Hashing-based Approaches to Spelling Correction of Personal Names

Raghavendra Udupa
Microsoft Research India
Bangalore, India
raghavu@microsoft.com

Shaishav Kumar
Microsoft Research India
Bangalore, India
v-shaisk@microsoft.com

Abstract

We propose two hashing-based solutions to the problem of fast and effective personal names spelling correction in People Search applications. The key idea behind our methods is to learn hash functions that map similar names to similar (and compact) binary codewords. The two methods differ in the data they use for learning the hash functions - the first method uses a set of names in a given language/script whereas the second uses a set of bilingual names. We show that both methods give excellent retrieval performance in comparison to several baselines on two lists of misspelled personal names. More over, the method that uses bilingual data for learning hash functions gives the best performance.

1 Introduction

Over the last few years, People Search has emerged as an important search service. Unlike general Web Search and Enterprise Search where users search for information on a wide range of topics including people, products, news, events, etc., People Search is about people. Hence, personal names are used predominantly as queries in People Search. As in general Web Search, a good percentage of queries in People Search is misspelled. Naturally, spelling correction of misspelled personal names plays a very important role in not only reducing the time and effort needed by users to find people they are searching for but also in ensuring good user experience.

Spelling errors in personal names are of a different nature compared to those in general text. Long before People Search became widely popular, researchers working on the problem of personal name matching had recognized the human tendency to be inexact in recollecting names from the memory and specifying them. A study of personal names in hospital databases found that only 39% of the errors in the names were single typographical errors (Friedman and Sideli, 1992)[1]. Further, multiple and long distance typographical errors (*Gregzorz Kondrak* for *Grzegorz Kondrak*), phonetic errors (as in *Erik Bryl* for *Eric Brill*), cognitive errors (as in *Silvia Cucerzan* for *Silviu Cucerzan*) and word substitutions (as in *Rob Moore* for *Bob Moore*) are observed relatively more frequently in personal names compared to general text.

In addition to within-the-word errors, People Search queries are plagued by errors that are not usually seen in general text. The study by Friedman and Sideli discovered that 36% of the errors were due to addition or deletion of a word (as in *Ricardo Baeza* for *Ricardo Baeza-Yates*) (Friedman and Sideli, 1992). Although word addition and deletion generally do not come under the purview of spelling correction, in People Search they are important and need to be addressed.

Standard approaches to general purpose spelling correction are not well-suited for correcting misspelled personal names. As pointed out by (Cucerzan and Brill, 2004), these approaches either try to correct individual words (and will fail to correct *Him Clijsters* to *Kim Clijsters*) or employ features based on relatively wide context windows

[1] In contrast, 80% of misspelled words in general text are due to single typographical errors as found by (Damerau, 1964).

Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing, pages 1256–1265,
MIT, Massachusetts, USA, 9-11 October 2010. ©2010 Association for Computational Linguistics

which are not available for queries in Web Search and People Search. Spelling correction techniques meant for general purpose web-queries require large volumes of training data in the form of query logs for learning the error models (Cucerzan and Brill, 2004), (Ahmad and Kondrak, 2005). However, query logs are not available in some applications (e.g. Email address book search). Further, unlike general purpose web-queries where word order often matters, in People Search word order is lax (e.g. I might search for either *Kristina Toutanova* or *Toutanova Kristina*). Therefore, spelling correction techniques that rely crucially on bigram and higher order language models will fail on queries with a different word order than what is observed in the query log.

Unlike general purpose Web Search where it is not reasonable to assume the availability of a high-coverage trusted lexicon, People Search typically employs large authoritative name directories. For instance, if one is searching for a friend on Facebook, the correct spelling of the friend's name exists in the Facebook people directory[2] (assuming that the friend is a registered user of Facebook at the time of the search). Similarly, if one is searching for a contact in Enterprise address book, the correct spelling of the contact is part of the address book. In fact, even in Web Search, broad-coverage name directories are available in the form of Wikipedia, IMDB, etc. The availability of large authoritative name directories that serve as the source of trusted spellings of names throws open the possibility of correcting misspelled personal names with the help of name matching techniques (Pfeifer et al., 1996), (Christen, 2006), (Navarro et al., 2003). However, the best of the name matching techniques can at best work with a few thousand names to give acceptable response time and accuracy. They do not scale up to the needs of People Search applications where the directories can have millions of names.

