Reliable spelling despite poor spoken letter recognition

Matt Marx and Chris Schmandt

MOTIVATION

Speech is a powerful, flexible, and familiar interaction modality -- after all, conversation is the medium of choice in human relations. Although speech recognizers promise to bring this rich, expressive channel to human-computer interaction, spoken language systems will never succeed commercially unless they compensate for imperfect recognition. Given the choice between a difficult-to-learn interface that works and one that is intuitive but unreliable, users will opt for the former.

The canonical approach to improving recognition performance is to reduce *perplexity*, the number of utterances the recognizer is asked to listen for. Restricting the vocabulary may prove useful where the options can be represented by a few key words. If the number of choices is large, however, restricting the vocabulary is not an option. If for example, the user is allowed to ask the price of any stock on the New York Stock Exchange, the perplexity is necessarily in the thousands. Poor recognizer performance is inevitable.

Many long lists will consist of names. Names present a special challenge for speech recognizers (as will be discussed below). Whether choosing between companies on the stock market, cities in a travel guide, friends in a rolodex, or artists in a jukebox, users must have alternative methods of specifying names when the speech recognizer fails.

WHY NAME RECOGNITION IS HARD

Recognizers can generally be divided into two groups: speaker-dependent and speaker-independent. The former is intended for a single user, whereas the latter is designed to be used by anyone.

A speaker-dependent recognizer may outperform a speaker-independent one on lists of high perplexity since its speech templates are tuned for a particular speaker through training. Precisely this training, however, renders speaker-dependent recognizers intractable for large lists: who wants to train the entire list of NYSE stocks? The setup for such a large list may prove too great a barrier for first-time users. Long-term users may be frustrated, as well, since the constant training of new names in a rolodex may be annoying.

Speaker-independent recognizers do not require user-specific training. Instead, they operate by pattern-matching against phonetic models. Words to be recognized are specified in a lexicon as a sequence of phonemes. Each phoneme has a number of possible spectral representations based on training data collected from a number of speakers. The recognizer, given a stream of audio, computes how closely the user’s utterance matches the concatenated spectral characteristics of a phoneme strings which make up various words and then returns the word with the best score. Since the process is essentially one of finding the best match from the vocabulary, the difficulty of the task increases with vocabulary size.

Since the models are trained before the fact, neither setup nor maintenance costs are of concern. Shorter, simpler ramp-up may make personal speech systems more appealing.

And any system with an unknown or dynamic user population, such as a 1-800 weather service or a jukebox at a bar, must use a speaker-independent system: casual users will not want to invest more time training a system than using it!

Grammar for Name Recognition

```
Start(name).
name ---> Aaron
name ---> Barbara
name ---> David
name ---> Eric
name ---> Franklin
name ---> Kim
name ---> Jim
name ---> Philip
name ---> Smith
name ---> Smithson
name ---> Tim
```

Lexicon of Proper Names

```
namelex:
Aaron: ER AX N
Barbara: B AR B R AX
David: D EY V AX D
Eric: ER IH K
Franklin: F R EY N K L IH N
Kim: K IH M
Jim: JH IH M
Philip: F IH L AX P
Smith: S M IH TH
Smithson: S M IH TH S AX N
Tim: T IH M
```

Proper names present a particular problem since many of them do not appear in the lexicons of speaker-independent recognizers. Spiegel estimates that there are over 1.5 million surnames in the United States alone. [SPIEGEL] Now, one may hand-code the series of phonemes which represent a particular name, but this requires knowledge of not only phonetics but also the notational conventions of the system's speech recognizer. Although administrators of large-scale systems may find sufficient payoff in doing so, it is doubtful that individual users will find the price worth paying.

Pronunciations may be generated automatically using text-to-phoneme rules, but since many proper names are of foreign origin, using English rules alone may lead to false pronunciations. One might attempt to divine the roots of a foreign name and then apply language-specific rules, but this is nontrivial. [VITALE] A lexicon full of faulty phonetic representations can only result in poorer recognition. For instance, applying text-to-phoneme rules for the name "Sidner" (SIDE-ner) results in /sihdner/. If the user then says "Sidner" when the recognizer thinks the name is pronounced /sihdner/, the recognizer's chances are even worse.

