation (9) is the conditional probability *P(G|S)* which is derived from the Viterbi algorithm which generates the probability of the observation sequence from each of the trained models. This component is equivalent to the acoustic model used in speech recognition. The analogue of the language model is the *phoneme model P(S)*. This can be represented either as an n-gram or using a *phonotactic grammar*, which constrains the sequence according to a set of rules.

A phonotactic grammar is the equivalent of a grammar for a natural language except that the units are phonemes. The grammar specifies legal sequences of phonemes, so for example the sequence /s k r/ at the start of a word is allowed but /s g r/ is not (the example is from [3]). Taylor identifies a particular advantage of this kind of grammar for use with text to speech applications. There is a need with TTS systems to generate phoneme sequences that are correct, because otherwise the system will not be able to generate a waveform from it. This is fairly obvious in that not all phoneme sequences can physically be created by a speaker, especially if they require two simultaneous and incompatible noise sources from different parts of the vocal tract. This is a disadvantage with the decision tree approach, which can result in phonotactically illegal sequences.

However, despite their advantage in generating only legal sequences, phonotactic grammars are not trivial to create and they have a dependence on the phoneme labels used. For this reason, the phoneme model used in this project is based on n-grams.

Summary of HMM Properties and Design

A possible model topology for grapheme to phoneme conversion has been discussed, and a model design developed, based on the use of hidden states representing phonemes which generate graphemes as observations. The model topology is different from that used in automatic speech recognition in that there are no looping states, but instead all the states have an arc to the final, non-emitting state. The observation probabilities are represented as a discrete quantity, since the observations are letters. This contrasts with automatic speech recognition, where the output probabilities are represented by Gaussian mixtures. The training and decoding of the model are similar to these processes in ASR.

Taylor identifies a number of advantages to the use of HMMs as a model for grapheme to phoneme conversion. While data driven techniques rely on a separate model to align graphemes to phonemes, for HMM-based models, alignment and decoding can be performed on the same model. HMMs are flexible in that a topology can be chosen to optimise the model for the problem domain, and the powerful Baum-Welch training algorithm can be used to optimise the model parameters. The next section now describes an implementation of a baseline grapheme to phoneme system based on the topology that has been described.

4 A Baseline Grapheme to Phoneme System

The purpose of the first set of experiments is to reproduce some of the results reported in [3] and to provide a baseline system by which developments can be assessed. The system in this original paper used several kinds of phoneme model, from unigram to 4-gram and it also featured enhancements such as a preprocessing step before the conversion. This reproduction uses only bigram and 4-gram models, and does not include any preprocessing.

The HMM-based system for grapheme to phoneme mapping was constructed using the Unisyn English pronunciation lexicon divided into training, evaluation and development sets in the proportions 85:10:5. The system uses some of the same scripts as the original research, but all the models were reconstructed and trained using the lexicon. The Hidden Markov Toolkit[8] versions 3.1 and 3.2 (for the language tools) were used for development.

Data Preparation

The first stage in creating the system is the preparation of data, since this is necessary for both the creation and training of the model. The preparation of the lexicon is one of the more time-consuming stages of creating a system since any pronunciation lexicon usually has to be converted into a form suitable for use. This processing typically involves a standard transformation specified by the lexicon authors and any conversion necessary to the project, such as changing letters to lower-case and removing single letter words. For the baseline system, the processed lexicon used in [3] was reused without alteration. This lexicon uses a set of 42 phonemes, which will be termed the baseline set. Each line in the lexicon is given a number so that each word has a unique identifier.

Since the Hidden Markov Toolkit is designed for speech input, the lexicon was transformed into binary "speech" files for use with the kit. This was accomplished using a script to write out an appropriate header, including a code for discrete parameters, followed by an offset ASCII value for each letter of the word. The training and evaluation data each consist of a directory containing one binary file per word, the file name being the index number in the lexicon.

Model Creation

Using UNIX command line tools and scripts, the set of unique phoneme labels was derived from the lexicon and this list used to create an HMM for each label. The scripts are listed in Appendix 2. The HTK tool HInit was used to create the HMM files from a prototype definition which defined three emitting states. This tool also provides initial values for parameters by analysis of the training set.

The phoneme model (the analogue of the language model in automatic speech recognition) was created by reprocessing the training lexicon into "sentences" consisting of a phoneme sequence per lexicon entry. Start and end tokens were added to the lexicon for this purpose. Both bigram and 4-gram models were created.

Training

After the models were created, they were trained using iterations of the Baum-Welch algorithm by running HTK tool HERest. Five training passes were tried, and the recognition accuracy measured after each run. To illustrate the process, Figures 5 and 6 show the resulting parameters for the phoneme labelled /au/. These values are actually drawn from the original research. The raw data is given in Appendix 3.

/au/	***State***				
	2	**3**	**4**	**5**	**6**
1	1				
2		0.96			0.04
3			0.02		0.98
4				0.52	0.48
5					1

Figure 5. Actual Transition Matrix for Phoneme /au/

Figure 5 bears a close similarity to Figure 3. It is different in that state 3 is very likely to transition to state 6, implying that the great majority of strings generated by this phoneme are no longer than two letters. Unsurprisingly, there are some three letter examples, and the values of 0.52 and 0.48 show that there are as many of these as four letter examples.

Figure 6 shows the observation probabilities derived from the real system. As in the constructed example, “a” and “o” are the letters most likely to be generated by the first emitting state, but in the real case, “o” is very much more likely. Similarly, the letters “u” and “w” are most likely to be generated by State 3, but in the actual model, “u” is very much more likely. It can be seen from both the observation and transition probabilities that the model is most likely to generate the sequence “ou”. A count[5] of sequences in the training lexicon reveals that of 1984 out of the 2753 examples of /au/ are the letter string “ou”.

As in the example, State 5 always generates the letter “h”. State 4 is most likely to

5 `grep " au " full_train.nlex | grep -c "ou"`

generate “g” but surprisingly it has a 40% chance of generating the letter “b”. An examination of the lexicon shows where this comes from. The word “doubt” is transcribed /d au t/, and the “b” is silent in this case.

/au/	***State***				
Letter	**2**	**3**	**4**	**5**	**6**
a	0.02				
g			0.52		
h				1	
o	0.97				
u		0.69			
w		0.30			
b			0.40		
e			0.06		

Figure 6. Actual Observation Probabilities for Phoneme /au/

Recognition

The recogniser was configured to generate a lattice as output, using 5 tokens and 20 hypotheses. The bigram model was used during recognition as a lattice input to the recogniser. After the recognition had completed, the 4-gram phoneme model was used to re-score the result lattice. The results were then analysed using the HTK tool HResults to compare the output of the recogniser and re-scored results with a reference file holding the correct phoneme sequences.

4.1 Experiments

Evaluation Measures

In each of the experiments, the *Phoneme Accuracy* records the number of phonemes correctly identified during recognition, with insertion, deletion and substitution errors rated equally. The *Words Correct* measures the number of words which have an exact match for the phoneme sequence in the reference file. It measures the quality of the transformation, while the phoneme accuracy provides further information on sequences that are partially correct.

Results

Figure 7 shows the recognition results for a Unisyn lexicon test set and using a three emitting state HMM with a bigram phoneme model. Both the *Words Correct* and *Phoneme Accuracy* values peak at the third pass of E-M training. Additional passes of training do not significantly improve or reduce accuracy.