In this work, we develop hashing-based name similarity search techniques and employ them for spelling correction of personal names. The motivation for using hashing as a building block of spelling correction is the following: given a query, we want to return the global best match in the name directory that exceeds a similarity threshold. As matching the query with the names in the directory is a time consuming task especially for large name directories, we solve the search problem in two stages:

- NAME_BUCKETING: For each token of the query, we do an approximate nearest neighbor search of the name tokens of the directory and produce a list of candidates, i.e., tokens that are approximate matches of the query token. Using the list of candidate tokens, we extract the list of candidate names which contain at least one approximately matching token.
- NAME_MATCHING: We do a rigorous matching of the query with candidate names.

Clearly, our success in finding the right name suggestion for the query in the NAME_MATCHING stage depends crucially on our success in getting the right name suggestion in the list of candidates produced by the NAME_BUCKETING stage search. Therefore, we need a name similarity search technique that can ensure very high recall without producing too many candidates. Hashing is best suited for this task of fast and approximate name matching. We hash the query tokens as well as directory tokens into d bit binary codes. With binary codes, finding approximate matches for a query token is as easy as finding all the database tokens that are at a Hamming distance of r or less from the query token in the binary code representation (Shakhnarovich et al., 2008), (Weiss et al., 2008). When the binary codes are compact, this search can be done in a fraction of a second on directories containing millions of names on a simple processor.

Our contributions are:

- We develop a novel data-driven technique for learning hash functions for mapping similar names to similar binary codes using a set of names in a given language/script (i.e. monolingual data). We formulate the problem of learning hash functions as an optmization problem whose relaxation can be solved as a generalized Eigenvalue problem. (Section 2.1).
- We show that hash functions can also be learnt using bilingual data in the form of name equivalents in two languages. We formulate the

[2] http://www.facebook.com/directory/people/

problem of learning hash functions as an optmization problem whose relaxation can be solved using Canonical Correlation Analysis. (Section 2.2)

- We develop new similarity measures for matching names (Section 3.1).
- We evaluate the two methods systematically and compare our performance against multiple baselines. (Section 5).

2 Learning Hash Functions

In this section, we develop two techniques for learning hash functions using names as training data. In the first approach, we use monolingual data consisting of names in a language whereas in the second we use bilingual name pairs. In both techniques, the key idea is the same: we learn hash functions that map similar names in the training data to similar codewords.

2.1 M-HASH: Learning with Monolingual Names Data

Let (s, s') be a pair of names and $w(s, s')$ be their similarity[3]. We are given a set of name pairs $T = \{(s, s')\}$ as the training data. Let $\phi(s) \in R^{d_1}$ be the feature representation of s. We want to learn a hash function f that maps each name to a d bit codeword: $f : s \mapsto \{-1, 1\}^d$. We also want the Hamming distance of the codeword of s to the codeword of s' be small when $w(s, s')$ is large. Further, we want each bit of the codewords to be either 1 or -1 with equal probablity and the successive bits of the codewords to be uncorrelated. Thus we arrive at the following optimization problem[4]:

$$
\begin{aligned}
minimize: \quad & \sum_{(s,s')\in T} w\left(s, s'\right) \left\| f(s) - f\left(s'\right) \right\|^2 \\
s.t.: \quad & \\
& \sum_{s:(s,s')\in T} f(s) = 0 \\
& \sum_{s:(s,s')\in T} f(s) f(s)^T = \rho^2 I_d \\
& f(s), f\left(s'\right) \in \{-1, 1\}^d
\end{aligned}
$$

where I_d is an identity matrix of size $d \times d$.

Note that the second constraint helps us avoid the trap of mapping all names to the same codeword and thereby making the Hamming error zero while satisfying the first and last constraints.

It can be shown that the above minimization problem is NP-hard even for 1-bit codewords (Weiss et al., 2008). Further, the optimal solution gives codewords only for the names in the training data. As we want f to be defined for all s, we address the out-of-sample extension problem by relaxing f as follows[5]:

$$f_R(s) = A^T \phi(s) = \left(a_1^T \phi(s), \ldots, a_d^T \phi(s)\right)^T \quad (1)$$

where $A = [a_1, \ldots, a_d] \in R^{d_1 \times d}$ is a rank d matrix ($d \leq d_1$).

After the linear relaxation (Equation 1), the first constraint simply means that the data be centered, i.e., have zero mean. We center Φ by subtracting the mean of Φ from every $\phi(s) \in \Phi$ to get $\hat{\Phi}$.

Subsequent to the above relaxation, we get the following optimization problem:

$$
\begin{aligned}
minimize: \quad & Tr\, A^T \hat{\Phi} L \hat{\Phi}^T A && (2)\\
s.t.: \quad & && (3)\\
& A^T \hat{\Phi} \hat{\Phi}^T A = \rho^2 I_d && (4)
\end{aligned}
$$

where L is the graph Laplacian for the similarity matrix W defined by the pairwise similarities $w(s, s')$.