SPELLING: A FALLBACK METHOD

Faced with almost certain misrecognition of proper names, the designer must devise fallback methods for specifying one in a large list of choices. An obvious suggestion is to spell out the desired word. Spelling can either be done continuously, with no breaks between the letters, or discretely, where each letter is prompted and processed individually.

Continuous-letter spelling

With a speech recognizer, recognizing the spelling of a name is not much different from recognizing the name itself. Each letter in the alphabet is in fact made up a phonemes: 's', for example, contains two phonemes: /eh/ and /s/. Thus the phoneme string which makes up the word is replaced by the phoneme string which makes up the list of letters that spell out the word. Chances are that the phoneme string for the letters will be longer than that for the name, and so the recognizer has more data to work from in distinguishing the various names, improving recognition. The diagram below compares the phonemes which make up the pronunciation of the name "Tommy" with its spelling. The spoken word has four phonemes whereas the spelling has nine, over twice as many.

Tommy: T-AA-M-IY

T-o-m-m-y: T-IY-OU-EH-M-EH-M-W-AY

This approach is far from foolproof, however. Certain liabilities of speech recognizers are exacerbated by continuous spelling. A major difficulty in speaker-independent recognition is the variation in speakers' pronunciations. This may be even more true in the case of spelling; in addition to varying their pronunciation, people spell at different words at different rates. I'll spell my own last name "M-A-R-X" very quickly, whereas someone else's name may require more concentration. In fact, I may alter the rate of spelling *within* a word. I may spell partway, stop to think, and then continue on, as seems to always be the case with "Y-A-N-<pause>-K-E-L-O-<quickly>-*VICH*". While I paused, the recognizer may have sensed the silence, assumed I was finished, and gone off to process the incomplete utterance.

Continuous spelling lacks the constraint of length. Since the user does not pause between letters, and since names are spelled at different rates, confusion can arise with respect to how many letters were spoken. For instance, "MARX" and "MARKS" could be mistaken for each other.

In summary, several factors may conspire to produce an incorrect recognition of a name spelled out continuously. Although careful user attention to meter and speed may improve the chances of correct recognition, there is no guarantee that the system will make the right guess. (Shifting the burden to the user is the wrong interface approach anyway.) In short, continuous spelling is not reliable.

Discrete-letter spelling

Discrete-letter spelling offers a reliable system for spelling names. Spelling a name one letter at a time, prompting for each letter individually, overcomes some pitfalls of continuous spelling. Since the letters are separated, there is an added constraint of length. The recognizer knows exactly how many letters have been spoken. Additionally, since each letter is prompted for individually, there is no variation in the rate of spelling or danger of coarticulated letters being interpreted as a single letter. "MARX" will never be mistaken for "MARKS".

A further advantage is gained by processing each letter separately. If a name can be uniquely specified with fewer than all of its letters, then time is save. For instance, if the user wants to specify "Franklin" and that is the only name in the list beginning with 'F', the user need only say 'F'; the system will figure out that "Franklin" is intended .

One obstacle not overcome by discrete spelling – exacerbated, in fact – is the high *confusability* of individual letters: although the perplexity of the letters is only 26, many letters are so similar as to be indistinguishable. The most obvious groupings are those consonant letters sharing a vowel in their individual pronunciation, such as the "e-set": 'b', 'c', 'd', 'e', 'g', 'p', 't', 'v', 'z'. The vowel sound dominates the pronunciation with only minimal influence from the consonants; thus the letters in the e-set are difficult to distinguish. (An "a-set" and "eh-set" exist as well.) There also exists a set of fricatives easily confused: ‘s’ and ‘x’ share a voiceless fricative /s/. Consonants which differ only in voicing, such as ‘p’ and ‘b’, are easy to confuse, as well.

