IMPROVED SPELLING RECOGNITION USING A TREE-BASED FAST LEXICAL MATCH

Carl D. Mitchell and Anand R. Setlur

Lucent Technologies Bell Laboratories
2000 N. Naperville Rd., Naperville, IL 60566, USA
{carlmitchell, asetlur}@lucent.com

ABSTRACT

This paper addresses the problem of selecting a name from a very large list using spelling recognition. In order to greatly reduce the computational resources required, we propose a tree-based lexical fast match scheme to select a short list of candidate names. Our system consists of a free letter recognizer, a fast matcher, and a rescoring stage. The letter recognizer uses n-grams to generate an n-best list of letter hypotheses. The fast matcher is a tree that is based on confusion classes, where a confusion class is a group of acoustically similar letters such as the e-set. The fast matcher reduces over 100,000 unique last names to tens or hundreds of candidates. Then the rescoring stage picks the best name using either letter alignment or a constrained grammar. The fast matcher retained the correct name 99.6% of the time and the system retrieved the correct name 97.6% of the time.

1. INTRODUCTION

Accurate spelling recognition of telephone bandlimited speech represents a challenging task for machines (and for people) due the existence of confusable letter groups, e.g., the e-set { *b, c, d, e, g, p, t, v, z* }. Good performance requires a tightly constrained grammar such as a list of all possible spellings. Unfortunately, searching a fully constrained grammar is often extremely computationally and memory intensive. This paper presents a two-pass method that greatly reduces the resources required for accurate spelling recognition. Although our approach is applicable to many types of tightly constrained grammars, we will focus on the task of recognizing a spelled name from a large list of candidate names, as in automated directory assistance.

2. RELATED WORK

Previous work on acoustic modeling for spelled word recognition has been reported by several authors and is summarized in [1]. Typical letter recognition accuracies vary between 85% and 90% for telephone bandlimited speech and a speaker independent system. In [2], letter recognition accuracy is improved from 88.2% to 98.1% by confining the search space of the recognizer to a pre-determined set of names and imposing probabilities on the search tree based on a language model. With this method, the memory and cpu requirements increase as the number of candidates increase. In our method, the search space is less dependent on the vocabulary size because the first stage is a self-looping grammar where any letter can be followed by any letter. In [3], a self-looping grammar is used in conjunction with a lexical matcher. This matcher, first proposed in [4], uses a distance based on the number of insertions, deletions, and substitutions to retrieve the N closest lexical neighbors for further processing. The drawback with this method is that even though the entire tree is not searched to find the top N lexical neighbors, the lexical matching can still be quite expensive for a large name list. The fast lexical matcher that we propose in this paper has the list of lexical neighbors stored ahead of time and uses a tree search to quickly generate a short list of candidates.

3. DATABASES

The acoustic training set consists of a set of 7200 spelled names from the MACROPHONE collection and another set of 1500 spelled New Jersey town names. This training set yielded 64,000 training tokens from which the acoustic models were estimated.

The full list of names consists of all the last names in New York City plus New Jersey town names. The lexicon is comprised of the 133,000 unique names in the list of over six million NY/NJ names. The n-grams, which were trained using the list of unique names, requires 0.1 Meg, 0.4 Meg, and 1.9 Meg of recognizer memory for the cases when n is 3, 4, and 5, respectively.

Since we do not have acoustic data for New York last names, our test set consisted of 500 New Jersey town names comprising a total of 5,100 spoken letters. The OGI spelled name corpus distributed by the Linguistics Data Consortium was not available to us at the time of publication. Since several results on the OGI database have been published (e.g., [2]), we will present our results on the OGI database at the conference.