The minimization problem can be transformed into a generalized Eigenvalue problem and solved efficiently using either Cholesky factorization or QZ algorithm (Golub and Van Loan, 1996):

$$\hat{\Phi} L \hat{\Phi}^T A = \hat{\Phi} \hat{\Phi}^T A \Lambda \quad (5)$$

where Λ is a $d \times d$ diagonal matrix.

Once A has been estimated from the training data, the codeword of a name s can be produced by binarizing each coordinate of $f_R(s)$:

$$f(s) = \left(sgn\left(a_1^T \phi(s)\right), \ldots, sgn\left(a_d^T \phi(s)\right)\right)^T \quad (6)$$

where $sgn(u) = 1$ if $u > 0$ and -1 otherwise for all $u \in R$.

[3] We used $1-$ length normalized Edit Distance between s and s' as $w(s, s')$.

[4] Note that the Hamming distance of a codeword y to another codeword y' is $\frac{1}{4}\|y - y'\|^2$.

[5] In contrast to our approach, Spectral Hashing, a well-known hashing technique, makes the unrealistic assumption that the training data is sampled from a multidimensional uniform distribution to address the out-of-sample extension problem (Weiss et al., 2008).

In the reminder of this work, we call the system that uses the hash function learnt from monolingual data as *M-HASH*.

2.2 B-HASH: Learning with Bilingual Names Data

Let (s,t) be a pair of name s and its transliteration equivalent t in a different language/script. We are given the set $T = \{(s,t)\}$ as the training data. Let $\phi(s) \in R^{d_1}$ (and resp. $\psi(t) \in R^{d_2}$) be the feature representation of s (and resp. t). We want to learn a pair of hash functions f, g that map names to d bit codewords: $f : s \mapsto \{-1,1\}^d$, $g : t \mapsto \{-1,1\}^d$. We also want the Hamming distance of the codeword of a name to the codeword of its transliteration be small. As in Section 2.1, we want each bit of the codewords to be either 1 or -1 with equal probablity and the successive bits of the codewords to be uncorrelated. Thus we arrive at the following optimization problem:

$$
\begin{aligned}
minimize: \quad & \sum_{(s,t)\in T} \|f(s) - g(t)\|^2 \\
s.t.: \quad & \\
& \sum_{s:(s,t)\in T} f(s) = 0 \\
& \sum_{t:(s,t)\in T} g(t) = 0 \\
& \sum_{s:(s,t)\in T} f(s) f(s)^T = \rho^2 I_d \\
& \sum_{t:(s,t)\in S} g(t) g(t)^T = \rho^2 I_d \\
& f(s), g(t) \in \{-1,1\}^d
\end{aligned}
$$

where I_d is an identity matrix of size $d \times d$.

As we want f (and resp. g) to be defined for all s (and resp. t), we relax f (and resp. g) as follows:

$$f_R(s) = A^T \phi(s) \tag{7}$$

$$g_R(t) = B^T \psi(s) \tag{8}$$

where $A = [a_1, \ldots, a_d] \in R^{d_1 \times d}$ and $B = [b_1, \ldots, b_d] \in R^{d_2 \times d}$ are rank d matrices.

As before, we center Φ and Ψ to get $\hat{\Phi}$ and $\hat{\Psi}$ respectively. Thus, we get the following optimization problem:

$$minimize: \quad Tr\ H\left(A, B; \hat{\Phi}, \hat{\Psi}\right) \tag{9}$$

$$s.t.: \tag{10}$$

$$A^T \hat{\Phi} \hat{\Phi}^T A = \rho^2 I_d \tag{11}$$

$$B^T \hat{\Psi} \hat{\Psi}^T B = \rho^2 I_d \tag{12}$$

where $H\left(A, B; \hat{\Phi}, \hat{\Psi}\right) = \left(A^T \hat{\Phi} - B^T \hat{\Psi}\right)\left(A^T \hat{\Phi} - B^T \hat{\Psi}\right)^T$.

The minimization problem can be solved as a generalized Eigenvalue problem:

$$\hat{\Phi} \hat{\Psi}^T B = \hat{\Phi} \hat{\Phi}^T A \Lambda \tag{13}$$

$$\hat{\Psi} \hat{\Phi}^T A = \hat{\Psi} \hat{\Psi}^T B \Lambda \tag{14}$$

where Λ is a $d \times d$ diagonal matrix. Further, Equations 13 and 14 find the canonical coefficients of $\hat{\Phi}$ and $\hat{\Psi}$ (Hardoon et al., 2004).