Anyone who has spelled their name and address over the phone is well-acquainted with these problems. Inevitably, operators question “is that ‘b’ as in boy or ‘p’ as in Paul?”. If only one or two letters is confusable, this may be an option, but as the confusability matrix below shows, multiple are often mistaken for each other. Asking, “is that ‘b’ as in boy, ‘c’ as in cat, ‘d’ as in dog, ‘g’ as in garden, ‘p’ as in pumpkin, ‘t’ as in Tom, ‘v’ as in vacation, or ‘z’ as in zeppelin?” is overwhelming.

One method of scaling down the technique for use by recognizers is to specify a one-to-one mapping from each letter onto a word beginning with that letter -- words which themselves are not confusable. One such set is the “alpha bravo charlie” convention used by the military. Although the perplexity is still 26, the confusability is much lower since the words are not easily mistaken for each other. This system is purportedly robust and may have advantages for those willing to master the convention, yet the learning curve is too high for first-time or casual users of public systems.

Wishing to avoid a steep learning curve, we reject the “alpha bravo charlie” method of disambiguation -- indeed, we disdain the notion of forcing the user to learn foreign conventions in order to compensate for the recognizer’s poor performance. Instead, we offer an algorithm for implicitly (i.e., without user input) disambiguating spoken letters.

Given that individual letters are often confused by the recognizer, we build a *confusability matrix* which describes the space of possible letter misrecognitions. This matrix consists of the confusable sets described above and the idiosyncratic misrecognitions of the individual recognizer. The confusability matrix for Dagger, a speaker-independent continuous-speech recognizer from Texas Instruments, is shown below. It was constructed by saying each letter in the alphabet 100 times (using multiple speakers) and recording which letters the recognizer thougt it heard. (Ideally, the same speakers would be used to construct this matrix as were used to collect the training data.)

A certain letter **returned** by the recognizer could have been *spoken* by the user as**...**

a --> *ah*	**h** --> *ah*	**n** --> *anrs*	**t** --> *dept*
b --> *abdepvz*	**i** --> *iy*	**o** --> *lo*	**u** --> *qu*
c --> *ctz*	**j** --> *adgjktz*	**p** --> *cdepvz*	**v** --> *bdepvz*
d --> *cdvz*	**k** --> *adjkq*	**q** --> *qu*	**w** --> *fmnw*
e --> *e*	**l** --> *l*	**r** --> *iry*	**x** --> *sx*
f --> *fx*	**m** --> *mn*	**s** --> *fs*	**y** --> *y*
g --> *gt*			**z** --> *defnstvxz*

The table is read as such: either a ‘d’, ‘e’, ‘f’, ‘n’, ‘s’, ‘t’, ‘v’, ‘x’, or ‘z’ returned by the recognizer could mean that the user said ‘z’. By contrast, if the recognizer returns ‘l’, we can be completely sure that the user said ‘l’. The letter ‘z’ is perhaps the hardest to identify, since any of nine letters could be confused for it.

The confusability matrix above is constructed on the assumption that the recognizer is listening for all 26 letters. In normal use, however, this is not necessary. In many lists, for instance, no name will begin with ‘q’ or ‘z’. Thus to listen for either is wasted effort and may lead to the consideration of irrelevant names whose letters match those in the confusability set for ‘z’. As the list shrinks, fewer and fewer letters are relevant. To take advantage of these added constraints, we listen for only the relevant letters. Note that we could, but do not construct a separate confusability matrix for each of the 26! subsets of the alphabet since the irrelevant alternatives are automatically screened out.

The algorithm whittles down a large list of names gradually until either a unique name can be converged upon, or until the user has specified all the letters in the name (signified by saying "that's all" instead of a letter). If the system converges upon a single name with fewer than N letters, where N is the length of the desired name, the system will tell the present the user with the name. If the user has finished and only one of the possible matches meets the constraint of length, the system will return the appropriate match. If the user has finished and several names match, the system goes through the remaining names one by one and asks for confirmation.