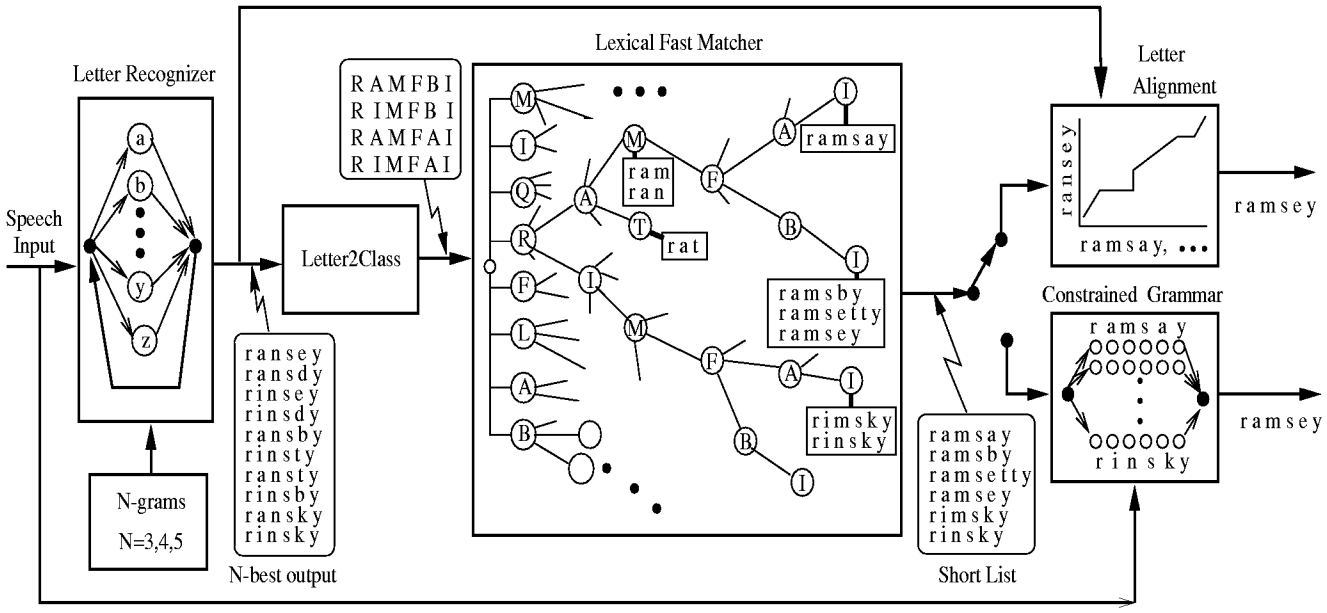

Figure 1: System Architecture.

4. SYSTEM OVERVIEW

Figure 1 depicts our system. The spoken letters processed by a free letter recognizer generate a list of n-best hypotheses. Each hypothesis is converted to a sequence of class letters that are used to search a tree. Starting at the root of the tree, the class sequence specifies a path to a leaf that contains NY/NJ names similar to the input letter hypothesis. The concatenation of names across all n-best leaves provides a short list of candidates that can be searched in more detail in the rescoring stage using either letter alignment or an acoustic search using a tightly constrained grammar.

5. LETTER RECOGNIZER

The acoustic models are continuous density, context independent, left-to-right, Hidden Markov Models (HMMs) with each letter of the alphabet being modeled as a separate unit. Vowels are modeled with 3 states and consonants are modeled with 6 states, where each state is modeled as a mixture of 16 Gaussians. An additional single state 32 Gaussian mixture silence model rounds out the 27 models. All models were discriminatively trained.

The topology of the letter recognizer allows any combination of letters, as shown at the far left of Figure 1. Although the topology does not include lexical constraints, an n-gram language model trained using the NY/NJ lexicon provides much of the same information. This solution requires far less memory and processing time than a fully constrained search. After experimenting with language weights between 5 and 30, a value of 15 was chosen for all of the experiments reported in this paper. Without n-grams, 88.1% of the letters spoken were recognized correctly, word accuracy (%correct - %insertions) was found to be 87.2%, and the string accuracy was only 39.4%. Using 5-grams that were built for the NY/NJ lexicon, the performance increased to 96.9%, 96.0%, and 75.4%, respectively. This illustrates how well the lexical constraints are captured by the n-gram language model.

6. LEXICAL FAST MATCH

We grouped the 26 letters of the English alphabet into 8 classes based on acoustic similarity, as shown in Table 1. A capital letter denotes a class name composed of similar sounding letters. For example, the set identified by B consists of the e-set letters: *b, c, d, e, g, p, t, v,* and *z*. (Note that letters *l* and *o* could be assigned separate classes since they are not similar acoustically; by merging these letters, each class label can be represented with 3 bits.)

