Learning Hash Functions for Cross-View Similarity Search

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

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

Abstract

Many applications in Multilingual and Multimodal Information Access involve searching large databases of high dimensional data objects with multiple (conditionally independent) views. In this work we consider the problem of learning hash functions for similarity search across the views for such applications. We propose a principled method for learning a hash function for each view given a set of multiview training data objects. The hash functions map similar objects to similar codes across the views thus enabling cross-view similarity search. We present results from an extensive empirical study of the proposed approach which demonstrate its effectiveness on Japanese language People Search and Multilingual People Search problems.

1 Introduction

In many applications in Multilingual and Multimodal Information Access, the database comprises of data objects with multiple (conditionally independent) views and there is a need for similarity search across the views. Consider, for instance, the Japanese language People Search problem where the names in the people directory are represented in one or more scripts[1], namely Kanji, Katakana, Hiragana, and Romaji. Here, the representation of a name in a specific script can be regarded as a view and thus each record in the people directory contains multiple views of the same name along with other information. In Japan, users can query the people directory in any of the four scripts and expect results in all of them. This is an example of a real world Multilingual Information Access problem that involves cross-view similarity search on large databases of multiview data objects[2].

[1] http://en.wikipedia.org/wiki/Japanese_script

[2] What makes the problem challenging is the fact that there is no one-to-one mapping between Kanji characters and the Kana and Latin characters. While Kana characters are small in number and are phonetic, there are over $10,000$ Kanji characters and many of them have multiple sounds associated with them. The correct sound of a Kanji character in a name depends upon the neighboring Kanji characters in the name [Suzuki *et al.*, 2009].

A more general People Search problem is the Multilingual People Search problem where there are not only multiple scripts involved but also mutliple languages. In this problem, name records can be in one or more languages and the query can be in any of these languages [Udupa and Khapra, 2010]. The representation of a name in a specific language can be seen as a view. Users of Multilingual People Search applications expect results in any of the languages in their profile.

As a second example of a real-world problem where cross-view similarity search plays a prominent role, consider the problem of content-based multimedia retrieval where the database consists of text documents and images and the query can be either of these [Larsen *et al.*, 2003]. The image and its associated text are the two views of the same semantic object and users expect similar images and texts as results for their queries.

In both of the problems discussed above, and also in many other problems, the database consists of a large number of high dimensional data objects with multiple views. Individual data objects in the database as well as the query might have one or more of the views unspecified. Further, similarity search across views is absolutely necessary for satisfying the information need of the user.

A possible line of attack for cross-view similarity search is to first "translate" each of the views of a multiview data object to one of the views and then employ single-view similarity search techniques. For example, we could first translate a name in Kanji to Hiragana and then use the translated name in Hiragana similarity search. However, such an approach suffers from at least two major flaws as discussed previously in [Udupa and Khapra, 2010; Platt *et al.*, 2010]. Firstly, translation of one view to another is application specific, approximate, and lossy. As a consequence, the effectiveness of cross-view similarity search goes down substantially because of translation [Udupa and Khapra, 2010]. Secondly, translation is in general very slow making it impractical in many real world applications [Platt *et al.*, 2010].

In this work, we take a hashing-based approach for solving the cross-view similarity search problem. We represent each view of a multiview data object as a compact binary codeword. To support cross-view similarity search, we need the codewords of a data object to be similar if not identical. Further, codewords of similar data objects should also be similar. Assuming that we can somehow map data objects

to binary codewords, cross-view similarity search can be reduced to the much simpler problem of retrieving all data objects with codewords within a small Hamming distance of the codeword(s) for the query. The latter problem can be solved very efficiently and even databases with millions of data objects can be searched in a fraction of a second on commodity processors [Weiss *et al.*, 2008].

The main challenge in hashing-based approach to cross-view similarity search is to learn a hash function for each view of multiview data objects from training data consisting of multiview data objects. The hash functions should have the following properties:

1. They should map the different views of a multiview data object to similar codewords and similar data objects to similar codewords so that given the codeword for any of the views of a data object as the query, we can retrieve similar data objects using the codeword.
2. They should be defined everywhere so that they may be applied on not only training data objects but also to those which are not present in the training set.
3. They should produce compact codewords for each of the views of the data objects so that similarity search using the codewords is fast and scalable.
4. They should be computationally efficient and parallelizable so that they may be used in real-world cross-view similarity search applications with strict latency constraints.
5. Learning them from training data should be computationally efficient.