As with monolingual learning, we get the codeword of s by binarizing the coordinates of $f_R(s)$[6]:

$$f(s) = \left(sgn\left(a_1^T \phi(s)\right), \ldots, sgn\left(a_d^T \phi(s)\right)\right)^T \tag{15}$$

In the reminder of this work, we call the system that uses the hash function learnt from bilingual data as *B-HASH*.

3 Similarity Score

In this section, we develop new techniques for computing the similarity of names at token level as well as a whole. We will use these techniques in the NAME_MATCHING stage of our algorithm (Section 4.2.1).

3.1 Token-level Similarity

We use a logistic function over multiple distance measures to compute the similarity between name tokens s and s':

$$K\left(s, s'\right) = \frac{1}{1 + e^{-\sum_i \alpha_i d_i(s,s')}}. \tag{16}$$

While a variety of distance measures can be employed in Equation 16, two obvious choices

[6] As a biproduct of bilingual learning, we can hash names in the second language using g:

$$g(t) = \left(sgn\left(b_1^T \psi(t)\right), \ldots, sgn\left(b_d^T \psi(t)\right)\right)^T$$

are the normalized Damerau-Levenshtein edit distance between s and s' and the Hamming distance between the codewords of s and s' ($= \|f(s) - f(s')\|$). In our experiments, we found that the continuous relaxation $\|f_R(s) - f_R(s')\|$ was better than $\|f(s) - f(s')\|$ and hence we used it with Damerau-Levenshtein edit distance. We estimated α_1 and α_2 using a small held out set.

3.2 Multi-token Name Similarity

Let $Q = s_1 s_2 \ldots s_I$ and $D = s'_1 s'_2 \ldots s'_J$ be two multi-token names. To compute the similarity between Q and D, we first form a weighted bipartite graph with a node for each s_i and a node for each s'_j and set edge weight to $K\left(s_i, s'_j\right)$. We then compute the weight (κ_{max}) of the maximum weighted matching[7] in this graph. The similarity between Q and D is then computed as

$$K(Q, D) = \frac{\kappa_{max}}{|I - J + 1|}. \quad (17)$$

4 Spelling Correction using Hashing

In this section, we describe our algorithm for spelling correction using hashing as a building block.

4.1 Indexing the Name Directory

Given a name directory, we break each name into its constituent tokens and form a set of distinct name tokens. Using the name tokens and the original names, we build an inverted index which, for each name token, lists all the names that have the token as a constituent. Further, we hash each name token into a d bit codeword as described in Equation 6 (and resp. Equation 15) when using the hash function learnt on monolingual data (and resp. bilingual data) and store in a hash table.

4.2 Querying the Name Directory

Querying is done in two stages: NAME_BUCKETING and NAME_MATCHING.

[7] In practice, a maximal matching computed using a greedy approach suffices since many of the edges in the bipartite graph have low weight.

4.2.1 Name Bucketing

Given a query $Q = s_1 s_2 \ldots s_I$, we hash each s_i into a codeword y_i and retrieve all codewords in the hash table that are at a Hamming distance of r or less from y_i. We rank the name tokens thus retrieved using the token level similarity score of Section 3.1 and retain only the top 100. Using the top tokens, we get all names which contain any of the name tokens as a constituent to form the pool of candidates $\mathcal{C}$ for the NAME_MATCHING stage.

4.2.2 Name Matching

First we find the best match for a query Q in the set of candidates $\mathcal{C}$ as follows:

$$D^* = \underset{D \in \mathcal{C}}{\operatorname{argmax}} K(Q, D). \quad (18)$$

Next we suggest D^* as the correction for Q if $K(Q, D^*)$ exceeds a certain empirically determined threshold.

5 Experiments and Results

We now discuss the experiments we conducted to study the retrieval performance of the two hashing-based approaches developed in the previous sections. Apart from evaluating the systems on test sets using different name directories, we were interested in comparing our systems with several baselines, understanding the effect of some of the choices we made (e.g. training data size, conjugate language) and comparative analysis of retrieval performance on queries of different complexity.

5.1 Experimental Setup

We tested the proposed hashing-based spelling correction algorithms on two test sets:

- DUMBTIONARY: 1231 misspellings of various names from *Dumbtionary*[8] and a name directory consisting of about $550,000$ names gleaned from the English Wikipedia. Each of the misspellings had a correct spelling in the name directory.
- INTRANET: 200 misspellings of employees taken from the search logs of the intranet of a large organization and a name directory

[8] http://www.dumbtionary.com

consisting of about $150,000$ employee names. Each of the misspellings had a correct spelling in the name directory.