Class Name	Letters
A	*a, h, j, k* (a-set)
B	*b, c, d, e, g, p, t, v, z* (e-set)
F	*f, s, x*
I	*i, y*
M	*m, n*
Q	*q, u, w*
L	*l, o*
R	*r*

Table 1: Letter Classes

Each name of the NY/NJ lexicon is converted to a key that will be stored in a tree, where a key is formed by replacing each letter with its class letter. A leaf in the tree

contains all names that map to the leaf's key. Since most mistakes made by the letter recognizer are within-class substitutions, the key accuracy will be higher than the letter accuracy. The most likely string of letters as provided by the letter recognizer is converted to a key that can then be found by traversing the tree. All of the names that reside at that leaf are added to the short list.

Since it is unrealistic that the top hypothesis from the letter recognizer is always going to yield the correct key, the n-best hypotheses from the letter recognizer are converted into n keys. Each unique key contributes one or more names to the short list of candidate names.

7. RESCORING STAGE

As shown at the far right of Figure 1, we consider two methods to select the best name from the short list of candidates provided by the lexical fast matcher.

7.1. Rescoring Using Letter Alignment

Once the short list of candidates is found by the fast matcher, each is aligned with the top hypothesis from the free letter recognizer. Using an edit score, dynamic programming minimizes the total number of insertions, deletions, and substitutions. The retrieved name is the candidate with the minimum edit score.

7.2. Rescoring Using Constrained Grammar

A second way to choose among the candidates provided by the fast matcher is to re-recognize the acoustic signal with a constrained grammar that corresponds exactly to the short list of candidates. The same thing could be done for the large list of names, but we found that this requires too much processing time for a task like directory assistance. This method works well, however, for the short list because the average number of candidates is usually well below one hundred.

8. RESULTS

In Tables 2, 3, and 4, the label "Fast" gives the performance of the lexical fast matcher, which is the percent of the time that the correct candidate is retained. This is not a measure of the system performance since the short list of candidates has not yet been resolved. The retrieval accuracies for the letter alignment rescoring method and the constrained grammar rescoring method are labeled with "Align" and "Rescore", respectively. Each of these scores must be less than or equal to "Fast", since an error by the fast matcher is propagated to the rescoring stage. The average number of candidates generated by the fast matcher is labeled "Avg" and represents an estimate for the size of the short list.

8.1. Baseline

The baseline system is summarized in the system overview described in Section 4. As can be seen from Table 2, increasing the order of the n-gram language model significantly improves performance while having little effect on the number of candidates that need to be rescored. Note that "Rescore" consistently performs better than "Align".

Ngram	Nbest	Fast	Avg	Align	Rescore
3	10	93.0	2.1	88.4	89.4
3	20	94.6	2.6	90.2	91.6
4	10	94.2	2.0	90.4	91.2
4	20	95.6	2.6	91.8	93.0
5	10	96.0	2.0	93.2	93.8
5	20	97.0	2.6	93.2	94.2

Table 2: Baseline System Performance

There is a tradeoff between the accuracy of the fast matcher and the number of candidates passed on to the rescoring stage. As expected, both the average number of candidates and the overall performance increase with more n-best hypotheses.

8.2. Improvements To Lexical Fast Match

The fast matcher can be improved by making it less sensitive to the errors made by the letter recognizer. We found that the letter recognizer is less accurate for the first and last letters, probably because people speak differently as they begin and end a string of letters. By skipping the first and/or last letters when creating the class tree key, the fast matcher will ignore mistakes made at these positions, leaving it to the more accurate rescoring stage to resolve any differences.

Rather than skipping just the last letter, we also tried truncating after 7 letters. In addition to tolerating mistakes at the ends of spelled names, this also has the benefit of greatly reducing the size of the class tree. Changing the maximum key length from 10 to 7 reduces the number of nodes in the tree from 115,000 to 72,000.

Insertions and deletions made by the letter recognizer can lead to the wrong key for the fast matcher. We can make the key tolerant of within-class insertions (e.g, *b* recognized as *b e*) and deletions (e.g, *b e* recognized as *b*) by removing duplicates of classes. For example, the last name "ramsetty" would have key RAMFBI instead of RAMFBBBI. Cross-class insertions and deletions will still result in the wrong key. We depend on the n-best algorithm to provide a hypothesis with no cross-class insertions or deletions.