We give a principled solution for the problem of learning hash functions for cross-view similarity search. Our approach produces hash functions that have all the properties discussed earlier. To the best of our knowledge, ours is the first attempt towards this end.

We formulate the problem of learning hash functions as a constrained minimization problem over the training data objects. The key idea is to find hash functions that minimize the weighted average Hamming distance of the codewords for the training objects over all the views (Property 1). We seek hash functions of explict form: the hash function for each view consists of a set of $d > 0$ 1-bit hash functions of the form $sgn\left(\langle a, x\rangle\right)$ [3]. As the hash functions are linear proejctions followed by binarization, they are defined everywhere (Property 2). The computational complexity of evaluating a hash function is linear in the dimensions of the corresponding view and hence very efficient in practice (Property 4). Further, linear projections can be easily parallelized. The resulting minimization problem is tractable and can be solved as a generalized eigenvalue problem (Property 5).

We show the effectiveness of our approach on two cross-view similarity search problems:

- Japanese Language People Search
- Multilingual People Search

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

The rest of the paper is organized as follows. In Section 2, we formulate the problem of learning hash functions as an optimization problem. As the resulting problem is NP-hard, we propose a novel relaxation in Section 2.1 which transforms the problem into a tractable problem. We discuss how to solve the optimization problem in Section 2.2 and the resulting hash functions in Section 2.3. In Section 3, we discuss a couple of interesting special cases of the general cross-view similarity search problem and the connections to some well-known dimensionality reduction techniques. In Section 4, we present the results from our experiments.

2 Learning Hash Functions

Let $O = \{o_i\}_{i=1}^n$ be a set of multi-view data objects and $x_i^{(k)}$ be the k^{th} view of the object o_i, where $x_i^{(k)} \in R^{d_k}$. Let W be the similarity matrix for O with W_{ij} being the similarity between o_i and o_j. Given O and W as input[4], we want to learn a hash function $f^{(k)} : x^{(k)} \mapsto \{-1, 1\}^d$ for the kth view, $1 \leq k \leq K$. For the sake of notational simplicity, we denote $f^{(k)}\left(x^{(k)}\right)$ by $y^{(k)}$ in the remainder of this paper. Further, let $d\left(y, y'\right)$ denote the Hamming distance between the codewords y and y'[5].

As our goal is to enable cross-view similarity search through hashing, we want the hash functions to map similar objects to similar codewords over all the views. More specifically, if o_i and o_j are two similar data objects, we would like each of the hash functions $f^{(k)}$ to map o_i and o_j to similar codewords. Now, the Hamming distance between the codewords of o_i and o_j summed over all the views is

$$d_{ij} = \sum_{k=1}^{K} d\left(y_i^{(k)}, y_j^{(k)}\right) + \sum_{k=1}^{K}\sum_{k'>k}^{K} d\left(y_i^{(k)}, y_j^{(k')}\right) \quad (1)$$

We seek hash functions that minimize the similarity weighted Hamming distance between the codewords of the training data objects. Further, along the lines of single-view hashing, we impose a couple of constraints: first, we want each bit to have an equal chance of being 1 or -1; second, we require the bits to be uncorrelated. Thus, we arrive at the following problem which is a generalization of the well-known single-view hashing technique, *Spectral Hashing* [Weiss *et al.*, 2008], to multiview data objects:

$$minimize: \quad \bar{d} = \sum_{i=1}^{n}\sum_{j=1}^{n} W_{ij} d_{ij} \quad (2)$$

$$subject\ to: \quad Y^{(k)} e = 0, for\ k = 1, \ldots, K \quad (3)$$

$$\frac{1}{n} Y^{(k)} Y^{(k)^T} = I_d, for\ k = 1, \ldots, K \quad (4)$$

$$Y_{ij}^{(k)} \in \{-1, 1\}, for\ k = 1, \ldots, K \quad (5)$$

where e is a $n \times 1$ vector of all 1s and I_d is an identity matrix of size $d \times d$.

[4] When W is not available, we assume that $W = I_n$.

[5] Note that $d\left(y, y'\right) = \frac{1}{4}\left\|y - y'\right\|^2$.

From Equations 1 and 2, it follows easily that

$$\bar{d} = \sum_{k=1}^{K} Tr\left(Y^{(k)} L' {Y^{(k)}}^T\right) - 2\sum_{k=1}^{K}\sum_{k'>k}^{K} Tr\left(Y^{(k)} W {Y^{(k')}}^T\right) \quad (6)$$

where $L' = 2L + (K-1)D$, D is a diagonal matrix such that $D_{ii} = \sum_{j=1}^{n} D_{ij}$ and $L = D - W$ is the Laplacian. Note that $\bar{d}$ is a convex function of Y.