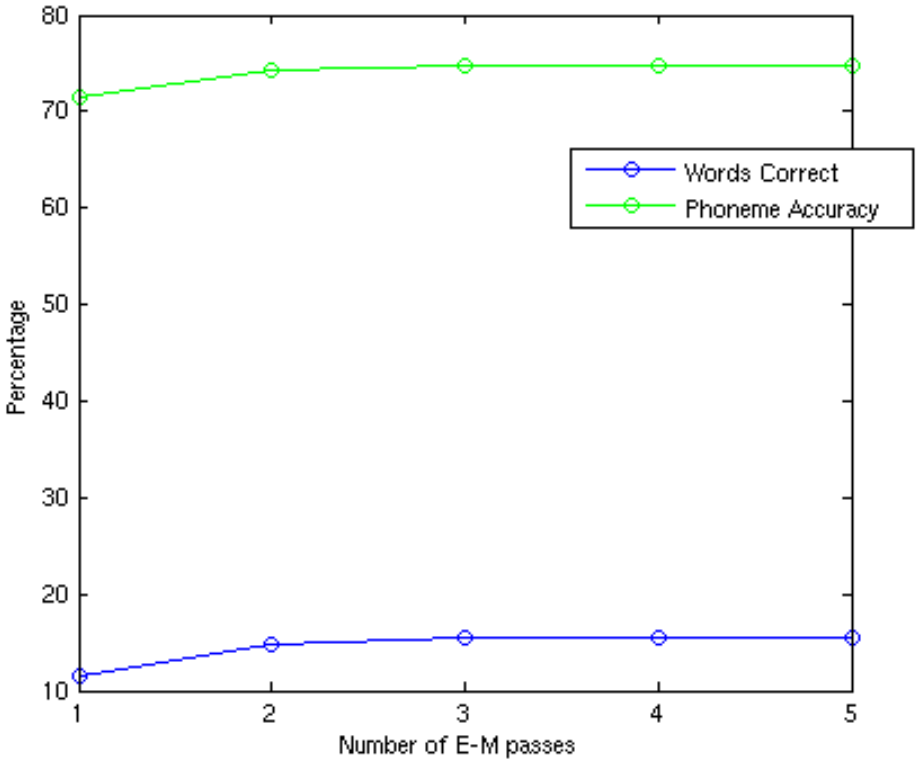

Figure 7. Recognition Accuracy vs Number of E-M Passes

A second experiment using four states for the model was trained using three E-M passes. Table 1 shows the results compared with the original research [3], which this experiment is attempting to reproduce.

	Bigram	***4-gram***
Unisyn[3]	15.56 (73.12)	39.13 (85.21)
Unisyn	16.23 (75.05)	41.63 (83.87)

Table 1. Words Correct (Phoneme Accuracy) for the Unisyn Lexicon

It can be seen that the experiment slightly outperforms the original values. The reasons are not readily apparent because the setup cannot be replicated without trying many combinations of parameter. Possible differences might be the number of training passes and the number of states used for the model prototype.

Error Analysis

A comparison of the recognised words with the lexicon shows that the largest class of error is the confusion of /ax/ (schwa) and vowels. For example, the word “kit” /k i t/ is recognised as /k ax t/. Inspection of the confusion matrix, which plots correct symbol versus recognised symbol, confirms that /ax/ is confused with vowels, and is subject to insertion and deletion errors.

Figure 8 shows the classes for which /ax/ are confused. This graph is derived from the output from the recogniser using a 4-gram phoneme model. It clearly shows the misclassified results peaking around the vowels, with i-o-e-a-ii in order of severity. The reason for these errors is that vowels are often pronounced as schwa, and the model cannot distinguish the cases when they are not. Taylor deals with this problem by adding a syllable level HMM to model the stress patterns of words. This and the other enhancements are described later.

Another class of error arises from the complexity of English spelling. The word “elate” has a lexicon transcription of /i l ei t/ but is recognised as /i l ax t i/. This is unsurprising because a phoneme will be sought by the recogniser to generate the final “e” in the word. In English, however, not only is the “e” silent but its presence changes the pronunciation of the “a”. The model cannot represent this dependency between observations.

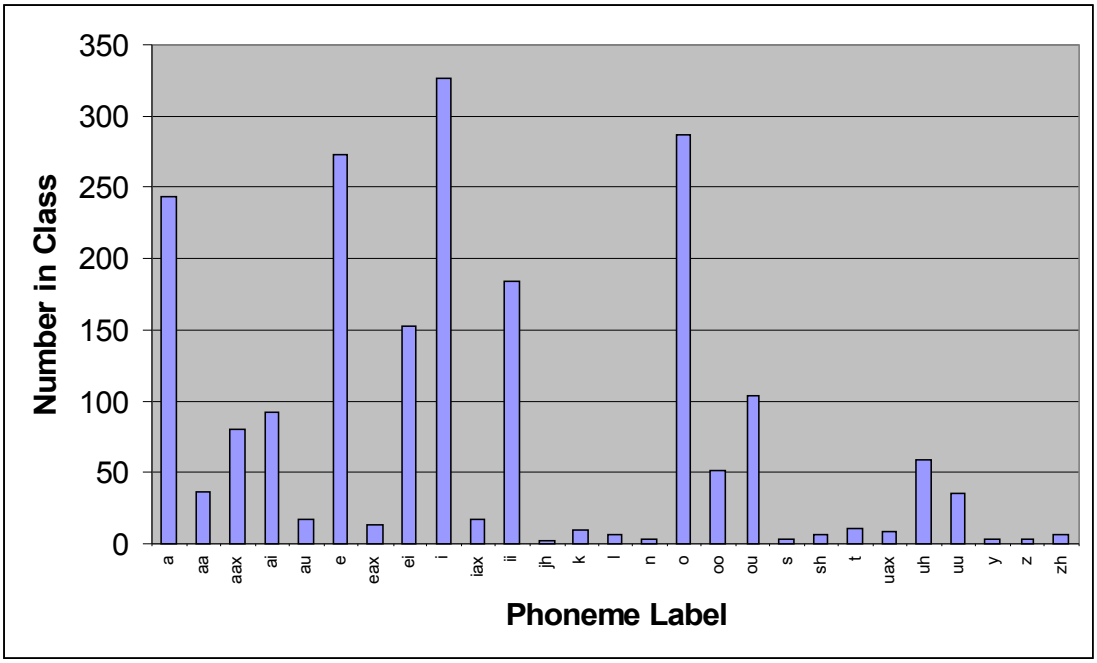

Figure 8. Confusion classes for Phoneme /ax/

Evaluation

The closeness of the reproduced results to the original shows that the baseline system has broadly reproduced the original results. The discrepancies are likely to be due to differences in experimental parameters, such as the number of training passes which have a relatively minor effect.

An advantage in using an HMM for grapheme to phoneme conversion is that a single model is used for both the alignment and transformation stages, whereas other methods have relied on pre-aligned data. However, the HMM approach does have limitations, many arising from the complexity of the grapheme to phoneme relationship. An example is the effect of a silent "e" mentioned in the previous section. Since the dependency is in the grapheme rather than the phoneme domain it cannot be modeled using an HMM; it represents a dependency between observations. Taylor[3] solves this problem by deterministic preprocessing to swap letter positions so that, in the example above "elate" becomes "elaet"[6].

Two other enhancements are described in [3]. The HMM does not explicitly model letter context, as rule-based systems do. For example, whether the letter "c" is

6 This simple rule does not allow for words like "material"

generated by a hard /k/ phoneme or a soft /s/ phoneme depends on its context; it is usually soft when the letter is followed by “e” and hard when followed by “o”. Assuming that these effects are mirrored in the phoneme sequence, it is straightforward to train additional HMMs to represent them.

