Scalable Similarity Search with Optimized Kernel Hashing

Junfeng He
Columbia University
New York, NY, 10027
jh2700@columbia.edu

Wei Liu
Columbia University
New York, NY, 10027
wliu@ee.columbia.edu

Shih-Fu Chang
Columbia University
New York, NY, 10027
sfchang@ee.columbia.edu

ABSTRACT

Scalable similarity search is the core of many large scale learning or data mining applications. Recently, many research results demonstrate that one promising approach is creating compact and efficient hash codes that preserve data similarity. By efficient, we refer to the low correlation (and thus low redundancy) among generated codes. However, most existing hash methods are designed only for vector data. In this paper, we develop a new hashing algorithm to create efficient codes for large scale data of general formats with any kernel function, including kernels on vectors, graphs, sequences, sets and so on. Starting with the idea analogous to spectral hashing, novel formulations and solutions are proposed such that a kernel based hash function can be explicitly represented and optimized, and directly applied to compute compact hash codes for new samples of general formats. Moreover, we incorporate efficient techniques, such as Nyström approximation, to further reduce time and space complexity for indexing and search, making our algorithm scalable to huge data sets. Another important advantage of our method is the ability to handle diverse types of similarities according to actual task requirements, including both feature similarities and semantic similarities like label consistency. We evaluate our method using both vector and non-vector data sets at a large scale up to 1 million samples. Our comprehensive results show the proposed method outperforms several state-of-the-art approaches for all the tasks, with a significant gain for most tasks.

Categories and Subject Descriptors

H.3 [**Information Storage and Retrieval**]: Information Search and Retrieval; H.4 [**Information Systems Applications**]: Miscellaneous

General Terms

Algorithms, measurement, performance

Keywords

hashing, search, indexing, nearest neighbor, scalable, kernel method, structure data

1. INTRODUCTION AND RELATED WORK

Internet nowadays makes it very easy to download a huge amount of data of diverse modalities like texts, images, videos, or others. To make use of such data sets in machine learning and data mining, a critical step is similarity search, i.e., finding nearest neighbors of a given query. The straightforward solution that exhaustively computes similarity between every data point and the query obviously does not scale up due to the prohibitive cost associated with computation and storage. To handle the scalability issue, approximate similarity search techniques exploring tradeoffs between accuracy and speed have been actively explored in recent literatures [1, 2, 3, 6, 16, 7, 12, 11, 9], where we trade certain accuracy for faster speed. In some early works, spatial partitions of the feature space via various tree structures[1, 2] have been extensively studied. Despite good results for low dimensional data, the performance of such tree-based approaches is known to degrade significantly when the data dimension is high, sometimes even worse than linear scan. Recently, many hash coding based algorithms have been proposed to handle similarity search of high dimensional data. These methods follow the idea that similar data points are expected to be mapped to binary codes within a small Hamming distance. When such hash codes are used as adequate indexing keys, often a sublinear time complexity becomes feasible for large scale similarity search.

One well known example is locality-sensitive hashing (LSH) [3]. In LSH, random vectors are utilized to generate codes, such that two points in database within a small distance are shown to have a higher probability of collision, i.e., having the same hash code. LSH algorithms with L_p norms [6], inner products [5], and learned metrics [8] have been proposed in recent literature. Another well known method is spectral hashing[12], which rather than using randomized projections, generates much more compact codes by thresholding some nonlinear functions on the projections along the principal component directions. It has been shown to achieve much better performance than LSH in some applications[12].

Despite of the aforementioned progress in similarity search, most existing hashing methods have one important limitation: they assume data is represented in a vector format and can be embedded as points in a vector space. Such assumption unfortunately are not compatible with many real

Table 1: Comparison with several hashing algorithms

method	work with non-vectors?	code efficiency	fast indexing/search speed	preserve various similarities ?
LSH	no	low	yes	only feature similarity
Kernelized LSH	yes	low	yes	only kernel similarity on features
Semantic Hashing	no	medium	yes	yes
Spectral Hashing	no	high	yes	only feature similarity $\exp(-\|\|X_i - X_j\|\|^2/\sigma^2)$
Optimized Kernel Hashing	yes	high	yes	yes