2.1 Linear Relaxation

The optimization problem described in the previous section is NP hard as it reduces trivially to the optimization problem of Spectral Hashing when $K = 1$ and the latter is known to be NP hard [Weiss *et al.*, 2008]. Unlike Spectral Partitioning which solves a relaxed problem by removing the constraint that $Y_{ij} \in \{1, -1\}$ and extends the solution to out-of-sample datapoints by assuming that the data objects have been sampled from a multidimensional uniform distribution, we take a different approach. We assume that $y_i^{(k)}$ is a low-dimensional linear embedding of $x_i^{(k)}$ but make no assumption on the distribution of the data objects:

$$y_i^{(k)} = {A^{(k)}}^T x_i^{(k)} \quad (7)$$

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

The above relaxation transforms the NP hard optimization problem of Equation 2 into the following tractable problem that can be solved as a generalized eigenvalue problem as we will see in Section 2.2:

$$minimize: \quad \bar{d} \quad (8)$$

$$subject\ to: \quad X^{(k)} e = 0, for\ k = 1, \ldots, K \quad (9)$$

$$\frac{1}{n} {A^{(k)}}^T X^{(k)} {X^{(k)}}^T A^{(k)} = I_d \quad (10)$$

where

$$\bar{d} = \sum_{k=1}^{K} Tr\left({A^{(k)}}^T X^{(k)} L' {X^{(k)}}^T A^{(k)}\right) - 2\sum_{k=1}^{K}\sum_{k'>k}^{K} Tr\left({A^{(k)}}^T X^{(k)} W {X^{(k')}}^T A^{(k')}\right).$$

Note that $\bar{d}$ is convex in $A^{(k)},\ k = 1, \ldots, K$.

The constraint in Equation 9 simply means that the data objects should have zero mean in each of the views. This can be easily ensured by centering the data objects.

2.2 Solution

In the previous section, we transformed the problem of learning hash functions into a parameter estimation problem. To estimate the parameters $A^{(k)}$, we set each of the partial derivatives of $\bar{d} - Tr\left({A^{(k)}}^T X^{(k)} {X^{(k)}}^T A^{(k)} \Lambda\right)$ to 0 where Λ is a $d \times d$ diagonal matrix. This results in the following generalized eigenvalue problem which can be solved in polynomial time [Golub and Van Loan, 1996]:

$$X^{(k)} L' {X^{(k)}}^T A^{(k)} - \sum_{k' \neq k}^{K} X^{(k)} W {X^{(k')}}^T A^{(k')} = X^{(k)} {X^{(k)}}^T A^{(k)} \Lambda \quad (11)$$

2.3 Hash Functions

The eigenvectors obtained by solving the generalized eigenvalue problem of Equation 11 are the linear projections used in our hash functions. The individual bits of the codeword are obtained by first projecting the view on to the different eigenvectors and then binarizing the projected value:

$$f^{(k)}\left(x^{(k)}\right) = \left(sgn\left(\left\langle a_1^{(k)}, x^{(k)} \right\rangle\right), \ldots, sgn\left(\left\langle a_d^{(k)}, x^{(k)} \right\rangle\right)\right)^T \quad (12)$$

3 Interesting Special Cases

We now look at two interesting special cases of the cross-view similarity search problem: $K = 1$ and $W = I_{n \times n}$.

3.1 $K = 1$

This is the special case where the data objects have only one view. Cross-view similarity search reduces to single-view similarity search in this case. When $K = 1$, we have $L' = 2L$ and therefore,

$$X^{(1)} L {X^{(1)}}^T A^{(1)} = X^{(1)} {X^{(1)}}^T A^{(1)} \Lambda. \quad (13)$$

The generalized eigenvalue problem in Equation 13 is similar in structure to Locality Preserving Indexing [Cai *et al.*, 2007]. However, the latter is primarily interested in finding a low-dimensional embedding and not in learning hash functions.

3.2 $W = I_{n \times n}$

This is the special case where the training data consists of multiview data objects but the affinity matrix is not available.