A third enhancement is an adjustment for stress, which cannot adequately be modelled using a 4-gram because the stressed syllables are too far separated within words. This can be solved by creating an explicit stress HMM to predict stress patterns, and comparing the result with the phoneme sequence. As mentioned earlier, this goes some way to solving the misclassification of schwa.

The result of about 40% words correct for English is not adequate for a real system, but the enhancements described improve the figure to 61.08% words correct and 92.28% phonemes correct [3]. There are other variables that affect the result. The quality of the lexicon, the phoneme set and the lexicon post-processing are likely to have a significant effect on accuracy. The language will also have a major influence, because some languages have a closer relationship between spelling and sounds than does English. The next section examines an extension to the baseline system using lexicons in other languages as well as a different English lexicon.

5 Extension to Multiple Languages

Having reproduced the results for the English Unisyn lexicon, we now look at extending the baseline system to work with non-English words. The purpose here is to train and evaluate a set of models using a language other than English and to use another English lexicon. Lexicons in as many languages as possible were sought, but for reasons of time and efficiency only three were used in the project.

The resources used were CELEX pronunciation lexicons for English, French and German. These were processed into SAMPA notation using the scripts based on the CELEX documentation. SAMPA is an ASCII representation of IPA which is used for phonemic transcription and it also has some limited support for prosody. The transcription was converted to the phonemic representation used in the baseline system and its prosodic markings removed. The mapping from SAMPA to the baseline set is given in Appendix 1. Each lexicon was then cleaned by removing single letter entries, words with hyphens or underscores, and lines that became duplicated following the phoneme mapping. The lexicon was then divided into training, evaluation and development sets in the ratio 85:10:5.

The experimental setup was the same as in the baseline system. Three iterations of Baum-Welch embedded training were applied to an HMM with three emitting states per phoneme using the training set for each lexicon. A recogniser was then run on the test set and the results analysed. The experiment was performed for each of the CELEX lexicons and also on a multi-language lexicon which was created by concatenating the three lexicons. A single model, using all the phoneme labels in the three lexicons was created for the combined lexicon. The results using bigram and 4-gram phoneme models are shown in Table 2.

	Bigram	***4-gram***
English	20.97 (75.70)	40.89 (82.74)
German	39.27 (90.44)	78.55 (97.08)
Dutch	44.40 (89.87)	74.48 (95.38)
Multi language	18.74 (80.68)	47.23 (88.25)

Table 2. Words Correct % (Phoneme Accuracy %) for CELEX Lexicons

Error Analysis and Evaluation

The result for English at 20.97% for the bigram model is better than that for the Unisyn lexicon at 16.23%. Inspection reveals that whereas in the Unisyn-derived lexicon schwa is commonly used, a more specific phoneme is often used in the

CELEX-derived lexicon. For example:

Unisyn	abdicated	a b d ax k ei t ax d
CELEX	abdicated	a b d i k ei t i d

A further example is where schwa is used for words with the suffix "-ing" in Unisyn, where /i/ is used in CELEX. Since, in the baseline system, schwa confusion accounts for the largest class of error, then the use of more specific phonemes might be expected to result in a reduction of errors. However, the confusion matrix still shows /ax/ as generating most of the errors as a result of insertion, deletion and confusion with vowels. The phonemes which are misclassified as schwa are shown in Figure 9. This result is for the 4-gram English CELEX model.

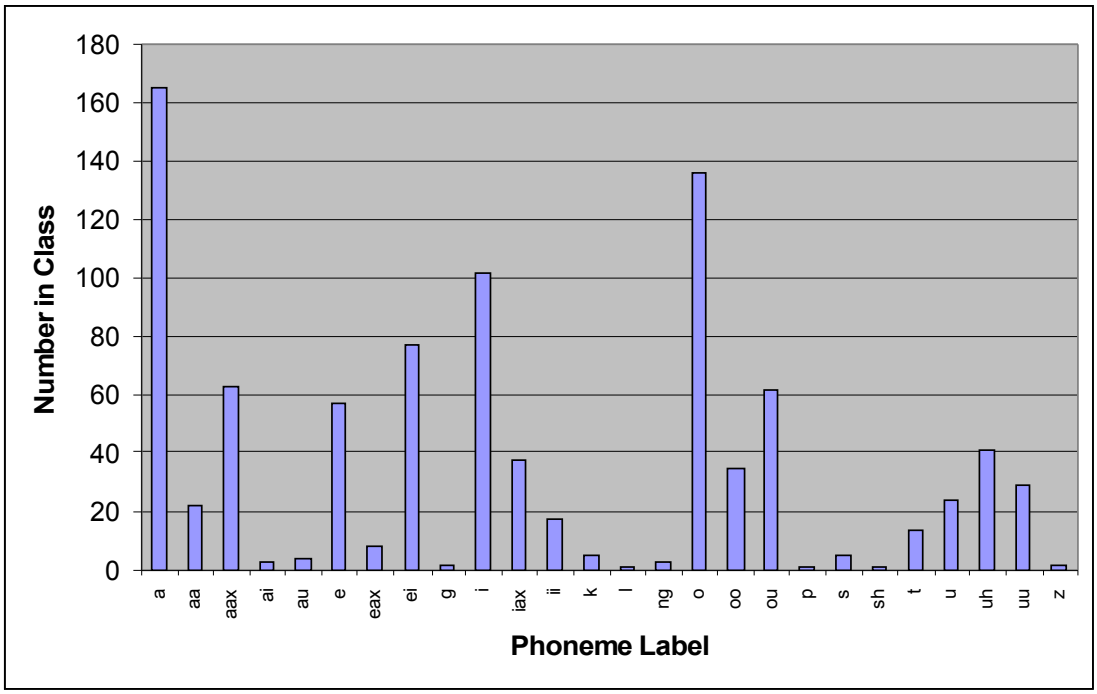

Figure 9. Confusion classes for Phoneme /ax/ in CELEX English

Again, Figure 9 shows that it is the vowels that tend to be confused with schwa. The pattern is different from that for the baseline system, shown in Figure 8. In that case, the errors were i-o-e-a-ii in order of severity. For the CELEX model, it is a-o-i-ei-aax. This difference might be attributable to the use of schwa for the suffix "-ing" and other differences between the lexicons.

The 4-gram model result at 40.89% is close to the Unisyn result of 41.63%. The closeness to the original result, unlike the case with the bigram, may be because the 4-gram model corrects errors that are manifest when using the more approximate bigram phoneme model.

The results for German and Dutch are approximately twice the accuracy of those for English. This is likely to be due to the more regular correspondence between pronunciation and orthographic form for these languages compared with English. An anomaly is that bigram model performs better for Dutch than German, while the 4-gram model performs better for German than Dutch. Both bigram and 4-gram models are different approximations to the likelihood of a sequence of phonemes, so this suggests that Dutch relies marginally less on an accurate prediction than German.

These results from the CELEX lexicons have confirmed the results from the baseline system for English and shown that the conversion from grapheme to phoneme sequences is more successful for languages other than English. This is to be expected since written English is less "phonetic" than written German or Dutch; that is, it has a less regular correspondence between spelling and pronunciation.

The multi-language model gives a 4-gram result of 47.23% words correct, which is higher than the 40.89% for English but much lower than the results for German or Dutch. The important characteristic of this model is that it is trained on all three languages, and so can potentially generate a phoneme sequence corresponding to the wrong language for the input word. However, the phoneme n-gram model does constrain phoneme sequences and in effect this acts as a hidden language identifier. This can be seen to some extent in comparing the bigram and 4-gram results for the English and multi-language models. Whereas the multi-language bigram model is actually worse than English, implying that at least some English words are being generated by German or Dutch phoneme sub-sequences, the 4-gram model is better, showing that the n-gram model is improving recognition beyond the English baseline. So some implicit language identification is taking place, as a result of the language dependence of the phoneme sequences. This is discussed further in the Section 6.