data types in the forms of graphs, trees, sequences, sets, or other formats existent in applications involving multimedia, biology, or Web. For such general data types, usually certain complex kernels are defined to define and compute the data similarities. For instance, random walk kernel and subtree kernel [26] are proposed for graph data; pyramid matching and spatial pyramid matching kernels [24] are proposed for data with the format of a set of vectors; Earth mover's distance kernel is proposed for sequence data. How to apply hashing algorithms to those non-vector data with complex kernels becomes an important problem. Moreover, even if the data are stored in the vector format, many machine learning solutions benefit from the use of domain-specific kernel functions, for which the underlying data embedding to the high-dimensional space is not known explicitly, namely only the pair-wise kernel function is computable. Kernelized locality sensitive hashing (KLSH) [13] is one of the very few methods that have been developed for similarity search based on kernels that are applicable to both vector and non-vector data. However, as a direct extension of LSH, KLSH may suffer from the same problem of producing inefficient codes, and hence will not perform well when the number of bits is small.

In this paper, we aim at developing a hashing algorithm that can

1. work on general types of data with any kernel function.
2. generate efficient and compact codes.
3. achieve fast indexing and search speed.
4. preserve diverse types of similarities including both feature similarity and semantic similarity like label consistency.

These properties are motivated by practical requirements in large-scale problems. Properties 2 and 3 are critical for making the hashing algorithm scalable to huge data sets, in terms of space and time complexity. Property 1 is desired in order to support general types of data from different areas. Finally, property 4 is crucial since usually no single similarity measure is sufficient for achieving robust performance in tasks like classification, retrieval, and so on.

Comparison of several state-of-the-art hashing algorithms based on the above properties is shown in Table 1. None of them have properties 1 and 2 at the same time, while our objective is to design a novel hashing solution that satisfy all 4 properties.

Our algorithm follows the idea of generating data-dependent optimal hash codes similar to that used in spectral hashing. However, without an explicit hash function, the original optimal hash code formulation in spectral hashing suffers from the inability in handling new data points. To derive approximate solutions, in spectral hashing some strict restrictions were used, which caused several limitations. To overcome such problems, in our solution a kernel hash function is explicitly represented and learned via optimization, which can be directly applied to novel samples even in nonvector data format. In addition, several speed-up techniques, such as those based on landmark points or Nystrom approximation, are incorporated to reduce the time complexity involved in the indexing and search stages. Finally, our method does not make any assumption about the similarity terms. Therefore, diverse types of similarities such as feature similarity, label consistence, or other association relations can be easily handled. Hence, our method can conveniently support unsupervised, supervised, or semi-supervised hashing.

We have evaluated our algorithm on several databases including vector data, graphs, and sets, with a data size ranging from several thousand to one million, for different tasks such as image retrieval, classification, and near duplicate detection. Compared to other stat-of-the-art approaches, our proposed method achieves significant performance gains for most of the data sets/tasks. In addition, our algorithm only needs to solve a small scale eigenvector problem, and hence is very easy to implement and reproduce.

2. OPTIMIZED KERNEL HASHING

2.1 Background: Spectral Hashing

Suppose we have N samples $\{X_j, j = 1, ...N\}$, W_{ij} is the similarity between sample X_i and sample X_j. As shown in spectral hashing, efficient hash codes can be obtained by the following optimization problem:

$$\begin{aligned} \min_{Y} \quad & \sum_{i,j=1}^{N} W_{ij}||Y_i - Y_j||^2 \\ s.t. \quad & \sum_{i=1}^{N} Y_i = 0 \\ & \frac{1}{N}\sum_{i=1}^{N} Y_i Y_i^{\top} = I \\ & Y_i \in \{-1, 1\}^M \end{aligned} \tag{1}$$

Note here Y is a $M \times N$ bit matrix, and Y_i is ith column of Y, which is hash bits of sample X_i.

Here, $\sum_{i,j=1}^{N} W_{ij}||Y_i - Y_j||^2$ tries to preserve feature similarity between original data points. In other words, on average, samples with high similarity, i.e., larger W_{ij}, should have

Table 2: Analysis of obtaining optimized kernel hash functions and indexing training samples

