Phonetic Models for Generating Spelling Variants

Rahul Bhagat and Eduard Hovy
Information Sciences Institute
University Of Southern California
4676 Admiralty Way, Marina Del Rey, CA 90292-6695
{rahul, hovy}@isi.edu

Abstract

Proper names, whether English or non-English, have several different spellings when transliterated from a non-English source language into English. Knowing the different variations can significantly improve the results of name-searches on various source texts, especially when recall is important. In this paper we propose two novel phonetic models to generate numerous candidate variant spellings of a name. Our methods show threefold improvement over the baseline and generate four times as many good name variants compared to a human while maintaining a respectable precision of 0.68.

1 Introduction

In this paper, we present a novel approach for generating variant spellings of a person's name.

Proper names in general are very important in text. Since news stories especially revolve around people, places, or organizations, proper names play a major role in helping one distinguish between a general event (like a war) and a specific event (like the Iraq war) [Raghavan and Allan, 2005].

Proper names in English are spelled in various ways. Despite the existence of one or more standard forms for someone's name, it is common to find variations in transliterations of that name in different source texts, such as Osama vs Usama. The problem is more pronounced when dealing with non-English names or when dealing with spellings by non-native speakers.

Missing some spelling variations of a name may result in omission of some useful information in a search about a person. For example, imagine a defense analyst or reporter searching English transcripts of Arabic TV news or newspaper articles for information about some person. Transcribers or different news sources, may spell the name of the person differently and some relevant information about this person may never be found. An automated spelling variant generator would help locate all potential mentions of the person in question.

Another useful application for a variant spelling generator is transliteration. Imagine a transliterator that uses a phoneme-based model to transliterate from a non-English source language to English [Knight and Graehl, 1997]. Such a transliterator is quite likely to generate a spelling that is not the most commonly used spelling of a name. In such a case, we could use the spelling variant model to match up this spelling to its more commonly accepted variant(s).

Of course generating all the possible spelling variants of a name is a next to impossible task, given that this would require one to simulate all the possible corruptions that a sequence of characters could undergo to produce a reasonable spelling. But one could approximate this by using different models.

In this paper we propose two previously unexplored methods to generate the spelling variants of a person name. Our first method is a unique phoneme-based approach using the CMU pronouncing dictionary[1]. Here we first employ the EM algorithm [Baum, 1972; Dempster et al., 1977] to learn the mappings between letters and the corresponding phonemes in both directions. We then use a noisy channel model based translator to first generate n-best phoneme sequences for a name and then use a reverse translator to generate the n-best name variants for the phoneme sequences.

Our second method is a completely unsupervised method in which we collect a list of over 7 million names (words) using a named entity extraction system applied to a large text corpus (about 10 GB plain text) collected from the web, and then use a popular sound based algorithm called Soundex [Knuth, 1973] to group together the variants in order to make them searchable.

As the baseline, we use a system based on the names list obtained from the US Census[2]. This list consists of about 89,000 last names and 5500 first names (male and female) collected by the US census bureau[3]. Given a name, we use levenshtein distance or edit distance [Hall and Dowling, 1980] to find the names that are possible variants of a given name.

To the best of our knowledge, no one has ever used a bi-directional phoneme-based approach the way we have to generate spelling variants. Raghavan and Allan [2004] have used Soundex codes to normalize names in a corpus for the purpose of story link detection task and Zobel and Dart

[1] http://www.speech.cs.cmu.edu/cgi-bin/cmudict
[2] http://www.census.gov/genealogy/names/names_files.html
[3] http://www.census.gov

[1996] have used Soundex as one of the phonetic string matching algorithms for name retrieval.

The rest of the paper is organized as follows. Section 2 discusses some related work, section 3 describes our methods in detail, section 4 describes our experiments, section 5 presents the results and section 6 discusses our conclusion and future work.