5.1 Conversion of English Place Names

Many of the words that are missing from the lexicon and in need of conversion will be names, since this open class is the one most difficult to maintain. Ideally the project would have investigated the accuracy of name conversion as a separate set, but there were difficulties in obtaining data. The CELEX lexicons do not contain enough proper nouns for an evaluation, and no other name sets with a convenient phonemic transcription were found in the time available. Instead, some of the English data used in Section 6 were converted using the CELEX English model to allow a qualitative evaluation of the result for English.

Experimental Procedure

The steps involved in processing the English GEONet database are described in Section 6. The resulting lexicon is a database which excludes Welsh names[7], compound or hyphenated words, and with all words in lower case. To limit the size of data, we selected one in every 300 of the database as an evaluation set.

Using the model that had been trained on the CELEX English database, the evaluation set was converted to a phonemic representation using the recogniser with CELEX-derived bigram model. The resulting lattice was then re-scored using a 4-gram phoneme model to generate a 1-best sequence. The results for each word are shown in Table 3.

Error Analysis

The results show that while many transcriptions are correct, there are some errors. In "chatham" the first two letters are rendered as /k/, and the "th" incorrectly as /th/. With "cosham" the initial letter is incorrectly converted to a soft "c" represented by phoneme /s/. This is an example of where the n-gram phoneme model is insufficient to predict such dependencies; the letter "c" is almost invariably rendered as a /k/ when it is followed by an "o". Other errors, such as schwa confusion and the pronunciation of a silent "e" have already been noted.

Some of the errors highlight a particular problem with place names. The suffix -ham is often pronounced as a unit, so the /th/ in the transcription for "chatham" is incorrect, although it is an unsurprising error . The following word "cosham" is an exception to this rule, since /sh/ here is correct. So without special knowledge, unique to the domain, there is no way of predicting the correct pronunciation from the letters.

7 Simply because the model was trained on English.

Name	*Phonemic Representation*
abberley	a b ax l i
azerley	a z ax l i
bingley	b i ng l i
brinsley	b r i n s l i
chatham	k a th ax m
cosham	s ou sh ax m
dover	d ou v ax r
evington	e v i ng sh ax n
garlinge	g oo l i ng z
harbledown	h aa b l ii d ou n
histon	h i s t ou n
keld	k ax l d
letchworth	l ii ch w aax th
medbourne	m e d b oo n
northorpe	n oo dh ax p i
pointon	p oi n t ax n
rothbury	r o b ax r i
shepherdswell	sh i p y aa d s w e l
stanhope	s t a n h ou p i
thatcham	ch a t k ei m
wadsley	w oo d s l i
wibsey	w i b ax l i
wyton	w i t ax n

Table 3. Place name phonemic representation using the CELEX-trained model.

6 Language Identification

6.1 Implicit Language Identification

The previous section showed that some language identification occurs with a model that is comprised of a number of languages because the phoneme sequences will be characteristic of a given language. A question that arises from this is how effective the combined model is at recognising the language without any explicit language identification. To answer this, we tagged the phonemes in the combined lexicon with a prefix denoting the language of origin. Thus the English word "hello" would have a corresponding phoneme sequence / en_h en_ax en_l en_ou /. The tags act as a marker for the language in the output lattice that results from the recognition. We re-ran the experiment to recognise the multi-lingual test set and analysed the results.

For the analysis, we grouped each phoneme into an equivalence class for its language so that the recognised phonemes were mapped to one class per language. Phoneme strings composed only of phonemes tagged with the correct language were counted as correct for the purposes of language identification. The results are shown in Table 4.

	Bigram	***4-gram***
%Correct	77.92	86.03

Table 4. Language identification using a multi-language model

The results show that the language for most words is identified correctly, even when only a bigram model is used. The failures of recognition are those where an optimum path through the HMM to generate the word is found to be a foreign phoneme sequence. This can happen when, for example, a grapheme to phoneme mapping is consistent for one language, but not another. The training ensures that the more consistent mapping will dominate. This kind of mis-recognition is investigated further in the next section.

6.1 Explicit Language Identification using Letter n-Grams

A pronunciation model trained on several languages has been shown, unsurprisingly, to have some implicit identification of language. We now investigate the performance of a model which identifies the language explicitly. The first stage in

doing this is to develop a model which is based on letter n-grams.

In the field of automatic speech recognition, n-grams are used to model the probability of a word sequence occurring. For this application, the observed sequences are of letters, rather than words, and the n-gram model has so far been applied to phonemes. We now apply the same principle to letters.

The probability of a word W with letters $(w_1, w_2,..w_n)$ in a given language c is

$$P_c(W) = P(w_0, w_1, .., w_n, w_{n+1}) \quad (10)$$

In this equation, w_0 and w_{n+1} denote start and end markers respectively. Equation (10), which expresses a joint probability, can be restated as a product of conditional probabilities of letters:

$$P_c(W) = P(w_1 \mid w_0) \,.\, P(w_2 \mid w_0, w_1) \,..\, P(w_{n+1} \mid w_0, w_1,.. w_n) \quad (11)$$

Language identification for an unknown word is accomplished by finding the most likely language given the letter sequence of the word:

$$\hat{C} = arg\ max_c\ P_c(W) \quad (12)$$

So, the class chosen is the one with the maximum posterior probability.

Since (10) cannot efficiently be computed, owing to the number permutations of a long sequence of letters, the letter history length is limited to four. In this way longer sequences are placed in an equivalence class which includes all the histories of sequences longer than four.

The problem is now one of estimating the conditional probabilities of letter sequences. This is done by dividing the number of occurrences of the sequence by the count of the sequence it is conditioned on:

$$\hat{P}\ (w_3 \mid w_1, w_2) = \frac{N(w1\,, w2\,, w3)}{N(w1\,, w2)} \quad (13)$$

Where the count is small, this approximation is inaccurate, and so a discounting procedure is used. The method used was Good-Turing discounting, where it is assumed that the probability mass for all events not seen in the training data is the same as the total for events seen once. Following Katz, this is applied only to a range of counts below which the event is discounted to zero and above which no discounting is applied.

6.2 Implementation of a Letter N-Gram Language Identifier

A 4-gram letter model of the kind just described was implemented for each of three languages: English Dutch and German. To do this, the cleaned CELEX lexicons were used and the phonemic data excluded to leave just the words. As before, the lexicons were divided into training, evaluation and test data in proportions 85:10:5 respectively. The language models were created using Stolcke's SRILM toolkit [9], with the default discounting range of 2 to 7. The implementation was made in the Python scripting language.

For each word in the test set, the probability $P_c(W)$ was estimated using the letter n-gram models for English, German and Dutch, and the word tagged according to the most probable language. The confusion matrix is shown in Table 5.

	Number of Words in Assigned Class			
	English	***German***	***Dutch***	***% Correct***
English	4907	633	992	75.12
German	488	28655	1709	92.88
Dutch	772	1790	27150	91.34

Total correctly classified: 90.48%

Table 5. Letter N-gram classification of test set by language

The result shows that the classifier functions with a word error rate of 25% for English and less than 10% for German and Dutch. Examination of the words that were confused reveals that for English words classified as German, some have a German origin, or are derived from another language. Examples include: “glockenspiel”, “liechtensteiner”, and “mezzo”. Other words are English in origin, but have a form which might be confused with German, as for example those with the letter pairs "au" or "er", which are common in German. (A count shows that the pair “au” is about ten times as common in German words compared with English, while “er” is about twice as common.)