When $W = I_{n \times n}$ we have $D = I_{n \times n}$ and $L = 0$. Therefore $L' = (K-1) I_{n \times n}$. Substituting this in Equation 11, we get:

$$\sum_{k' \neq k}^{K} X^{(k)} {X^{(k')}}^T A^{(k')} = X^{(k)} {X^{(k)}}^T A^{(k)} \Lambda' \quad (14)$$

where $\Lambda' = (K-1) I_{d \times d} - \Lambda$.

Further, when $K = 2$, we get:

$$X^{(1)} {X^{(2)}}^T A^{(2)} = X^{(1)} {X^{(1)}}^T A^{(1)} \Lambda' \quad (15)$$

$$X^{(2)} {X^{(1)}}^T A^{(1)} = X^{(2)} {X^{(2)}}^T A^{(2)} \Lambda' \quad (16)$$

This is exactly the generalized eigenvalue formulation of Canonical Correlation Analysis, a well-known method of correlating linear relationships between two multidimensional variables[Hardoon *et al.*, 2004]. Therefore, the hash functions for this special case are given by the CCA embedding of the training data. We note that [Udupa and Khapra, 2010] also use a CCA-based approach for the Multilingual People Search problem. However, instead of employing hash functions resulting from the CCA embedding, they use a geometric search technique.

4 Experimental Study

We now describe our experiments for evaluating the effectiveness of the hashing-based cross-view similarity search technique developed in Section 2. We report results for two problems: Japanese Language People Search and Multilingual People Search. To compare our results with published results, we follow the experimental regimen described in [Udupa and Khapra, 2010; Udupa and Kumar, 2010].

4.1 Experimental Setup

Both Japanese People Search and Multilingual People Search involve two stages: an offline indexing stage and an online querying stage. In the indexing stage, given a name directory, we break each name into its constituent tokens and form the set of distinct name tokens in the directory. These tokens are then used to create an inverted index, mapping tokens to the names they are part of. We then produce the codeword for each of these tokens by applying the hash functions of Section 2.3.

The index generated in the first stage is used in the querying stage. Given a query name, we break it into constituent tokens and generate the codewords for each of the tokens by applying appropriate hash functions. Using the codeword of each of the tokens, we retrieve all tokens in the index that are within a particular Hamming radius[6] of this codeword. We score the retrieved tokens and retain only the top 250 tokens based on their similarity scores. The score is computed using the Euclidean distance in the low dimensional embedding (Equation 7). The tokens that remain are approximate matches of the corresponding query token. Using the inverted index, we then retrieve all names that have any of these tokens. Finally, we score the names using the graph matching algorithm of [Udupa and Khapra, 2010].

4.2 Japanese Language People Search

In the Japanese Language People Search problem, the database consists of name records of people in any of the three scripts used in Japanese text as well as the Latin script and users query the database for finding people relevant to their information need. The name queries can be in any of the four scripts (Hiragana, Katakana, Kanji or Latin). Misspellings and Kanji characters make the problem very challenging as each Kanji character has multiple pronunciations. Further, phonetic variants are also observed in queries which along with misspellings necessitate similarity search.

As there is an almost one-to-one mapping between Katakana and Hiragana, we treat Kana as one of the views and Kanji as the other view. As the number of Kanji characters is high, we romanize the Kanji characters using a simple Romanization scheme. Thus, all Kanji names in the directory are romanized before indexing and the Kanji query is also romanized before querying.

Training

For learning hash functions for the Japanese Language People Search problem, we used $20,000$ single token parallel Japanese names in Kanji and Kana. The tokens were transformed into feature vectors by using the same feature extraction scheme as used by [Udupa and Khapra, 2010]. Character bigrams were the features and the presence of a character bigram in a name token activates the corresponding feature in the feature vector for that token. Because of the Romanization step, the dimension of the Kanji feature vector was 563 which is manageably small compared to the number of Kanji bigrams ($> 10,000 \times 10,000$). The dimension of the Kana view was 3107. After the name tokens were transformed into feature vectors, we learned the hash functions for the two views using the apparatus described in Section 2.2. We used MATLAB to solve the generalized eigenvalue problem and selected the top 32 eigenvectors. This gave us a 32-bit hash function for each of the views.

Test Set

To test our approach, we need a name directory and a test set. In our experiments, we used a name directory consisting of $56,300$ Japanese names which were written in both Kana and Kanji. We selected a random subset of these names and asked native Japaneses speakers to generate common phonetic variants and misspellings of the names in the subset. The resulting test set consisted of 332 Kanji queries and 69 Kana queries.