Implicit disambiguation of spoken letters works primarily because the vowels a,e,i,o,u are robustly distinguished by the recognizer. An 'a' returned by the recognizer was either a spoken 'a' or 'h', an 'e' was most certainly an 'e', an 'i' was either an 'i' or a 'y', an 'o' was an 'o', and a 'u' was a 'q' or a 'u'. Since each vowel is confusable with at most one other letter, the chances of converging are heightened.

Examples of discretely spelling names

In the following examples, we assume that the user spells the name correctly. In some cases the recognizer will get the letters right; in others, it will make mistakes. For these examples we will assume the following list of names:

{Aaron, Barbara, Charles, David, Danny, Dazzle, Dennis, Eric, Franklin, Kim, Jim, Philip, Smith, Smithson, Tim, Van}

1. Assume the user wants to specify the name David, and that the recognizer gets each letter right. The system will converge without needing to hear all the letters in the name.

   The user says the letter D.

      The recognizer listens for A,B,C,D,E,F,K,J,P,S,T,V (the first letter in each name) and hears D.

      Consulting the confusability matrix, D is replaced by {**C**,**D**,**V**,**Z**}.

      Charles, **D**avid, **D**anny, **D**azzle, **D**ennis, and **V**an have a C,D,V or Z in the initial position.

      We reduce the list to {Charles, David, Daniel, Dazzle, Dennis, and Van}.

   The user says the letter A.

      The recognizer listens for A,E,H (the second letter in each name) and hears A.

      Consulting the confusability matrix, A is replaced by {**A**,**H**}.

      C**h**arles, D**a**vid, D**a**nny, D**a**zzle and V**a**n (but not D**e**nnis!) have an A or H in the second position

      We reduce the list to {Charles, David, Daniel, Dazzle, Van}.

   The user says the letter V.

      The recognizer listens for A,V,N and hears V.

      Consulting the confusability matrix, V is replaced by {**B,D,E,P,V,Z**}.

      Both Da**v**id and Da**z**zle have a B, D, E, P, V or Z in the third position.

      We reduce the list to {David, Dazzle}.

   The user says the letter I.

      The recognizer listens for I,Z and hears I.

      Consulting the confusability matrix, I is replaced by {**I**}.

      Only Da**v**id has an I in the third position.

      We reduce the list to {David}.

   The system returns the name "David", having only needed the four letters DAVI to uniquely specify the name.

2. Assume again that the user wants to specify the name David, but this time that the recognizer performs dreadfully. The system will still converge to one name.

The user says the letter D.

The recognizer listens for A,B,C,D,E,F,K,J,P,S,T,V and hears V.

Consulting the confusability matrix, V is replaced by {**B,D,E,P,V,Z**}.

Barbara, **D**anny, **D**avid, **D**azzle, **D**ennis, **E**ric, **P**hilip, and **V**an have a B, D, E, P, V or Z in the initial position.

We reduce the list to {Barbara, Daniel, David, Dazzle, Dennis, Eric, Philip, Van}.

The user says the letter A.

The recognizer listens for A,E,R,H and hears H.

Consulting the confusability matrix, H is replaced by {**A**,**H**}.

B**a**rbara, D**a**nny, D**a**vid, D**a**zzle, P**h**ilip, and V**a**n have an A or H in the second position

We reduce the list to {Barbara, Daniel, David, Dazzle, Philip, Van}

The user says the letter V.

The recognizer listens for I,N,V,Z and hears Z.

Consulting the confusability, Z is replaced by {**D,E,F,N,S,T,V,X,Z**}.

Da**n**ny, Da**v**id, Da**z**zle, and Va**n** have a D, E, F, N, S, T, V, X, or Z in the third position.

We reduce the list to {Danny, David, Dazzle, Van}.

The user says the letter I.

The recognizer listens for N,I,Z and hears I.

Consulting the confusability matrix, I is replaced by {**I**}.

Only Dav**i**d has an I in the third position.

We reduce the list to {David}.