Confusions with Dutch result less commonly from loan words. The reasons for misclassification of Dutch words are less apparent than with German, though many examples have repeated letters such as "tt" or "oo". Most German words that are classified as English are in fact English loan words recorded in the German lexicon; some are French. This is also the case for Dutch words classified as English, but there are fewer loan words in this case. It should be noted that the confusion of loan words is an artifact of the test, rather than a problem with the model, since the task is to find the word origin, not the lexicon it was from. Ideally, the loan words would

have been removed from the lexicon, but this is a manual process and given the sizes of the lexicons, there was not enough time for it.

The fact that French words in the German lexicon are misclassified as English exposes a weakness with the implementation. Since there was no floor placed on the probability, the nearest language was used, rather than an "unknown" result being given. In fact, *unknown* might be a more informative classification for the output of the rather than using a misleading nearest class. In general, however, the use of letter n-grams as a means of estimating the language has been successful, and could be used as part of a language identification system.

6.3 Second implementation using HMMs

A second implementation of the language classifier was made using the language modelling component of the Hidden Markov Toolkit. In this case, the 4-gram letter model as used in the previous section was run with a "null" HMM. The null HMM is a model with a topology of one emitting state per letter per language. Its observed output always matches the name of the model. For example all the phoneme models *en_a*, *ge_a* and *du_a* always generate the letter "a".

The recogniser was run on the general test set, and the recognised language-letter sequences were compared with the original lexicon. Since it is the language, and not the letter sequences that is of interest, the letters were mapped to an equivalence class for each language.

The result was a figure of 91.98% correct language classification. This is close to the figure of 90.48% in the Python implementation, and inspection of confused words shows that they are broadly the same as before. The exact reason for the minor difference in performance is not obvious, but nor is it surprising given the different means of implementation.

6.4 Language Identification of Place Names

The evaluation includes few proper nouns since these are not generally included in the CELEX lexicon, with some exceptions. Yet since the main application of grapheme to phoneme conversion is likely to apply to names, it seems wise to test the effectiveness of the method on a test set composed entirely of names.

Lists of names in various languages are widely available, but they need to be sourced with care. After some exploration, we decided to use the GEONet names[10], which are available from the US National Geospatial Intelligence Agency. The GEONet names server provides lists of geographic names, downloadable by country. The

database includes latitude, longitude and the type of feature described by the name.

The databases for England, Germany and The Netherlands were downloaded and each processed into a form suitable for evaluation. Only populated areas were included, rather than geographical features, so as to limit the variation of the name's source between different countries. The names themselves were processed by converting characters to lower case and removing accented, hyphenated and compound words.

The UK database included a significant number of Welsh names and, since the classifier had been trained on English, it was thought best to remove these. Since the GEONet database included geographic coordinates, it was a simple matter of removing names with longitude less than a certain value. This of course leaves Scottish names in the test set and excludes those from the west of England, but this bias is judged to be acceptable.

As before, for each name, the probability $P_c(W)$ was estimated using the three letter n-gram models, and the most probable language recorded against the name. For this test, the Python n-gram recogniser was used. The confusion matrix is shown in Table 6.

	Number of Words in Assigned Class			
	English	***German***	***Dutch***	***% Correct***
English	3303	945	2396	49.71
German	4648	45346	20444	64.38
Dutch	542	1242	9980	84.84

Total correctly classified: 65.99%

Table 6. Letter N-gram classification of geographic place names by language

The results show a poorer result, 65.99%, for the number of names correctly classified compared with that for the general test set of words from the lexicon. For the English names, a greater proportion are classified as Dutch compared with the lexicon words. This might be caused geographic or historical factors, since some place names, especially in the East of England are derived from Dutch names. Further analysis might confirm this conjecture.

Dutch names are the most easily classified, with a result of nearly 85%, a value which is relatively near to the general set result of 91%. Many more German names are misclassified as Dutch compared with the general test set. Again, this might be due to reasons that go beyond the scope of this study. It does, however, show that the notion of word origin might involve more than a simple classification problem.

7 Language Identification with Grapheme to Phoneme Conversion

So far, the problem of grapheme to phoneme conversion has been approached using an HMM-based model. This has been done for three languages and the results compared with those from previous research. A multi-language model has been developed, and we have examined the implicit language identification properties of this system. In the last section, an explicit language identifier, based on letter n-grams, has been described and it has been shown to be a method of varying effectiveness, depending on the language and the evaluation set.

In this final section, we look at the problem of combining language identification with grapheme to phoneme conversion. This can be achieved by using the letter n-gram classifier to estimate the language of the unknown word, then converting the letter sequence to phonemes using the pronunciation model appropriate to the most likely language. This approach has the disadvantage that a hard decision is made at an early stage in the conversion process, so losing information. To avoid this loss, the output probabilities of the n-gram classifier may be combined with the pronunciation models to obtain an overall posterior probability from both stages. Both these strategies are implemented in the following system.

7.1 Implementation of a combined system

For the implementation of an integrated system, the test set lexicon from the multi-language model was re-used, and 10% of this set was used for the experiments for reasons of time and efficiency. The language identifier stage of the integrated system, the letter n-gram model, is the same Python program that was used for the work on n-gram classification of language in Section 6. The output of the first stage was thus a probability value for each word in each of the three languages, English, German and Dutch, based on the letter n-gram.

The grapheme to phoneme stage comprised the CELEX pronunciation models that were created for Section 5, that is, an HMM for each of the languages, trained on the training set for that language.

For the experiments, the language recogniser was run on the test set using the HMM for each of the three languages. Firstly, a baseline result was established, where the probabilities returned by the recogniser for the English, German and Dutch grapheme to phoneme HMMs were recorded, and the highest scoring model was used. The letter n-gram result was not used for the baseline.

In the second experiment, the output probabilities from the three recognisers were ignored, and instead, the language was identified using the best n-gram recogniser output, and this was used to choose the HMM for the grapheme to phoneme conversion.

In a third experiment, the n-best list from the recogniser output was recored using the result of the n-gram classification. That is, the probabilities of the top 20 possible sequences output from each recogniser were multiplied by the output probability from the matching n-gram language classifier. The sequence with the highest likelihood was then chosen.

The results are shown in Table 7.

Experiment		***% Words Correct***	
Name	***Model***	***Bigram***	***4-gram***
`Baseline`	$\max(P_{rec})$	34.43	68.85
`Hard Decision`	$\max(P_{ngram})$	38.00	70.85
`Soft Decision`	$\max(\text{n-best}_L . P_{ngramL})$	37.39	70.83

Table 7. Grapheme to Phoneme Conversion using Language Identification

Evaluation

When the maximum recogniser probability is used to choose the model output, 34.38% of the words are transformed to exactly the correct sequence using a bigram phoneme model, 68.85% using the 4-gram. The result may be compared with a figure of 18.74% and 47.23% respectively for the multi-language model reported in Table 2. This is to be expected, since the multi-language model is a single HMM trained on all three languages, so information about language of origin is lost. When three individual models are used, the language of the grapheme to phoneme mapping is retained, and the most appropriate model is chosen by examining the probabilities of each transformation.

The second result of 38.00% bigram, 70.85% 4-gram is found where the model output is chosen using the n-gram letter model to classify the language. The result is better than when the individual recognition probabilities are used. Since the grapheme to phoneme models are the same in both experiments, the improvement must come about through improved language recognition; in the first experiment, the wrong language will occasionally be chosen because an incorrect transformation has a lower probability than a correct one.