Table 1 shows the average edit distance of a misspelling from the correct name. Compared to DUMBTIONARY, the misspellings in INTRANET are more severe as the relatively high edit distance indicates. Thus, INTRANET represents very hard cases for spelling correction.

Test Set	Average	Std. Dev.
DUMBTIONARY	1.39	0.76
INTRANET	2.33	1.60

Table 1: Edit distance of a misspelling from the correct name.

5.1.1 Training

For M-HASH, we used 30,000 single token names in English (sampled from the list of names in the Internet Movie Database[9]) as training data and for B-HASH we used 14,941 parallel single token names in English-Hindi [10]. Each name was represented as a feature vector over character bigrams. Thus, the name token *Klein* has the bigrams $\{\cdot k, kl, le, ei, in, n\cdot\}$ as the features.

We learnt the hash functions from the training data by solving the generalized Eigenvalue problems of Sections 2.1 and 2.2. For both M-HASH and B-HASH we used the top 32 Eigenvectors to form the hash function resulting in a 32 bit representation for every name token[11].

5.1.2 Performance Metric

We measured the performance of all the systems using Precision@1, the fraction of names for which a correct spelling was suggested at Rank 1.

5.1.3 Baselines

The baselines are two popular search engines (S1 and S2), Double Metaphone (DM), a widely used phonetic search algorithm (Philips, 2000) and BM25, a very popular Information Retrieval algorithm (Manning et al., 2008). To use BM25 algorithm for spelling correction, we represented each name as a bag of bigrams and set the parameters K and b to 2 and 0.75 respectively.

[9] http://www.imdb.com

[10] We obtained the names from the organizers of NEWS2009 workshop (http://www.acl-ijcnlp-2009.org/workshops/NEWS2009/pages/sharedtask.html).

[11] We experimented with codewords of various lengths and found that the 32 bit representation gave the best tradeoff between retrieval accuracy and speed.

5.2 Results

5.2.1 DUMBTIONARY

Table 2 compares the results of the hashing-based systems with the baselines on DUMBTIONARY. As the misspellings in DUMBTIONARY are relatively easier to correct, all the systems give reasonably good retrieval results. Nevertheless, the results of M-HASH and B-HASH are substantially better than the baselines. M-HASH reduced the error over the best baseline (S1) by 13.04% whereas B-HASH reduced by 46.17% (Table 6).

M-HASH	B-HASH	S1	S2	DM	BM25
87.93	**92.53**	86.12	79.33	78.95	84.70

Table 2: Precision@1 of the various systems on DUMBTIONARY.

To get a deeper understanding of the retrieval performance of the various systems, we studied queries of varying complexity of misspelling. Table 3 compares the results of our systems with S1 for queries that are at various edit distances from the correct names. We observe that M-HASH and B-HASH are better than S1 in dealing with relatively less severe misspellings. More interestingly, B-HASH is consistently and significantly better than S1 even when the misspellings are severe.

Distance	M-HASH	B-HASH	S1
1	96.18	**96.55**	89.59
2	81.79	**87.42**	75.76
3	44.07	**67.80**	59.65
4	21.05	**31.58**	29.42
5	0.00	**37.50**	0.00

Table 3: Precision@1 for queries at various edit distances on DUMBTIONARY.

5.2.2 INTRANET

For INTRANET, search engines could not be used as baselines and therefore we compare our systems

with Double Metaphone and BM25 in Table 4. We observe that both M-HASH and B-HASH give signficantly better retrieval results than the baselines. M-HASH reduced the error by 36.20% over Double Metaphone whereas B-HASH reduced it by 51.73%. Relative to BM25, M-HASH reduced the error by 31.87% whereas B-HASH reduced it by 48.44%.

M-HASH	B-HASH	DM	BM25
70.65	**77.79**	54.00	56.92

Table 4: Precision@1 of the various systems on INTRANET.

Table 5 shows the results of our systems for queries that are at various edit distances from the correct names. We observe that the retrieval results for each category of queries are consistent with the results on DUMBTIONARY. As before, B-HASH gives signficantly better results than M-HASH.

Distance	M-HASH	B-HASH
1	82.76	**87.93**
2	57.14	**72.86**
3	34.29	**65.71**
4	38.46	**53.85**
5	6.67	**26.67**

Table 5: Precision@1 for queries at various edit distances on INTRANET.

Test Set	M-HASH	B-HASH
DUMBTIONARY	13.04	**46.17**
INTRANET	36.20	**51.73**

Table 6: Percentage error reduction over the best baseline.