These variations often result in a larger list of candidates but are more likely to preserve the winner in the short list, as shown in Table 3. All experiments reported in Table 3 use 5-grams with n-best set to 20. The first column gives

the maximum length of the key. Keys with more class letters than the maximum are truncated, resulting in a smaller tree at the expense of a larger list of candidates. The second column specifies whether duplicate class letters are removed. The third column indicates if the first class letter was deleted and the fourth column indicates if the last class letter was deleted.

Len	Dup	First	Last	Fast	Avg	Align	Rescore
7	N	N	N	98.0	7.6	95.0	96.0
7	N	Y	N	98.2	12.7	94.2	95.6
7	N	N	Y	98.2	11.5	94.8	96.0
7	N	Y	Y	98.4	54.0	94.0	95.4
7	Y	N	N	99.2	11.4	96.6	97.6
7	Y	Y	N	98.4	29.6	95.4	96.6
7	Y	N	Y	99.2	29.0	96.4	97.4
7	Y	Y	Y	99.2	124.4	95.8	97.2
10	N	N	N	97.0	2.6	93.2	94.2
10	N	Y	N	96.8	9.3	92.4	93.8
10	N	N	Y	98.2	8.3	93.8	95.0
10	N	Y	Y	98.4	45.0	93.0	94.4
10	Y	N	N	97.2	8.0	94.6	95.6
10	Y	Y	N	97.2	28.1	94.2	95.4
10	Y	N	Y	98.6	27.2	95.8	96.8
10	Y	Y	Y	98.6	118.6	95.2	96.6

Table 3: Effects of Varying Key Length and Skipping

From the table, it can be seen that removing duplicates and truncating after 7 class letters is advantageous. Removing the last class letter usually helps, but removing the first class letter usually does not.

8.3. Merging Classes

The 8 classes listed in Table 1 were chosen based on acoustic similarity. The only variation considered in this paper was to merge the a-set with the e-set since confusion between these classes accounts for many errors. Table 4 gives results for the original 8 class set and the reduced 7 class set for the case when the letter recognizer used 5-grams with n-best set to 20, the key length set to 7, and within-class duplicates removed.

There is only one case in Table 4 where 7 classes led to better results than 8 classes, and this minor gain comes at the expense of greatly increasing the number of candidates in the short list. However, the 7 class variation provides some benefit when either the number of hypotheses or the order of the language model is reduced. We also expect that 7 classes might be required for tasks where the letter recognition accuracy is lower.

First	Last	#cl	Fast	Avg	Align	Rescore
N	N	7	99.4	66.2	96.4	97.2
N	N	8	99.2	11.4	96.6	97.6
Y	N	7	98.8	150.1	95.4	96.6
Y	N	8	98.4	29.6	95.4	96.6
N	Y	7	99.6	168.3	96.8	97.6
N	Y	8	99.2	29.0	96.4	97.4
Y	Y	7	99.6	466.6	95.0	96.2
Y	Y	8	99.2	124.4	95.8	97.2

Table 4: Effect of Merging Classes

9. CONCLUSIONS

We have proposed a fast method to select a short list of candidates from a large list of names used in a spelling recognition task. For the best case, as shown in row 5 of Table 3, the correct name is retained 99.2% of the time for an average short list length of 11.4 and the overall system accuracy was 97.6%. Compared to related systems, our two-stage approach uses far less memory and cpu time. Hence this approach lends itself well to a multiple channel implementation on a memory-constrained hardware platform such as the Lucent Speech Processing Solutions board.

10. ACKNOWLEDGMENTS

The authors acknowledge David Thomson and Rafid Sukkar for helpful discussions, and Greg Szeszko for providing the initial set of acoustic models.

11. REFERENCES

[1] P. C. Loizou, A. S. Spanias, "High-Performance Alphabet Recognition," *IEEE Transactions on Speech and Audio Processing,* pp. 430-445, 1996.

[2] H. Hild and A. Waibel, "Recognition of Spelled Names Over the Telephone," *Proceedings Fourth International Conference on Speech and Language Processing,* pp. 346-349, 1996.

[3] G. Gravier, F. Yvon, G. Etorre and G. Chollet, "Directory Name Retrieval Using HMM Modeling and Robust Lexical Access," *Proceedings IEEE Workshop on Automatic Speech Recognition and Understanding,* pp. 558-565, 1997.

[4] K. Oflazer, "Error-Tolerant Finite-State Recognition With Applications To Morphological Analysis and Spelling Correction," *Computational Linguistics,* pp. 73-89, 1996.