2 Related Work

The problem of spelling variants is a well studied problem in the information retrieval community, especially in the context of query expansion and word conflation. For this purpose, various methods including stemming and more complex morphological analysis have been proposed and widely used [Hull, 1996; Krovetz, 1993; Tzoukermann et al., 1997]. These methods work very well with normal English words. However they cannot be effectively applied to proper names given the difference in the nature of proper names and other English words

In the past, people have worked with proper names. Knight and Graehl [1997] proposed a cascaded finite state transducer (FST) based approach for name transliteration from Japanese to English. Virga and Khudanpur [2003] proposed a similar name transliteration approach for Chinese to English to use in cross language information retrieval. Both these approaches use sounds or pronunciations as their bridge to go from one language to the other. They however are targeted for transliteration between different languages (say Japanese to English) and hence are not applicable for generating variants in the same language. Raghavan and Allan [2005] studied the problem of proper name spelling variants in automatic speech recognition (ASR). They proposed a number of methods to match inconsistently spelled names in ASR. Their methods however are tailored for speech recognition errors and require a parallel corpus of speech and the corresponding transcriptions. Zobel and Dart [1996] used various phonetic string matching algorithms including Soundex for name retrieval. Their approach however deals only with matching phonetically similar names as against generating them.

None of these methods is suitable for our task since we want to be able to model reasonable transformations a name spelling could undergo to actually generate variant spellings for a name.

3 Generating Spelling Variants

In this section, we present two distinct and complementary approaches to generating the different spellings of a name. In the first method, we use a supervised learning approach to overgenerate a large number of variant spellings by variations in pronunciation. We call this the **Pronunciation learning method**.

In the second method we use a novel approach to first obtain a large list of names, group the similar sounding names using Soundex [Knuth, 1973], and then use Soundex to find the different spellings for a name. We call this the **List Soundex method**.

3. 1 Pronunciation Learning Method

In this method, we use the CMU pronouncing dictionary[1], a pronunciation dictionary for North American English, as the data for training our models. The CMU dictionary consists of over 125,000 English words along with their pronunciations in a fixed set of 39 phonemes.

3.1.1 Noisy Channel Model

We use the *noisy channel model*, which is commonly used in machine translation and speech recognition, as our basic model. In this framework for machine translation, the target language sentence E has supposedly been corrupted into the source language sentence F due to a noisy channel [Brown et al., 1993].

$$\underset{E}{\operatorname{argmax}}\, P(E \mid F) = \underset{E}{\operatorname{argmax}}\, P(E)*P(F \mid E) \qquad (3.1)$$

where –

P(E): represents the probability of the target language sentence E (language model)

P(F | E): represents the probability of generating the source language sentence F given the target language sentence E (translation model)

P(E | F): represents the probability of generating the target language sentence E given the source language sentence F, estimated using the noisy channel model

Similarly, in our case we assume a variant spelling of a name to be the result of a two translation processes:

1. The characters in the original name are translated into n-best sequences of phonemes
2. Phoneme sequences are translated back into n-best character sequences, thus generating the variants.

We therefore have two noisy channels, one representing the translation from a character sequence (C) to a phoneme sequence (Ph) (text to speech) and other representing the translation form a phoneme sequence back to a character sequence (speech to text).

Text to speech:

$$\underset{Ph}{\operatorname{argmax}}\, P(Ph \mid C) = \underset{Ph}{\operatorname{argmax}}\, P(Ph)*P(C \mid Ph) \qquad (3.2)$$

Speech to text:

$$\underset{C}{\operatorname{argmax}}\, P(C \mid Ph) = \underset{C}{\operatorname{argmax}}\, P(C)*P(Ph \mid C) \qquad (3.3)$$

3.1.2 Training

We apply the EM algorithm [Baum, 1972; Dempster et al., 1977] in two directions to the CMU dictionary to obtain the IBM Model-3 [Brown et al, 1993] alignments between the characters and phonemes in both the directions. This gives us the two translation models - P(Ph | C) and P(C | Ph).

To obtain the phoneme language model P(Ph) we use the 125,000+ phoneme sequences from the CMU dictionary. Using this as training data, we build a phoneme trigram language model.

To obtain the character language model P(C), we use the list of 7.3 million names that we extracted from text. Using this as training data, we build a character trigram language model.