5.2.3 Effect of Training Data Size

As both M-HASH and B-HASH are data driven systems, the effect of training data size on retrieval performance is important to study. Table 7 compares the results for systems trained with various amounts of training data on DUMBTIONARY. B-HASH trained with just 1000 name pairs gives 95.5% of the performance of B-HASH trained with 15000 name pairs. Similarly, M-HASH trained with 1000 names gives 98.5% of the performance of M-HASH trained with 30000 name pairs. This is probably because the spelling mistakes in DUMBTIONARY are relatively easy to correct.

Table 8 shows the results on INTRANET. We see that increase in the size of training data brings substantial returns for B-HASH. In contrast, M-HASH gives the best results at 5000 and does not seem to benefit from additional training data.

Size	M-HASH	B-HASH
1000	86.60	**88.34**
5000	87.36	**91.13**
10000	86.96	**92.53**
15000	87.19	**92.20**
30000	87.93	-

Table 7: Precision@1 on DUMBTIONARY as a function of training data size.

Size	M-HASH	B-HASH
1000	**66.04**	66.03
5000	70.65	**72.67**
10000	68.09	**75.26**
15000	68.60	**77.79**
30000	65.40	-

Table 8: Precision@1 on INTRANET as a function of training data size.

5.2.4 Effect of Conjugate Language

In Sections 5.2.1 and 5.2.2, we saw that bilingual data gives substantially better results than monolingual data. In the experiments with bilingual data, we used English-Hindi data for training B-HASH. A natural question to ask is what happens when we use someother language, say Hebrew or Russian or Tamil, instead of Hindi. In other words, does the retrieval performance, on an average, vary substantially with the conjugate language?

Table 9 compares the results on DUMBTIONARY when B-HASH was trained using English-Hindi, English-Hebrew, English-Russian, and English-Tamil data. We see that the retrieval results are good despite the differences in the script and language. Clearly, the source language (English in our experiments) benefits from being paired with any target language. However, some languages seem

to give substantially better results than others when used as the conjugate language. For instance, Hindi as a conjugate for English seems to be better than Tamil. At the time of writing this paper, we do not know the reason for this behavior. We believe that a combination of factors including feature representation, training data, and language-specific confusion matrix need to be studied in greater depth to say anything conclusively about conjugate languages.

Conjugate	DUMBTIONARY	INTRANET
Hindi	92.53	77.79
Hebrew	91.30	71.68
Russian	89.42	64.94
Tamil	90.48	69.12

Table 9: Precision@1 of B-HASH for various conjugate languages.

5.2.5 Error Analysis

We looked at cases where either M-HASH or B-HASH (or both) failed to suggest the correct spelling. It turns out that in the DUMBTIONARY test set, for 81 misspelled names, both M-HASH and B-HASH failed to suggest the correct name at rank 1. Similarly, in the case of INTRANET test set, both M-HASH and B-HASH failed to suggest the correct name at rank 1 for 47 queries. This suggests that queries that are difficult for one system are also in general difficult for the other system. However, B-HASH was able to suggest correct names for some of the queries where M-HASH failed. In fact, in the INTRANET test set, whenever B-HASH failed, M-HASH also failed. And interestingly, in the DUMBTIONARY test set, the average edit distance of the query and the correct name for the cases where M-HASH failed to get the correct name in top 10 while B-HASH got it at rank 1 was 2.96. This could be because M-HASH attempts to map names with smaller edit distances to similar codewords.

Table 10 shows some interesting cases we found during error analysis. For the first query, M-HASH suggested the correct name whereas B-HASH did not. For the second query, both M-HASH and B-HASH suggested the correct name. And for the third query, B-HASH suggested the correct name whereas M-HASH did not.

Query	M-HASH	B-HASH
John Tiler	**John Tyler**	John Tilley
Ddear Dragba	**Didear Drogba**	**Didear Drogba**
James Pol	James Poe	**James Polk**

Table 10: Error Analysis.

5.3 Query Response Time

The average query response time is a measure of the speed of a system and is an important factor in real deployments of a Spelling Correction system. Ideally, one would like the average query response time to be as small as possible. However, in practice, average query response time is not only a function of the algorithm's computational complexity but also the computational infrastructure supporting the system. In our expriments, we used a single threaded implementation of M-HASH and B-HASH on an Intel Xeon processor (2.86 GHz). Table 11 shows the average query response time. We note that M-HASH is substantially slower than B-HASH. This is because the number of collisions in the NAME_BUCKETING stage is higher for M-HASH.