When the product of recogniser n-best result and letter n-grams is used to choose the model, the result is similar to the use of an n-gram letter classifier on its own, although the bigram model shows a significant degradation of performance. The reasons for this lie in the difference in range between the recogniser n-best probabilities and the n-gram letter probabilities. The latter are smaller in range and so need to be multiplied by a factor, in the same way that a language model needs a scaling factor in an automatic speech recognition system.

Model Scaling Factor

To estimate the correct value of the scaling factor, the recognition results were measured for a range of factors, and the results plotted in Figure 10 for the bigram model and Figure 11 for the 4-gram.

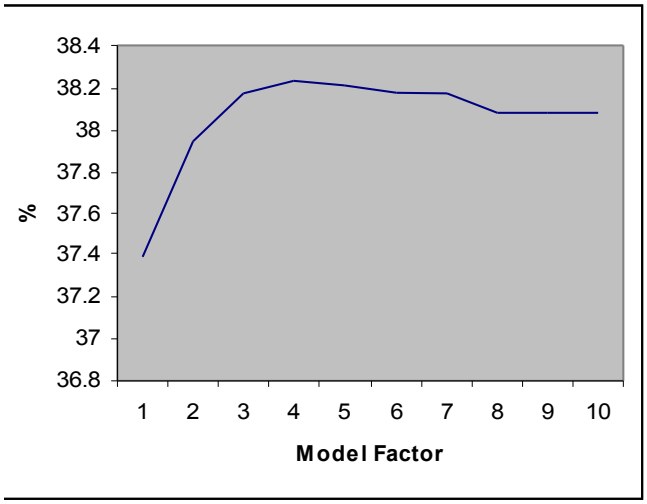

Figure 10. Model Scaling Factor for Bigram

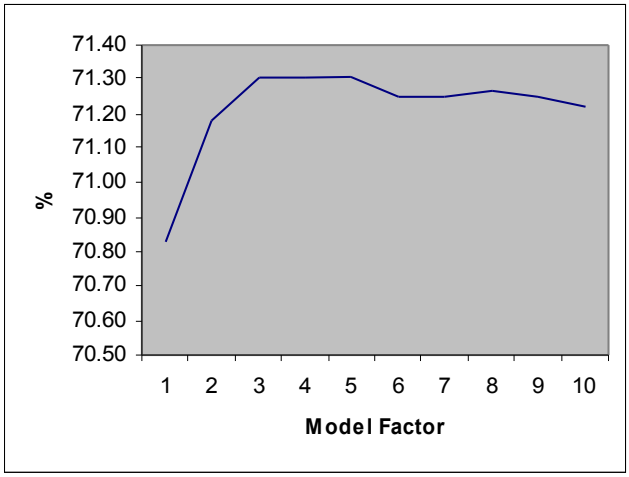

Figure 11. Model Scaling Factor for 4-gram

Both graphs show a peak in posterior probability at a log probability scale factor of 4, and Table 8 shows this result.

Experiment		*% Words Correct*	
Name	***Model***	***Bigram***	***4-gram***
`Soft:`	$\max(\text{n-best}_L \cdot P^4_{ngramL})$	38.24	71.30

Table 8. Grapheme to Phoneme Conversion using Soft Language Identification

It can be seen therefore that a "soft" decision on the output sequence gives a better result than a hard decision being made in the first stage using the most likely language to choose the appropriate grapheme to phoneme model.

There might, though, be reasons to use a hard decision in practice. If there is a large number of languages to choose from, as would be the case in any real system, the running of all recognisers in order to optimise the result by a few percent may not be worthwhile. A compromise might be to use a 'beam' method to run grapheme to phoneme conversion only on the most likely languages.

8 Conclusion

A Hidden Markov Model has been used for grapheme to phoneme conversion and the results for bigram and 4-gram phoneme models published in [3] have been replicated to within a few percent. Hidden Markov Models are an effective method for this task and they have the advantage that a single model is used for both the alignment and transformation stages. They also have a limitation in that there is an assumption of independence between observations. This manifests itself in the way that the model tends to generate a symbol for silent 'e' at the end of words. The problem can be solved by a pre-processing step in which letters are re-arranged to be more amenable to finite state analysis. Another enhancement is to model context-sensitivity of graphemes to detect, for example, hard or soft “c”. A large class of errors is due to incorrect stress assignment, manifesting itself in schwa confusion. This may be addressed by a separate model that predicts the stress pattern of words and modifies the output hypothesis accordingly.

The results obtained using models trained using the CELEX English lexicon with a 4-gram phoneme model, were CELEX at 40.89% and Unisyn 41.63% words correct. The bigram model for CELEX at 20.97% words correct performs better than Unisyn at 16.23% this being attributable to more accurate pronunciation transcriptions. The results for German and Dutch were better than those for English and this was explained by differences in the mapping between spelling and pronunciation.

A quantitative analysis of the pronunciation names was not possible owing to resource limitations, but the CELEX English converter was run on an evaluation set of English place names. This showed many of the problems of the more general set, but also revealed problems that are specific to names. Often the pronunciation of place names is idiosyncratic, and this seems likely to apply to other kinds of name, such as surnames. There is a limit on the accuracy ceiling when knowledge of the domain is not available.

A language classifier based on letter n-grams was implemented and this was found to work well for German and Dutch, though less well for English. If a language has a regular spelling, it is likely that it will be accurately classified by an n-gram model. Clearly, there are limits to this approach because some words are homographs across languages, as for example, “table” in both English and French. To resolve this example would need an examination of word context (to check, for example, if it was preceded by a language-dependent article such as “la”) and any practical system would have to include strategies such as this. However, the classifier described in this report does allow a preliminary investigation of the problem.

The language classifier was run on place names and it returned an accuracy of 65.99% correct, compared with 90.48% correct for a general set of words. The work

with names showed that the concept of word origin is not a simple one. Words and names are shared between languages and it can be hard to define the exact origin.

The results from the combined system which performs language classification before grapheme to phoneme conversion show that there is a benefit from classifying the language before conversion takes place. This is not surprising since the correct pronunciation model may then be chosen according to the language of origin. A further benefit can be gained by a soft decision of language origin, although this was found to be a less than 1% difference in the words correct. There may be practical reasons why an early stage decision on language origin is necessary, in order to avoid the expense of unnecessary conversion.

This project has demonstrated the feasibility of grapheme to phoneme conversion for words of unknown language origin. It also suggests a number of avenues for further work. There is plenty of scope for improving the grapheme to phoneme component by preprocessing and using additional models for stress and phoneme context. Work could be done in looking into the issues that affect grapheme to phoneme conversion which relate to specific languages. The examples given earlier of grapheme dependencies, context and stress are all drawn from English. If each language needs specific adjustments, this does lead to a less elegant model, although it might be the best practical solution. If, however, models are applicable to groups of languages, then a more general solution can be found.

Further work may be done on the grapheme to phoneme conversion of names, since this class of words is likely to need converting. While the use of a set for training and evaluation gives some indication of the general problems, those specific to names may be different. There may also be a need to model words from a particular domain such as surnames or place names.

Finally, the language identification for a real system would be more elaborate than the n-gram classifier described here, and other methods of language classification might be investigated. The work done on combining language class probabilities with an n-best hypotheses from the recogniser gave an interesting result, and this could be explored further.

References

[1] Damper, R. I., Marchand, Y., Adamson, M. J. and Gustafson, K. (1998) " A comparison of letter-to-sound conversion techniques for English text-to-speech synthesis". Proceedings of the Institute of Acoustics 20(6), 1999.