Baseline

We implemented a baseline system using Double Metaphone, a well-known phonetic search algorithm for names [Philips, 2000]. As Double Metaphone can handle only strings in the Latin alphabet, we first romanized the Kana and Kanji names and then generated the Double Metaphone index. The baseline system is similar to the hash-based system except that the candidate tokens are retrieved using Double Metaphone codewords instead of hash codewords and scored using edit distance.

Results

For each of the name in the test set, we checked the position at which the correct result was retrieved by our system and the baseline. A perfect system would always retrieve the correct name at position 1 for every query in the test set. Therefore, the precision of the retrieval system at position 1 (P@1) is an indicator of how well the system is doing. The closer P@1 of a system is to 1, the closer it is to perfection. Therefore, a high P@1 score is desirable. Oftentimes, the precision of the retrieval system at position $n > 1$ (P@n) is also a good indicator of the system's performance as in some applications more than one results can be shown to the user. So we report P@3 along with P@1.

Table 1 compares P@1 of the hashing-based system with the Double Metaphone baseline. We note that the P@1 of the hashing-based system is 87.50% higher than that of the baseline system. The improvement in P@1 can be attributed to the ability of the learnt hash functions to retrieve all candidate tokens that are similar to the given query token. From Table 1, we observe that P@3 of the hashing-based system is 72.03% more than that of the baseline system. Further, P@3 of the baseline is still substantially smaller than P@1 of the hashing-based system.

[6] In our experiments, the Hamming radius was 4 and codewords were each 32 bits long.

Table 1: Comparative Performance in Japanese Language People Search

	P@1	P@3
HASH	0.765	0.855
DM	0.408	0.497

4.3 Multilingual People Search

In the Multilingual People Search problem, the database consists of name records of people in one or more languages and users query the database in any of these languages. Unlike Japanese Language People Search which was essentially a 2-view problem, in Multilingual People Search the number of views can be greater than 2. Therefore, we consider two scenarios. In the first scenario, $K = 2$ and this allows us to compare our approach with that of [Udupa and Khapra, 2010]. In the second scenario, $K > 3$ and more than two languages are involved.

Training

We learnt the hash functions using parallel names extracted from Wikipedia. In the $K = 2$ scenario, we use the same training data as used by [Udupa and Khapra, 2010] to do a fair comparison of our system with theirs. In the $K = 3$ scenario, we used a 3-way parallel subset of the training data used by [Udupa and Khapra, 2010]. This subset had $8,209$ 3-way parallel single token names. In the $K = 4$ scenario, we used $7,908$ 4-way parallel single token names.

Test Sets

We used the same test sets as used by [Udupa and Khapra, 2010]. The name directory was created using Wikipedia and had about $500,000$ names. In all the experiments, English was the language of the name directory and the queries were in Hebrew, Kannada, Tamil and Hindi. Table 2 gives the sizes of the training and test sets.

Baseline

The baseline against which we compared our system was the system described in [Udupa and Khapra, 2010]. They use Canonical Correlation Analysis to find a common feature space for the two views of the data and employ an approximate nearest neighbor algorithm for similarity search. We use Tie-Aware Mean Reciprocal Ratio (MRR) as the measure of retrieval performance as this is what was reported by [Udupa and Khapra, 2010] in their work.

Results

Table 2 summarizes the results for the $K = 2$ scenario. We see that the hashing-based system gives substantially better results than tree-based geometric search, with an MRR improvement of 9.1% on an average.

Table 2: Summary of $K = 2$ Experiments

	Training Set Size	Test Set Size	MRR HASH	MRR GEOM
ENG-HIN	15541	1027	**0.725**	0.686
ENG-RUS	11527	1124	**0.629**	0.563
ENG-HEB	16317	998	**0.794**	0.723

Table 3 summarizes the results for the $K > 2$ scenario. The consistently good MRR of the systems show that our approach scales well for the $K > 2$ scenario. Note that in the $K = 4$ experiments, the results are comparable for the different directions of search.

In all the scenarios, the hashing-based system took about 200 milliseconds on an average to retrieve the results for a query. Note that our system ran on a single thread on a server and multithreading can improve the speed significantly.