We would like to point out that both NAME_BUCKETING and NAME_MATCHING stages can be multi-threaded on a multi-core machine and the query response time can be decreased by an order easily. Further, the memory footprint of the system is very small and the codewords require 4.1 MB for the employees name directory (150,000 names) and 13.8 MB for the Wikipedia name directory (550,000 names).

Test Set	MHASH	BHASH
DUMBTIONARY	190	87
INTRANET	148	75

Table 11: Average response time in milliseconds (single threaded system running on 2.86 GHz Intel Xeon Processor).

6 Related Work

Spelling correction of written text is a well studied problem (Kukich, 1996), (Jurafsky and Martin, 2008). The first approach to spelling correc-

tion made use of a lexicon to correct out-of-lexicon terms by finding the closest in-lexicon word (Damerau, 1964). The similarity between a misspelled word and an in-lexicon word was measured using Edit Distance (Jurafsky and Martin, 2008). The next class of approaches applied the noisy channel model to correct single word spelling errors (Kernighan et al., 1990), (Brill and Moore, 2000). A major flaw of single word spelling correction algorithms is they do not make use of the context of the word in correcting the errors. The next stream of approaches explored ways of exploiting the word's context (Golding and Roth, 1996), (Cucerzan and Brill, 2004). Recently, several works have leveraged the Web for improved spelling correction (Chen et al., 2007),(Islam and Inkpen, 2009), (Whitelaw et al., 2009). Spelling correction algorithms targeted for web-search queries have been developed making use of query logs and click-thru data (Cucerzan and Brill, 2004), (Ahmad and Kondrak, 2005), (Sun et al., 2010). None of these approaches focus exclusively on correcting name misspellings.

Name matching techniques have been studied in the context of database record deduplication, text mining, and information retrieval (Christen, 2006), (Pfeifer et al., 1996). Most techniques use one or more measures of phonetic similarity and/or string similarity. The popular phonetic similarity-based techniques are Soundex, Phonix, and Metaphone (Pfeifer et al., 1996). Some of the string similarity-based techniques employ Damerau-Levenshtein edit distance, Jaro distance or Winkler distance (Christen, 2006). Data driven approaches for learning edit distance have also been proposed (Ristad and Yianilos, 1996). Most of these techniques either give poor retrieval performance on large name directories or do not scale.

Hashing techniques for similarity search is also a well studied problem (Shakhnarovich et al., 2008). Locality Sensitive Hashing (LSH) is a theoretically grounded data-oblivious approach for using random projections to define the hash functions for data objects with a single view (Charikar, 2002), (Andoni and Indyk, 2006). Although LSH guarantees that asymptotically the Hamming distance between the codewords approaches the Euclidean distance between the data objects, it is known to produce long codewords making it practically inefficient. Recently data-aware approaches that employ Machine Learning techniques to learn hash functions have been proposed and shown to be a lot more effective than LSH on both synthetic and real data. Semantic Hashing employs Restricted Boltzmann Machine to produce more compact codes than LSH (Salakhutdinov and Hinton, 2009). Spectral Hashing formalizes the requirements for a good code and relates them to the problem of balanced graph partitioning which is known to be NP hard (Weiss et al., 2008). To give a practical algorithm for hashing, Spectral Hashing assumes that the data are sampled from a multidimensional uniform distribution and solves a relaxed partitioning problem.

7 Conclusions

We developed two hashing-based techniques for spelling correction of person names in People Search applications.To the best of our knowledge, these are the first techniques that focus exclusively on correcting spelling mistakes in person names. Our approach has several advantages over other spelling correction techniques. Firstly, we do not suggest incorrect suggestions for valid queries unlike (Cucerzan and Brill, 2004). Further, as we suggest spellings from only authoritative name directories, the suggestions are always well formed and coherent. Secondly, we do not require query logs and other resources that are not easily available unlike (Cucerzan and Brill, 2004), (Ahmad and Kondrak, 2005). Neither do we require pairs of misspelled names and their correct spellings for learning the error model unlike (Brill and Moore, 2000) or large-coverage general purpose lexicon for unlike (Cucerzan and Brill, 2004) or pronunciation dictionaries unlike (Toutanova and Moore, 2002). Thirdly, we correct the query as a whole unlike (Ahmad and Kondrak, 2005) and can handle word order changes unlike (Cucerzan and Brill, 2004). Fourthly, we do not iteratively process misspelled name unlike (Cucerzan and Brill, 2004). Fifthly, we handle large name directories efficiently unlike the spectrum of name matching techniques discussed in (Pfeifer et al., 1996). Finally, our training data requirement is relatively small.