3.1.3 Implementation

We use the GIZA++ package [Och and Ney, 2003] to train our translation models. We obtain the alignments between letters and phonemes in both the directions using GIZA++ and then based on the alignments, we build translation models for both "text to speech" and "speech to text".

We use the CMU-Cambridge statistical language modeling toolkit[4] to build the language models. We use trigram based language models for both the letters and phonemes.

Following Knight and Graehl [1997], we represent each of our language models as a weighted finite state automaton (WFSA) and each of our translation models as a weighted finite state transducer (WFST). We then use the USC/ISI finite state transducer toolkit Carmel[5] to perform the composition between the corresponding WFST and WFSA to obtain two noisy channel based WFST decoders – one going from letters to phonemes (text to speech) and the other going from phonemes to letters (speech to text).

To obtain variant spellings for a given name, we place the two noisy channel based WFST decoders in a cascade. The first generates an n-best list of pronunciations for a given input name. The second then produces an n-best list of spellings for each of the pronunciations. We then combine the n-best spellings generated by each of the pronunciations and rank the combined output by sorting it based on (a) increasing order of edit distance from the original name, (b) decreasing order of the weight a name variant gets from the decoder and (c) decreasing order of the number of times a variant is generated by the different pronunciations. Ties, if any, are broken randomly.

3.2 List Soundex Method

In this method, we use a huge database of names combined with a phonetic sound matching algorithm called Soundex to find variant spellings.

3.2.1 Creating a List of Names

The web contains many proper names. One could search and create databases of names by manually collecting various name lists like the lists for baby names, census lists etc.

We have found that trying to create name lists manually is not only a tedious task but also results in very sparsely populated lists. Even the best resource of names that we know of, the US census name list, contains fewer than 100,000 names, and those are US names only. Apart from this, hand-created lists that are meticulously prepared by experts are mostly well spelled names. We are interested in finding the common *misspellings* of names.

To overcome these limitations we decided to create a list of names automatically. Using a corpus containing about 10GB of English text gathered from the web in a previous project [Ravichandran et al., 2005], we applied the BBN IdentiFinder, a state of the art named-entity extraction system [Bikel et al., 1999], to it. We extracted all the entities that the named-entity extractor tagged as "person names". This gave us a list of about 7.3 million unique names.

[4] http://svr-www.eng.cam.ac.uk/~prc14/toolkit.html

[5] http://www.isi.edu/licensed-sw/carmel

3.2.2 Soundex Algorithm

Levenshtein distance or edit distance [Hall and Dowling, 1980] is a measure of similarity between two strings, measuring the number of insertions, deletions, and substitutions required to transform a string into the other.

Traditionally, edit distance has been used for spelling error detection and correction [Kukich, 1992]. We can use the same edit distance as a measure for finding names that are close to a given name by doing a lookup in a name list. But calculating edit distance between two strings is $O(n^2)$ in the length of the strings. It is impossible in practice to use this measure when comparing against a large list. We need to do better. We should at least initially prune our candidates down and then use edit distance on this pruned list of candidates.

The Soundex algorithm [Knuth, 1973], first patented in 1918 by Margaret O'Dell and Robert C. Russel, is an approximate string matching algorithm. It produces a coarse-grained representation for a string using six phonetic classes of human speech sounds (bilabial, labiodental, dental, alveolar, velar, and glottal). The representation consists of the first letter of the word followed by three digits that together represent the phonetic class of that word.

We use Soundex to divide our huge list of names into bins of similar sounding names and then index our list on the Soundex four-character code. Now, whenever we get a name whose variants have to be found, we can look only at the appropriate Soundex-bin instead of the whole list, thus making the search for variants possible in practice.

3.2.3 Implementation

We use the named-entity extraction system BBN Identifinder to identify named entities in our corpus. We then collect all the "person" names from the tagged corpus and find all the unique names along with their corpus frequencies. Then we run Soundex on the list of names and divide then into bins of similar sounding names. We obtain about 7000 bins, with on average 1000 names in each bin.

To obtain the variants of a name, we first find the Soundex code for that name. Then in the corresponding Soundex-bin, we find the n-best variants by sorting them based on (a) increasing order of edit distance from the original name and (b) decreasing order of frequency in the corpus. Ties, if any, are broken randomly.