The system returns the name “David”, having only needed the four letters DAVI to uniquely specify the name, even though the recognizer misrecognized them as VHZR.

3. Assume that the user wants to specify the name Smith. Assume that the recognizer heard the letters SMITH correctly and was able to reduce the list to {Smith, Smithson}. This time, two names match though the intended name has been spelled out.

The user says the phrase “that’s all”.

The system incorporates the constraint of length to converge on one name. Since five letters have been spoken, and “Smith” has five letters whereas “Smithson” has seven, only “Smith” could have been meant.

The list is reduced to {Smith}.

The system returns the name “Smith”, having needed five letters plus the delimiter “that’s all” to infer which name was meant.

4. Assume that the user wants to specify the name Jim. Assume that the recognizer recognizes the letters correctly, which reduces the list to {Kim, Jim, Tim}. Again, three names match though the intended name has been fully spelled out.

The user says the phrase “that’s all”.

The system incorporates the constraint of length, looking only for names of length three. But this includes {Kim, Jim, Tim}, so the length constraint has not helped.

The system falls into interactive disambiguation mode.

The system asks “Was it Kim?”

The user says no, and the list is reduced to {Jim, Tim}.

The system asks “Was it Jim?

The user says yes, and the name Jim is returned.

The system needed all three letters plus “that’s all” plus interaction with the user to specify the name.

EVALUATION

Algorithm performance

Assuming that the spoken letters are recognized within the confusability matrix, the system will either converge to one name or come up with several names which could match the letters; the latter case is known as a *collision*. Since collisions require further effort on the part of the user in disambiguating among the possibilities, we take as the measure of the algorithm's performance how often it can avoid collisions.

Several sample sets were input to the algorithm to test its performance. Misrecognition of individual letteres was simulated by "mutating" the words -- switching each letter with one randomly selected from its corresponding set in the confusability matrix. For instance, "Marx" might be given to the algorithm as "Njrs", "Mkrx", or "Marx". Then, using as few letters as possible, the algorithm attempts to decipher which name the user was spelling.

The table below describes the results of this testing. For each sample set, six measurements are presented: (1) the average number of letters per word (calculated independent of the algorithm); (2) the number of words in the set which could be converged upon given a random mutant of the original word; (3) the average number of letters per word required to converge; (4) the number of collisions or words which couldn't be converged upon -- that is, a word whose mutant leads to more than one name; (5) the same statistic represented as a percentage; (6) the average number of names involved in a collision -- how many names the user must then choose between.

desccription of word set	avg. letters per word	# uniquely specified	# of letters required	# of collisions	% of words colliding	avg. # words per collision
108 rolodex names	6.67	108	3.4	0	0	0
361 first names	5.19	284	4.3	77	21.40	2.4
1620 technical words	10.00	1505	5.8	115	7.10	2.6
5306 NYSE stocks	14.79	5266	7.4	40	0.08	2.3
6070 last names	6.74	5272	6.4	798	13.20	2.3
30312 words (Oxford)	8.08	23418	7.6	6894	22.10	not available

The data show that it is feasible to use spelling as a fallback method of spelling even in large lists. Even in the list of over 6000 names, a unique name could be converged upon nearly 9 times out of 10. And where the name could not be completely resolved, the user had to choose between only two or three options.

Two main factors affect performance: the number of words and their average length. Although the set of technical words is larger than the set of first names, performance is better since the average length is greater. In fact, the algorithm performed almost as well on the Oxford English Dictionary, which is almost 100 times the size with an average length barely 1 1/2 times that of the list of first names.

The most impressive performance is on the set of NYSE stocks. Nearly as large as the set of last names, its company names can be spelled almost without collision due to their long length. Clearly a stock system using this algorithm to compensate for poor recognizer performance would be relieable. By the same token, names in a rolodex like the one tested could be specified with confidence.