[2] A. F. Llitjos, A. W. Black (2005), Knowledge of Language Origin Improves Pronunciation Accuracy of Proper Names, Eurospeech 2001, Aalborg, Denmark

[3] P. Taylor (2005), Grapheme-to-Phoneme conversion using Hidden Markov Models. Proc. Interspeech 2005.

[4] Baayen, H., Piepenbrock, R. and van Rijn, H. (1993) The CELEX Lexical Database, Release 1 (CD-ROM). Linguistic Data Consortium, University of Pennsylvania, Philadelphia, PA.

[5] Damper, R. I. and Marchand, Y. (1998) Improving pronunciation by analogy for text-to-speech applications. In Proceedings of 3rd ESCA/COCOSDA International Workshop on Speech Synthesis, Jenolan Caves, Australia, pp. 65-70.

[6] S. F. Chen, "Conditional and joint models for Grapheme-to-Phoneme conversion", Eurospeech 2003.

[7] Jelinek, F. "Statistical Methods for Speech Recognition", MIT Press, 1998..

[8] S. Young, G. Evermann, D. Kershaw, G. Moore, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, The HTK Book, Cambridge University Engineering Department, 2002.

[9] A. Stolcke (2002), SRILM -- An Extensible Language Modeling Toolkit. Proc. Intl. Conf. on Spoken Language Processing, vol. 2, pp. 901-904, Denver

[10] GEONet Names Server (GNS). National Geospatial Intelligence Agency, Washington.

Acknowledgments

The author acknowledges the help of Paul Taylor, who supervised this M.Phil project. Thanks also to Graeme Blackwood for advice on language modelling tools.

Appendix 1. Mapping of SAMPA Symbols to Baseline Set

SAMPA	Baseline	SAMPA	Baseline
i	ii	m	ax m
j	y	N	ng
u	uu	n,	ax n
3	aax	N,	ax n
A	aa	O	oo
aI	ai	OI	oi
@	ax	O~	o n
@U	ou	Q	o
A~	o n	S	s
aU	au	T	th
{	a	tS	ch
{~	i n	U	u
D	dh	U@	ax
dZ	jh	V	uh
E	e	Z	z h
E@	eax	eI	ei
I	i	:	null
I@	iax	'	null
l	ax l		

Additions to the set for languages other than English

SAMPA	Baseline	SAMPA	Baseline
&	ap	}	rb
(	cb	!	ex
)	bc	*	sr
=	eq	B	bb
/	fs	G	gg
^	ha	K	kk
|	pi	M	mm
+	pl	W	ww
<	la	X	xx
L	ll	Y	yy

Appendix 2. List of scripts

The following short scripts were created by the author or adapted from originals written by the supervisor.

Script Name	Function	Original •/ Created	Lines
code_letters_fix	Creates data files for recogniser	Modified	36
lat_run	Runs recogniser	•	40
train_models	Trains models using HInit	Modified	27
lex2mlf	Makes label file from lexicon	•	18
number_lines	Adds line numbers to lexicon	•	29
printnthline	Prints lines modulo n	•	19
word_per_line	Prints one word per line	•	18
create_data_sets	Splits file 85:10:5	•	26
run_herest	Trains HMM	Created	9
expand_states	Adds states to an initialised HMM	•	98
tag_phon	Adds language tags to phonemes	Created	15
Dedup	Deduplicates lines	Created	22
conv2sam	CELEX specified converter	Created	18
proc_lex*	Processes CELEX lexicons	Created	52
subst_words*	Processes CELEX words	Created	6
subst_phonemes*	Processes CELEX phonemes	Created	26
pp.pl*	Converts to baseline	Created	112
n_words	Language recogniser	Created	340
ana_names	Writes confusion matrix	Created	12
ana_conf	Write confusion files	Created	6
nbest	Section 7 experiments	Created	159

*One script per CELEX lexicon, held in the lex directory

Appendix 3. HMM Definition File for Phoneme /au/

This model was used to illustrate the transition and observation probabilities in the baseline model. The format is the HTK's HMM definition language. The list of values under <DPROB> are scaled log probabilities for each of the letters a to z. The actual probabilities *P* can be calculated from the values *d* using the formula

$$P = \exp(-d/2371.8)$$

```
~h "au"
<BEGINHMM>
<NUMSTATES> 6
<STATE> 2
<NUMMIXES> 27
<DPROB>
 9213 32767 17164 32767*4 13121 32767*4 17169*2 73 32767*2
 15526 32767*2 14995 32767 17169 32767*4
<STATE> 3
<NUMMIXES> 27
<DPROB>
 32767*14 12518 32767*5 878 32767 2819 32767*4
<STATE> 4
<NUMMIXES> 27
<DPROB>
 32767 2157 32767*2 6578 32767 1556 32767*10 9804 32767*2
 26204 32767*6
<STATE> 5
<NUMMIXES> 27
<DPROB>
 32767*7 0 32767*9 21212 32767*9
<TRANSP> 6
 0.000000e+00 1.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
 0.000000e+00 0.000000e+00 9.632804e-01 0.000000e+00 0.000000e+00 3.671957e-02
 0.000000e+00 0.000000e+00 0.000000e+00 1.939581e-02 0.000000e+00 9.806042e-01
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 5.189452e-01 4.810548e-01
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 1.000000e+00
 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00 0.000000e+00
<ENDHMM>
```