4 Experiments

4.1 Experimental Setup

It is not immediately clear how one can evaluate a name spelling generator. We therefore perform experiments to measure the performance of the different methods on the spelling variants generation task.

First, we randomly select 30 US names from a list of baby names. We run each of the systems (including the baseline) on these names and generate the top 25 variants for each of these names. We then ask a human to look at the names generated by each system and mark then as good or

# of top outputs	Pronunciation learning			List Soundex			Baseline		
	Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score
5	0.97	0.28	0.39	0.80	0.24	0.32	0.43	0.18	0.21
10	0.89	0.45	0.54	0.76	0.36	0.46	0.32	0.23	0.22
15	0.78	0.47	0.54	0.74	0.50	0.55	0.26	0.27	0.21
20	0.71	0.50	0.53	0.73	0.53	0.56	0.21	0.27	0.20
25	0.68	0.52	0.54	0.68	0.58	0.58	0.19	0.27	0.19

Table 1: Results

bad variants. We use this human judgment as the measure for precision (Section 5.1). To measure recall (Section 5.1), we ask a human to generate possible variants for each of the 30 names and then compare the output of each of the systems with the name variant list generated by the human.

4.2 Baseline

The US Census list, one of the largest publicly available list of US names, contains about 89,000 last names and about 5,000 first names (male and female). Apart from the names itself, the list provides information about the frequency of each name. We use this list as the resource for building our baseline system.

For each of the 30 test names, we obtain variants by finding the top 25 names that are closest in levenshtein edit distance to the given name and sort them based on (a) increasing order of edit distance, (b) decreasing order of frequency.

5. Results

5.1 Evaluation Metrics

We use the standard Natural Language Processing measures of precision, recall, and F-score to measure the performance of each of the systems.

To obtain precision, we ask a human to look at all the outputs generated by each of the systems and mark then as correct or incorrect variants. Then the precision (P) for each system can be obtained as-

$$P = \frac{\#\text{ of correct system outputs}}{\#\text{ of system outputs}} \qquad (5.1)$$

Given that we do not have a comprehensive gold standard list of variants of a name, finding true recall is quite hard. But we can obtain recall of each system relative to a human. To do this, we ask a human to look at each of the 30 test names and generate a set of possible variants for each name. We then compare the output of each system to that of the human generated list. Then the recall (R) for each system can be obtained as-

$$R = \frac{\#\text{ of system outputs in the human list}}{\#\text{ of outputs in the human list}} \qquad (5.2)$$

The F-score (F) is the harmonic mean of the precision and recall. It is calculated as-

$$F = \frac{2 * P * R}{P + R} \qquad (5.3)$$

5.2 Result Scores and Graphs

Table 1 shows the comparison of precision, recall, and F-scores of the pronunciation generation, list Soundex, and baseline systems for different numbers of outputs. The outputs are varied from top 5 to top 25 and their effect on the macro-averaged precision, macro-averaged recall, and macro-averaged F-score is shown.