	Description	Time complexity
step 1	Compute $K_{P\times N}$ and $K_{P\times P}$ as defined in (6) and (8);	$O(PNT_K)$
step 2	Compute C from (4) for sparse W or (25) for non sparse W;	$O(PN_S)$ or $O(P^2N + PLN)$;
step 3	Compute G from (5);	$O(P^2N)$
step 4	Apply SVD on G as in (15);	$O(P^3)$
step 5	Compute the matrix $\tilde{C}$ in (18);	$O(P^3)$
step 6	Get $\tilde{A}$ as M eigen vectors of $\tilde{C}$;	$O(P^3)$
step 7	Compute A from (16);	$O(P^3)$
step 8	Compute hash codes for N trraining samples with (9);	$O(NMP)$
In total	$L = O(P), N_S = O(PN)$	$O(PNT_K + P^2N + NMP)$

Table 3: Analysis of computing hash codes for a novel sample

	Description	Time complexity
step 1	Compute k_x from (20);	$O(PT_K)$
step 2	Compute hash codes from (19);	$O(PM)$
In total		$O(PT_K + PM)$

similar hash codes, i.e., smaller $||Y_i - Y_j||^2$. The constraint

$$\sum_{i=1}^{N} Y_i = 0$$

is to make sure every single bit component of the M hash bits should be balanced, i.e., 50% to be 1 and 50% to be -1, while the constraint

$$\frac{1}{N}\sum_{i=1}^{N} Y_i {Y_i}^\top = I$$

is to ensure low correlation among different bits. Both these two constraints are helpful to create compact hash bits[12].

Unfortunately, as pointed out in [12], the above problem is equivalent to balanced graph partitioning problem, and hence is NP-hard. In spectral hashing, first, the constraint $Y_i \in \{-1, 1\}^M$ is ignored to relax the problem, and the relaxed version turns out to be eigenvectors of graph Laplacian. However even after relaxation, the above solution does not produce hash functions that can be used to handle novel input samples. Therefore, several additional assumptions are made in [12] in order to obtain an approximate solution that can handle novel samples. First of all, the data are assumed to be uniformly distributed vectors. Moreover, the similarity matrix W has to be fixed as

$$W_{ij} = \exp(-||X_i - X_j||^2/\sigma^2)$$

With these strict assumptions, spectral hashing obtains an approximate solution by thresholding some nonlinear functions on the projections along the principal component directions [12].

2.2 Optimized Kernel Hashing

2.2.1 Formulation

In order to obtain efficient hash codes, we start with an idea and formulation similar as shown above for spectral hashing. However, unlike spectral hashing where no explicit hash functions are included, we explicitly include hash functions based on kernels, so that the learned hash functions can be directly applied to novel input samples of generic types. Specifically, we adopt the following formulation

$$\begin{aligned} \min_{A,b}\ & \frac{1}{2}\sum_{i,j=1}^{N} W_{ij}||Y_i - Y_j||^2 + \lambda \sum_{m=1}^{M} ||V_m||^2 \\ s.t.\ & \sum_{i=1}^{N} Y_i = 0 \\ & \frac{1}{N}\sum_{i=1}^{N} Y_i {Y_i}^\top = I \\ & Y_i \in \{-1,1\}^M \\ & Y_{mi} = h_m(X_i) = sign(V_m^\top \varphi(X_i) - b_m) \\ & V_m = \sum_{p=1}^{P} A_{pm}\varphi(Z_p),\ i = 1,\cdots,N,\ m = 1,\cdots,M \end{aligned} \tag{2}$$

Here Y_{mi} is the element in ith column and mth row in bit matrix Y, and hence is the mth bit for Y_i. There are M hash functions $\{h_m, m = 1, ...M\}$ in total, each of which is for one hash bit. Each hash function $h_m(X_i) = sign(V_m^\top \varphi(X_i) - b_m)$ is represented in the kernel form, as in most kernel learning methods [20], where V_m is the hyperplane vector in the kernel space, φ is the function for embedding samples to the kernel space and usually is not computable,and b_m is the threshold scalar. Since it is infeasible to define the hyperplane vector V_m directly in the kernel space, we use an approach similar to that in Kernelized LSH [13] to represent V_m as a linear combination of landmarks in the kernel space with combination weights denoted as A_{pm}. $\{Z_p, p = 1, ..., P\}$ are landmark samples, which for example can be a subset randomly chosen from the original N samples. Moreover, if the data are in vector form, the landmarks can also be some "basis" vectors generated by projections like PCA, or some cluster centers. Note that the weight matrix A is a $P \times M$ matrix. b is $M \times 1$ vector, where b_m is mth element in b. The term $\sum_{m=1}^{M} ||V_m||^2$ is utilized to a regularized term to control the smoothness of the kernel function.