Appendix 4. Code Listings

```
#!/usr/bin/python

import sys
import os

# n_words
#
# letter n-gram language classifier
# reads ARPA-MIT LM format
#

emiss = 0
gmiss = 0
dmiss = 0


def list2string(l):

    out = ""
    for i in l:
       if i[0:3] == 'en_':
           out += i[3]
            prefix = 'en_'
       elif i[0:3] == 'ge_':
           out += i[3]
            prefix = 'ge_'
       elif i[0:3] == 'du_':
           out += i[3]
            prefix = 'du_'
       elif i[0:3] == "<s>":
           prefix = "xx_"
           out += '$'
       elif i[0:4] == "</s>":
           prefix = "xx_"
           out += '.'
       else:
           out += i
    return prefix + out


#
# Read language model
#

fplm = open(sys.argv[1], 'r')

lm = fplm.readlines()


en_lm= { }
ge_lm= { }
du_lm= { }
en_lm1= { }
ge_lm1= { }
du_lm1= { }
en_lm2= { }
ge_lm2= { }
du_lm2= { }
en_lm3= { }
ge_lm3= { }
```

```
du_lm3= { }


got1 = 0
got2 = 0
got3 = 0
got4 = 0

for line in lm:

     if (line == '\\2-grams:\n'):
        got1 = 0
        got2 = 1
        continue

     if (line == '\\3-grams:\n'):
        got2 = 0
        got3 = 1
        continue

     elif (line == '\\4-grams:\n'):
        got3 = 0
        got4 = 1
        continue

     if got1:
            els = line.split()
            if len(els) == 0: continue

            if len(els) > 2 :
              val = els[0] + " " + els[2]
            else:
              val = els[0] + " 0"

            preq = list2string(els[1:2])
            if preq[0:3] ==  'en_':
                en_lm1[preq[3:]] = val
            if preq[0:3] ==  'ge_':
                ge_lm1[preq[3:]] = val
            if preq[0:3] ==  'du_':
                du_lm1[preq[3:]] = val

     if got2:
            els = line.split()
            if len(els) ==0: continue

              if len(els) > 3 :
              val = els[0] + " " + els[3]
            else:
              val = els[0] + " 0"

            preq = list2string(els[1:3])
            if preq[0:3] ==  'en_':
                en_lm2[preq[3:]] = val
            if preq[0:3] ==  'ge_':
                ge_lm2[preq[3:]] = val
            if preq[0:3] ==  'du_':
                du_lm2[preq[3:]] = val

     elif got3:
            els = line.split()
            if len(els) ==0: continue

              if len(els) > 4 :
              val = els[0] + " " + els[4]
            else:
```

```
            val = els[0] + " 0"

            preq = list2string(els[1:4])
            if preq[0:3] ==  'en_':
                en_lm3[preq[3:]] = val
            if preq[0:3] ==  'ge_':
                ge_lm3[preq[3:]] = val
            if preq[0:3] ==  'du_':
                du_lm3[preq[3:]] = val

     elif got4:
            els = line.split()

            if len(els) ==0: break
             val = els[0]
            preq = list2string(els[1:5])
            if preq[0:3] ==  'en_':
                en_lm[preq[3:]] = val
            if preq[0:3] ==  'ge_':
                ge_lm[preq[3:]] = val
            if preq[0:3] ==  'du_':
                du_lm[preq[3:]] = val
     else:
       if  (line == '\\1-grams:\n'): got1 = 1


#
# lang_model
#
def lang_model(lang, size):

    if lang == 'en':
          if size == 4: return en_lm
          elif size == 3: return en_lm3
          elif size == 2: return en_lm2
          elif size == 1: return en_lm1

    elif lang == 'ge':
          if size == 4: return ge_lm
          elif size == 3: return ge_lm3
          elif size == 2: return ge_lm2
          elif size == 1: return ge_lm1

    elif lang == 'du':
          if size == 4: return du_lm
          elif size == 3: return du_lm3
          elif size == 2: return du_lm2
          elif size == 1: return du_lm1

    else: print "Unknown lang in lang_model: " + str(lang)

#
# nlookup1
#
def nlookup_1(lang, cand, bo):

    if cand == '$' or cand == '.': return 0

    lm_dict = lang_model(lang,1)
    if lm_dict.has_key(cand):
         vals = lm_dict[cand].split()
         if bo: out = float(vals[0]) + float(vals[1])
         else: out = float(vals[0])
    else:
       print "Symbol not found: " + cand
    return out
```

```
#
# nlookup2
#
def nlookup_2(lang, cand, bo):

    lm_dict = lang_model(lang,2)
    if lm_dict.has_key(cand):
         vals = lm_dict[cand].split()
         if bo: out = float(vals[0]) + float(vals[1])
         else: out = float(vals[0])
    else:
         out = nlookup_1(lang,cand[1:2],1)
    return out

#
#
# nlookup3
#
def nlookup_3(lang, cand, bo):

    lm_dict = lang_model(lang,3)
    if lm_dict.has_key(cand):
         vals = lm_dict[cand].split()
         if bo: out = float(vals[0]) + float(vals[1])
         else: out = float(vals[0])
    else:
        out = nlookup_2(lang,cand[1:3],1)
    return out

#
# nlookup4
#
def nlookup_4(lang, cand):

    lm_dict = lang_model(lang,4)
    if lm_dict.has_key(cand):
         out = float(lm_dict[cand])
    else:
         out = nlookup_3(lang,cand[1:4],1)
    return out


#
# ngram lookup
#
def nlookup(lang, cand):

    if len(cand) == 4:
       return nlookup_4(lang, cand)

    elif len(cand) == 3:
       return nlookup_3(lang, cand, 0)

    elif len(cand) == 2:
       return nlookup_2(lang, cand, 0)

    else:
       return nlookup_1(lang, cand, 0)


#
# idlang
#
def idlang(aword):
```

```
        en_lp = 0
        ge_lp = 0
        du_lp = 0

        for i in range(3):
                en_lp += nlookup('en',aword[0:i+1])
                ge_lp += nlookup('ge',aword[0:i+1])
                du_lp += nlookup('du',aword[0:i+1])

        if len(aword) > 3:
                for a in range(len(aword) - 1):

                        quad = aword[a:a+4]

                        en_lp += nlookup('en',quad)
                        ge_lp += nlookup('ge',quad)
                        du_lp += nlookup('du',quad)

        if en_lp > ge_lp and en_lp > du_lp :
                lang = 'en'
        elif ge_lp > en_lp and ge_lp > du_lp :
                lang = 'ge'
        elif du_lp > en_lp and du_lp > ge_lp :
                lang = 'du'
        else:
                lang = 'xx'

        out = lang + " " + str(en_lp) + " " + str(ge_lp) + " " + str(du_lp)
        return( out)

#
# Main
#

fp = open(sys.argv[2], 'r')
entries = fp.readlines()

for word in entries:

         aword = "$" + word.strip() + '.'
        lang = idlang(aword)
        print word.strip() + " " + lang
```

```
#!/usr/bin/python

import sys

#
# n_best - script to run Section 7 Experiments.
#

# fourgram
#flet = "lex/x_eval_10.let"
#fen1 = "results/en1_4e.res"
#fge1 = "results/ge1_4d.res"
#fdu1 = "results/du1_4d.res"

# bigram
flet = "lex/x_eval_10.let"
fen1 = "results/en1_2a.res"
fge1 = "results/ge1_2a.res"
fdu1 = "results/du1_2a.res"

# open result files
fp = open(fen1, 'r')
en1 = fp.readlines()

fp = open(fge1, 'r')
ge1 = fp.readlines()

fp = open(fdu1, 'r')
du1 = fp.readlines()

def read_mlf(l):

    index = -1
    out = { }
    line_no =0
    record = ""

    for i in l:
       if i[0] == '"' :
            record += i
            index += 1
       elif i[0] != '.':
            record += i
       elif  i[0] == '.':
            out[index] = record
             record = ""
    return out


def get_prob(lstr):

    line_no =0
    record = ""
    logprob = 0

    best = -99999
    for i in lstr.split('\n'):
        if i =='':
            if logprob > best:
               best = logprob
             logprob = 0
       elif i[0] == '"' or i[0] == '#' :
            line_no += 1
       elif i[0] != '.' and i[0:3] != '///':
            record = i.split()
             # fix for en
```

```
            # if record[2] != '<s>' and record[2] != '</s>':
            logprob += float(record[3])
       elif  i[0] == '.' or i[0:3] == '///':
            if logprob > best:
              best = logprob
            logprob = 0
    return best



def get_best(lp, ens, ges, dus):

    lp_en = lp[0]
    lp_ge = lp[1]
    lp_du = lp[2]

    pen = get_prob(ens) + float(lp_en)*8
    pge = get_prob(ges) + float(lp_ge)*8
    pdu = get_prob(dus) + float(lp_du)*8

    # phoneme model only
    #pen = float(lp_en)
    #pge = float(lp_ge)
    #pdu = float(lp_du)

    if pen > pge and pen > pdu : res = 'en'
    if pge > pen and pge > pdu : res = 'ge'
    if pdu > pen and pdu > pge : res = 'du'

    return res


# read in the mlfs and index by test word index
en_mlf = read_mlf(en1)
ge_mlf = read_mlf(ge1)
du_mlf = read_mlf(du1)


fp = open(flet, 'r')
let = fp.readlines()

n = 0

# read through the lines in the list of n-gram letter probs
for line in let:

    lp = let[n].split()[2:]

    lang = get_best(lp, en_mlf[n], ge_mlf[n], du_mlf[n])

    if lang == 'en':
       sys.stdout.write(en_mlf[n])
       print '.'

    if lang == 'ge':
       sys.stdout.write(ge_mlf[n])
       print '.'

    if lang == 'du':
       sys.stdout.write(du_mlf[n])
       print '.'

    n = n + 1
```