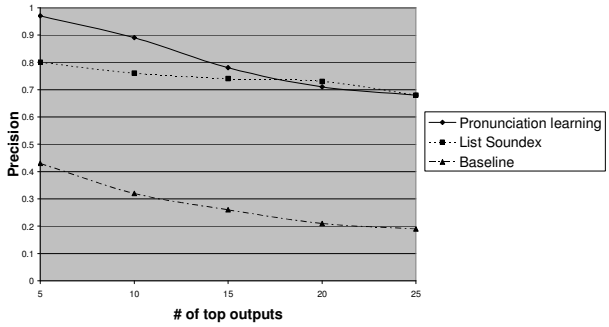

Figure 1: Precision graph

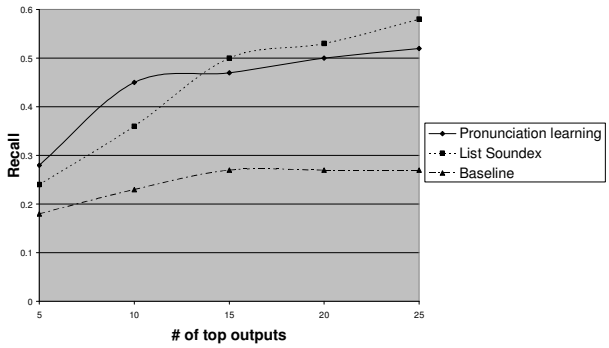

Figure 2: Recall graph

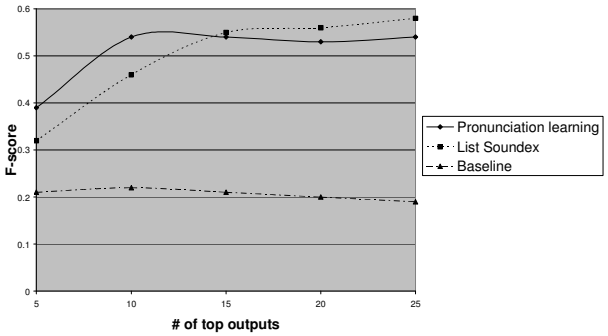

Figure 3: F-score graph

Figures 1, 2 and 3 graph the precision, recall, and F-scores respectively of the three systems corresponding to the number of outputs of the system varying from top 5 to top 25.

We find that for the top 25 name variants, the performances of the pronunciation learning and the list Soundex methods are very close to each other. The pronunciation learning method has an F-score of 0.54 while the list soundex method has an F-score of 0.58. Both the systems show roughly three-fold improvement over the baseline, which has an F-score of 0.19. The result is validated by the two-tailed paired Student's t-test [Manning and Schütze, 1999]. The t-test shows a statistically significant difference at $p<0.0005$ between the baseline and both the systems. However, it shows no statistically significant difference between the performances of the pronunciation learning and list Soundex systems themselves.

The graphs for the three systems show an intuitive trend. We find that as we increase the number of outputs that each system generates from 5 to 25, the precision of each system goes down slowly while the recall goes up.

5.3 Discussion

It is clear from the results above that the pronunciation based methods for generating name variants reap good results. Further it is clear that by using these methods, significant improvement over the baseline can be achieved.

What is also encouraging is the relative recall compared to humans. We find that for the 30 test names, a human was able to generate on an average four variant spellings. However both the pronunciation generation and list soundex methods generated 25 variants with a precision of 0.68, i.e., about 17 correct variants, which is four times what a human could generate. This of course goes in line with the general observation that humans are good at telling a good thing from a bad but are not so good at generating a set of possible good things.

What is also interesting is that the precision, recall, and F-scores of the pronunciation generation and list Soundex methods are significantly better than the baseline at all the output sizes. This came as a bit of a surprise initially, because we expected the baseline to perform well at least when the number of outputs is small, given that the baseline system uses a large meticulously prepared list of names (US census list). We however feel now that the very fact that the US census list was meticulously prepared went against the baseline given the nature of the task, where it is as important to have misspellings as to have correct spellings.

6. Conclusion

The results here clearly show the success of using the phonetic models for the purpose of generating spelling variants. We attribute this success to the fact that spelling variants more often than not have to do with the ambiguous mapping between pronunciations and their corresponding rendering into letter sequences (spellings). Our models derive their power from the very fact that different people on one hand tend to pronounce the same letter sequences differently and on the other tend to write different spellings for the same pronunciation.

One other striking observation is the four-fold increase in the number of good name variations compared to the variants generated by a human. This clearly reinforces the need of a variant generator for the name query expansion task.

Another interesting conclusion is the clear superiority of the automatic acquisition of large name lists compared to the slow manual preparation for the purpose of generating variants even in the low recall ranges (top 5 outputs), as is demonstrated by the high performance of the list Soundex method compared to the baseline.

In the future, we would like to build a hybrid system that can combine the outputs of both the algorithms and rank the combined output in a meaningful way. We would also like to test our algorithms with non-English names. Our initial experiments with non-English names look very promising though we presently do not have a baseline to compare against for these names. We would further like to apply our algorithms to other entities, such as technical terms, that are likely to be misspelled for the same reason as person names. Also, in the future we plan to modify the algorithms to tailor the variants for the kind of mistakes that are more likely to occur while spelling names from different languages and by native speakers of different languages owing to the different sounds present in those languages.