In the following, we will derive the analytical solutions of the above optimization problem and analyze the complexity of the method. Specifically, we will show the optimal kernel hash functions can be found elegantly by solving an eigenvector problem.

2.2.2 *Derivation*

Proposition 1: With the same relaxation as in spectral hashing by ignoring the constraint of $Y_i \in \{-1,1\}^M$, the above optimization problem is equivalent to the following:

$$\begin{aligned} \min_A \quad & tr(A^\top \frac{(C+C^T)}{2} A) \\ s.t. \quad & A^\top G A = I \end{aligned} \tag{3}$$

with

$$b = A^\top \bar{k}.$$

where

$$C = K_{P\times N}(D-W)K_{P\times N}^\top + \lambda K_{P\times P} \tag{4}$$

and

$$G = \frac{1}{N}\sum_{i=1}^{N} K_{P\times N}K_{P\times N}^\top - \bar{k}\bar{k}^\top \tag{5}$$

Here $K_{P\times N}$ is the kernel matrix between P lamdmarks and N samples. More specifically the element of ith row and jth column for $K_{P\times N}$ is defined as

$$(K_{P\times N})_{p,i} = K(Z_p, X_i),\ p = 1,\cdots,P,\ i = 1,\cdots,N. \tag{6}$$

K_i is the ith column of $K_{P\times N}$, and

$$\bar{k} = (\sum_{i=1}^{N} K_i)/N. \tag{7}$$

$K_{P\times P}$ is the kernel matrix among P landmarks. More specifically the element of ith row and jth column for $K_{P\times P}$ is defined as

$$(K_{P\times P})_{i,j} = K(Z_i, Z_j),\ i = 1,\cdots,P,\ j = 1,\cdots,P \tag{8}$$

and D is a diagonal matrix with $D_{ii} = (\sum_{j=1}^{N} W_{ij} + \sum_{j=1}^{N} W_{ji})/2$, $(i = 1,\cdots,N)$.

Proof of Proposition 1:

The kernel hashing function $Y_{mi} = h_m(X_i) = sign(V_m^\top \varphi(X_i) - b_m)$ can be reformulated as

$$\begin{aligned} Y_{mi} = h_m(X_i) &= sign(V_m^\top \varphi(X_i) - b_m) \\ &= sign(A_m^\top K_i - b_m) \end{aligned} \tag{9}$$

where A_m is mth column of A or equivalently,

$$Y_i = sign(A^\top K_i - b) \tag{10}$$

and $b = [b_1, ..., b_M]^T$.

With the same relaxation as in spectral hashing by ignoring the constraint of $Y_i \in \{-1,1\}^M$, we will have

$$Y_i = A^\top K_i - b$$

Hence, from the constraint of $\sum_{i=1}^{N} Y_i = 0$, we can get

$$\sum_{i=1}^{N} (A^\top K_i - b) = 0 \Rightarrow b = A^\top \bar{k} \tag{11}$$

Moreover, since

$$\begin{aligned} \sum_{i=1}^{N} Y_i Y_i^\top &= \sum_{i=1}^{N} (A^\top K_i - b)(A^\top K_i - b)^\top \\ &= A^\top (K_{P\times N}K_{P\times N}^\top - \sum_{i=1}^{N} K_i \bar{k}^\top \\ &\quad - \sum_{i=1}^{N} \bar{k} K_i^\top + \sum_{i=1}^{N} \bar{k}\bar{k}^\top)A \\ &= A^\top (K_{P\times N}K_{P\times N}^\top - N\bar{k}\bar{k}^\top)A \end{aligned} \tag{12}$$

from the constraint of $\frac{1}{N}\sum_{i=1}^{N} Y_i Y_i^\top = I$, we can get

$$A^\top \frac{1}{N}(K_{P\times N}K_{P\times N}^\top - N\bar{k}\bar{k}^\top)A = I$$

And because

$$\begin{aligned} &\frac{1}{2}\sum_{i,j=1}^{N} W_{ij}||Y_i - Y_j||^2 + \lambda \sum_{m=1}^{M} ||V_m||^2 \\ &= tr\left(A^\top K_{P\times N}(D-W)K_{P\times N}^\top A\right) + \lambda tr\left(A^\top K_{P\times P} A\right) \\ &= tr\left(A^\top (K_{P\times N}(D-W)K_{P\times N}^\top + \lambda K_{P\times P})A\right) \end{aligned} \tag{13}$$