As future work, we would like to explore the possibility of learning hash functions using 1) bilingual

and monolingual data together and 2) multiple conjugate languages.

References

Farooq Ahmad and Grzegorz Kondrak. 2005. Learning a spelling error model from search query logs. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 955–962, Morristown, NJ, USA. Association for Computational Linguistics.

Alexandr Andoni and Piotr Indyk. 2006. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *FOCS*, pages 459–468.

Rahul Bhagat and Eduard H. Hovy. 2007. Phonetic models for generating spelling variants. In *IJCAI*, pages 1570–1575.

Mikhail Bilenko, Raymond J. Mooney, William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. 2003. Adaptive name matching in information integration. *IEEE Intelligent Systems*, 18(5):16–23.

E. Brill and R. Moore. 2000. An improved error model for noisy channel spelling correction. In *Proceedings of ACL '00*, pages 286–293.

Moses Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388.

Qing Chen, Mu Li, and Ming Zhou. 2007. Improving query spelling correction using web search results. In *EMNLP-CoNLL*, pages 181–189.

P. Christen. 2006. A comparison of personal name matching: techniques and practical issues. *Technical Report TR-CS-06-02, Dept. of Computer Science, ANU, Canberra*.

William W. Cohen, Pradeep D. Ravikumar, and Stephen E. Fienberg. 2003. A comparison of string distance metrics for name-matching tasks. In *IIWeb*, pages 73–78.

S Cucerzan and E. Brill. 2004. Spelling correction as an iterative process that exploits the collective knowledge of web users. In *Proceedings of EMNLP '04*, pages 293–300.

F.J. Damerau. 1964. A technique for computer detection and correction of spelling errors. *Communications of ACM*, 7(3):171–176.

C. Friedman and R. Sideli. 1992. Tolerating spelling errors during patient validation. *Computers and Biomedical Research*, 25:486–509.

Andrew R. Golding and Dan Roth. 1996. Applying winnow to context-sensitive spelling correction. *CoRR*, cmp-lg/9607024.

Gene H. Golub and Charles F. Van Loan. 1996. *Matrix Computations*. Johns Hopkins University Press, Baltimore, MD, 3rd edition.

David R. Hardoon, Sándor Szedmák, and John Shawe-Taylor. 2004. Canonical correlation analysis: An overview with application to learning methods. *Neural Computation*, 16(12):2639–2664.

Aminul Islam and Diana Inkpen. 2009. Real-word spelling correction using google web 1tn-gram data set. In *CIKM*, pages 1689–1692.

D. Jurafsky and J.H. Martin. 2008. *Speech and Language Processing*. Prentice-Hall.

Mark D. Kernighan, Kenneth W. Church, and William A. Gale. 1990. A spelling correction program based on a noisy channel model. In *COLING*, pages 205–210.

K. Kukich. 1996. Techniques for automatically correcting words in a text. *Computing Surveys*, 24(4):377–439.

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. 2008. *Introduction to Information Retrieval*. Cambridge University Press.

G. Navarro, R. Baeza-Yates, and J. Azevedo-Arcoverde. 2003. Matchsimile: a flexible approximate matching tool for searching proper names. *Journal of the American Society for Information Science and Technology*, 54(1):3–15.

U. Pfeifer, T. Poersch, and N. Fuhr. 1996. Retrieval effectiveness of proper name search methods. *Information Processing and Management*, 32(6):667–679.

L. Philips. 2000. The double metaphone search algorithm. *C/C++ Users Journal*.

Eric Sven Ristad and Peter N. Yianilos. 1996. Learning string edit distance. *CoRR*, cmp-lg/9610005.

Ruslan Salakhutdinov and Geoffrey E. Hinton. 2009. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978.

Gregory Shakhnarovich, Trevor Darrell, and Piotr Indyk. 2008. Nearest-neighbor methods in learning and vision. *IEEE Transactions on Neural Networks*, 19(2):377–377.

Xu Sun, Jianfeng Gao, Daniel Micol, and Chris Quirk. 2010. Learning phrase-based spelling error models from clickthrough data. In *Proceedings of ACL 2010*.

K. Toutanova and R. Moore. 2002. Pronounciation modeling for improved spelling correction. In *Proceedings of ACL '02*, pages 141–151.

Yair Weiss, Antonio B. Torralba, and Robert Fergus. 2008. Spectral hashing. In *NIPS*, pages 1753–1760.

Casey Whitelaw, Ben Hutchinson, Grace Chung, and Ged Ellis. 2009. Using the web for language independent spellchecking and autocorrection. In *EMNLP*, pages 890–899.