These results are comparable to those obtained for disambiguation of names entered via a TouchTone keypad. Davis found a 5.9 percent collision for the same 6000+ name database when using Touchtones as compared to 13.2 percent as we found for speech recognition [DAVIS]. The results for the dictionary test were similar: Davis used a dictionary of 48,000 words but with the same average letters per word; he found that 11% of the words collided -- again, half the rate of this algorithm.

It is no surprise that the TouchTone error rate is smaller, for each letter is confusable with two others on a standard keypad, whereas a letter like ‘b’ is confusable with six others. Though the number of collisions for speech is double that of TouchTones, the number of words-per-collision is comparable; for both speech and TouchTones, most collisions involve two or three words.

Usability

Performance has the easy metric of convergence; usability is not so easily determined. The motivation for this paper was to make speech recognition systems more usable by providing a reliable method of specifying names when recognition fails. Indeed, few things are as exasperating as repeating a command to a speech system that just isn’t getting it. Being able to overcome errors in recognition of proper names by spelling is a boon to users.

How long it takes to spell words out, however, is another question entirely. Clearly, spelling a word takes longer than saying it, particularly when each letter is prompted individually. Spelling “Marx” discretely can take ten seconds instead of one second to say it. Still, if spelling is the only way to get the system to recognize the name, the extra time may be worth it. A study would be worthwhile to determine if this is so.

Another aspect of spelling names is dealing with collisions. Though collisions occur with varying frequency, they take extra time to deal with and drop the user into yet another modality, that of answering ‘yes’ or ‘no’ to the question “Is it <name>?” If the user has already said the name, spelled it continuously, and spelled it discretely, a fourth step may prove to be too much.

The need to explicitly or interactively disambiguate names is not unique, however. Many lists may contain duplicate entries which require disambiguation: in my rolodex, for example, I have several entries with the last name of “Marx”. Regardless of whether I speak the name, spell it out, or type it on a keyboard, I will have to explicitly disambiguate by saying *which* of the several Marxes I meant. Hence, although spelling names with a speech recognizer may require some explicit disambiguation on the part of the user, this may only increase the amount of disambiguation already required in the system. It is not an additional task.

CONCLUSION

As many speech systems will be used to let users select an item from a potentially large menu, it is essential that reliable methods of specifying an item are available. Present recognizers do not provide robust recognition where perplexity is high, and even as recognizers improve, they will be situated in noisy environments such as airports and used over noisy channels such as cellular phones which will limit their performance. Hence spelling names may be a fallback technique, but it is not a temporary one.

Spellings usually converge to a single name, and where they do not, the user’s task is usually as simple as choosing between two or maybe three alternatives. As many lists will contain duplicate entries requiring explicit disambiguation anyway, the additional effort required by the user to disambiguate collisions is unlikely to present an unreasonable burden.

Robust name specification by spelling opens up many possibilities for real-world deployment of speech recognition systems. Dialup services which require the user to specify a name, be it a city, state, stock, musical artist, or family member can perform reliably, even when recognition fails. Dictation systems can offer spelling as a means for specifying words which fail to be recognized. Speech recognition systems, supported by the fallback method of spelling misrecognized names, can match TouchTone systems for reliability while eclipsing them in usability.

ACKNOWLEDGMENTS

Thanks to Charles Hemphill at Texas Instruments for his countless hours of help with recognizer issues, and to Raja Rajasekaran for making it possible for us to use Dagger.

This work was sponsored by Sun Microsystems.

REFERENCES

[DAVIS] Davis, Jim. “Let Your Fingers do the Spelling: Implicit Disambiguation of Words Spelled With the Telephone Keypad.” *Journal of the American Voice I/O Society*, 9:57-66, March 1991.

[SPIEGEL] Spiegel, M.F. “Pronouncing Surnames Automatically.” In *Proceedings of the 1985 Conference.* San Jose, CA: American Voice I/O Society, September 1985.

[VITALE] Vitale, T. “An Algorithm for High Accuracy Name Pronunciation by a Parametric Speech Synthesizer.” *Journal of computational Linguistics*, 17(1):257-276, 1991.