Table 3: Summary of $K > 2$ Experiments

Training Data	ENG-HIN	ENG-KAN	ENG-TAM
ENG-HIN	0.725	-	-
ENG-KAN	-	0.712	-
ENG-TAM	-	-	0.71
ENG-HIN-KAN	0.716	0.722	-
ENG-HIN-KAN-TAM	0.69	0.678	0.708

5 Related Work

Similarity Search in large databases of high-dimensional data objects is a key task in many applications [Gupta and Jain, 1997; Datta *et al.*, 2008; Manning *et al.*, 2008; Croft *et al.*, 2009]. Unfortunately, even the theoretically best data structures for indexing multi-dimensional data are, on an average, as bad as a brute force exhaustive search of the database for high-dimensional data objects such as texts and images [Arya *et al.*, 1998]. Therefore, approximate similarity search algorithms that give a good tradeoff between retrieval accuracy and the computational complexity of search are desirable.

A very important class of approximate similarity search algorithms in both Computational Geometry and Machine Learning is hashing where data objects are represented using binary codewords [Charikar, 2002; Shakhnarovich *et al.*, 2008; Salakhutdinov and Hinton, 2009]. Given a query data object, similar data objects are retrieved from the database simply by retrieving all data objects with codewords within a small Hamming distance of the codeword for the query. When the codewords are compact, similarity search using hashing is computationally efficient [Weiss *et al.*, 2008].

The problem of learning hash functions for single-view data objects has been well studied. 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.

Whereas similarity search is well studied for data objects with a single view, not much research has been done on the subject of cross-view similarity search of multi-view data objects. In some applications, it is still possible to use some of these techniques provided there exists an application-specific method of transforming one view of a data object to another view (for instance by machine translating a document in Spanish to Russian). However, a principled approach for *learning to hash* multiview data objects that does not involve an application-specific intermediate stage is very much desirable.

Multilingual name search is an important problem with applications in Web Search and Enterprise Search. [Udupa and Khapra, 2010] proposed a solution for the problem for the two view scenario. They sought to find a common feature space for names in two languages and used Canonical Correlation Analysis for learning one such subspace. They employed a tree-based approximate nearest neighbor algorithm for similarity search in the common feature space.

Spelling correction of names using hash functions was studied by [Udupa and Kumar, 2010]. They proposed two different methods for learning hash functions and 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.

6 Conclusions

We gave a principled solution to the problem of learning hash functions for multiview data objects. We formulated the learning problem as a NP hard minimization problem and transformed it into a tractable eigenvalue problem by means a novel relaxation. We showed that some special cases of the general problem are related to dimensionality reduction techniques such as Locality Sensitive Indexing and Canonical Correlation Analysis. Finally, we showed that the proposed hashing method gives very good retrieval performance compared to baselines in two Multilingual Information Access problems.

References

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

[Arya *et al.*, 1998] Sunil Arya, David M. Mount, Nathan S. Netanyahu, Ruth Silverman, and Angela Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[Cai *et al.*, 2007] Deng Cai, Xiaofei He, Wei Vivian Zhang, and Jiawei Han. Regularized locality preserving indexing via spectral regression. In *CIKM*, pages 741–750, 2007.

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

[Croft *et al.*, 2009] Bruce Croft, Donald Metzler, and Trevor Strohman. *Search Engines: Information Retrieval in Practice*. Addison-Wesley Publishing Company, 2009.

[Datta *et al.*, 2008] Ritendra Datta, Dhiraj Joshi, Jia Li, and James Z. Wang. Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40(2):1–60, 2008.

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

[Gupta and Jain, 1997] Amarnath Gupta and Ramesh Jain. Visual information retrieval. *Commun. ACM*, 40(5):70–79, 1997.

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

[Larsen *et al.*, 2003] J. Larsen, L. K. Hansen, T. Kolenda, and F. Å. Nielsen. Independent component analysis in multimedia modeling. In *Fourth International Symposium on Independent Component Analysis and Blind Source Separation*, pages 687–696, 2003.

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

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

[Platt *et al.*, 2010] John Platt, Kristina Toutanova, and Wen-Tau Yih. Translingual document representations from discriminative projections. In *EMNLP*, 2010.

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

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

[Suzuki *et al.*, 2009] Hisami Suzuki, Xiao Li, and Jianfeng Gao. Discovery of term variation in japanese web search queries. In *ACL*, 2009.

[Udupa and Khapra, 2010] Raghavendra Udupa and Mitesh Khapra. Improving the multilingual user experience of wikipedia using cross-language name search. In *NAACL-HLT*, 2010.

[Udupa and Kumar, 2010] Raghavendra Udupa and Shaishav Kumar. Hashing-based approaches to spelling correction of personal names. In *EMNLP*, 2010.

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