Acknowledgments

The authors would like to thank Mr. Andrew Philpot for providing the list of baby names with their variants and Dr. Patrick Pantel for his expert advice on evaluation.

References

[Baum, 1972] L.E. Baum. An Inequality and Associated Maximization Technique in Statistical Estimation of a Markov Process. *Inequalities* 3:1–8, 1972.

[Bikel et al., 1999] Daniel M. Bikel, Richard L. Schwartz and Ralph M. Weischedel. An algorithm that learns what's in a name. *Machine Learning*, vol. 34, no. 1-3, pages. 211–231, 1999.

[Brown et al., 1993] Peter Brown, Stephan Della Pietra, Vincent Della Pietra, and Robert Mercer. Mathematics of machine translation. *Computational Linguistics*, 19(2), 1993.

[Dempster et al., 1977] A.P. Dempster, N.M. Laird, and D. Rubin. Maximum-likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39(B):1–38, 1977.

[Hall and Dowling, 1980] P. Hall and G. Dowling. Approximate string matching. *Computing Surveys*, 12(4):381–402, 1980.

[Hull, 1996] D. A. Hull. Stemming algorithms: A case study for detailed evaluation. *Journal of the American Society of Information Science*, 47(1):70–84, 1996.

[Knight and Graehl, 1997] Kevin. Knight and Jonathan Graehl. Machine Transliteration. In *Proceedings of the Thirty-Fifth Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 128–135, Somerset, New Jersey, 1997. Association for Computational Linguistics.

[Krovetz, 1993] R. Krovetz. Viewing Morphology as an Inference Process,. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 191–203, 1993.

[Knuth, 1973] D. Knuth. *The Art of Computer Programming – Volume 3: Sorting and Searching*. Addison-Wesley Publishing Company, 1973

[Kukich, 1992] K. Kukich. Techniques for Automatically Correcting Words in Text. *Computing Surveys*, 24(4):377–440, Dec 1992.

[Manning and Schütze, 1999] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*, 1999. The MIT Press, Cambridge, MA.

[Och and Ney, 2003] Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational Linguistics*, 29(1):19–51, 2003.

[Raghavan and Allan, 2005] Hema Raghavan and James Allan. Matching Inconsistently Spelled Names in Automatic Speech Recognizer Output for Information Retrieval. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 451–458, Vancouver, British Columbia, Canada, October 2005. Association for Computational Linguistics.

[Raghavan and Allan, 2004] Hema Raghavan and James Allan. Using Soundex Codes for Indexing Names in ASR documents. In *Proceedings of the Workshop on Interdisciplinary Approaches to Speech Indexing and Retrieval at Humal Language Technology Conference and North American chapter of Association of Computational Linguistics*, pages 22–27, Boston, MA, USA, 2004. Association for Computational Linguistics.

[Ravichandran et al., 2005] Deepak Ravichandran, Patrick Pantel and Eduard Hovy. Randomized Algorithms and NLP: Using Locality Sensitive Hash Functions for High Speed Noun Clustering. In *Proceedings of the forty-third Annual Meeting of the Association for Computational*, pages 622–629, Ann Arbor, Michigan, USA, June 2005. Association for Computational Linguistics.

[Tzoukermann, 1997] E. Tzoukermann, J. Klavans, and C. Jacquemin. Effective use of natural language processing techniques for automatic conation of multi-word terms: The role of derivational morphology, part of speech tagging, and shallow parsing. In *Research and Development in Information Retrieval*, pages 148–155, 1997.

[Virga and Khudanpur, 2003] Paola Virga and Sanjeev Khudanpur. Transliteration of proper names in cross-language applications. In *Proceedings of the 26th ACM SIGIR conference*, pages 365–366, 2003. ACM Press.

[Zobel and Dart, 1996] Justin Zobel and Philip Dart. Phonetic String Matching: Lessons from Information Retrieval. In *Poceedings of the 19th International Conference on Research and Development in Information Retrieval*, pages 166–172, Zurich, Switzerland, 1996. ACM Press.