So the optimization problem in (2) becomes:

$$\begin{aligned} \min_A \quad & tr(A^\top C A) \\ s.t. \quad & A^\top G A = I \end{aligned} \tag{14}$$

where

$$C = K_{P\times N}(D-W)K_{P\times N}^\top + \lambda K_{P\times P}$$

$$G = \frac{1}{N}\sum_{i=1}^{N} K_{P\times N}K_{P\times N}^\top - \bar{k}\bar{k}^\top$$

Moreover, note that

$$tr(A^\top C A) = tr((A^\top C A)^T) = tr(A^\top C^T A)$$

so

$$tr(A^\top C A) = tr(A^\top \frac{(C+C^T)}{2} A)$$

which completes the proof of Proposition 1.

Note here C and G are both $P\times P$ matrix.

2.2.3 *Implementation*

The above optimization problem in (3) can be further rewritten into an eigen vector problem for simpler implementation.

More specifically, suppose the SVD decomposition of G is

$$G = T_0 \Lambda_0 T_0^\top \tag{15}$$

and denote $\tilde{A}$ as

$$A = T\Lambda^{-\frac{1}{2}}\tilde{A} \tag{16}$$

where Λ is a diagonal matrix consisting of M largest elements of Λ_0, while T is the corresponding columns of T_0.

The problem in (3) equals to

$$\min_{\tilde{A}} \quad tr\left(\tilde{A}^\top \Lambda^{-\frac{1}{2}} T^\top \frac{(C+C^T)}{2} T \Lambda^{-\frac{1}{2}} \tilde{A}\right)$$
$$s.t. \quad \tilde{A}^\top \tilde{A} = I \qquad (17)$$

The solution $\tilde{A}$ is a $M \times M$ matrix, which is the M eigen vectors for matrix

$$\tilde{C} = \Lambda^{-\frac{1}{2}} T^\top \frac{(C+C^T)}{2} T \Lambda^{-\frac{1}{2}}. \qquad (18)$$

Given $\tilde{A}$, A can be obtained from equation (16). For a novel sample x, its mth bit code y_m can be computed as

$$y_m = h_m(x) = sign(A_m^\top k_x - b_m) \qquad (19)$$

where

$$k_x = [K(x, Z_1), ..., K(x, Z_P)]^\top \qquad (20)$$

namely, the kernel values between x and the landmark points. Equally, $y = sign(A^\top k_x - b)$.

A complete workflow of our algorithm can be found in Table 2 and 3. As shown in the above, kernel based hash functions $\{h_m, m = 1, ..., M\}$ can be optimized by solving an eigen vector problem on a matrix with a size around $M \times M$. (Recall (19), (18) and (16)). After $\{h_m, m = 1, ..., M\}$ are learned via optimization, they can directly hash new samples of any data format using properly defined kernel function, as shown in (19) and (20).

2.2.4 Discussion and extension

W in our algorithm does not need to be fixed as $W_{ij} = \exp(-||X_i - X_j||^2/\sigma^2)$ like in spectral hashing. Furthermore, the common requirements for similarity matrix, like positive semi-definite, or no negative elements, are unnecessary here either. Finally, from equation (18) and (4), it is easy to see even nonsymmetric W is applicable. Actually, any real W with reasonable physical meaning of some kinds of similarity can be applied in our method. So, besides the usual feature similarities, other kinds of similarities, e.g., those based on class label consistency, can also be used. Hence, our method can conveniently support unsupervised, supervised, or semi-supervised hashing.

Besides the formulation in equation (2), other variation or extension is possible. For instance, suppose no similarity score W_{ij} is directly provided, and only some ranking tuples are available, i.e., we know sample j is ranked as more similar to i than k is. In this case ranking information instead of similarity scores are supposed to be preserved via "relative comparison", i,e., if a pair of sample (i, j) is ranked as more similar than pair (i, k), then the distance of hash codes between j and i are supposed to be smaller than that between k and i.

The cost function in equation (2) would be changed to

$$\sum_{(i,j)>(i,k)} (||Y_i - Y_j||^2 - ||Y_i - Y_k||^2) + \lambda \sum_{m=1}^{M} ||V_m||^2 \qquad (21)$$

where $(i, j) > (i, k)$ means j is ranked as more similar to i than k is, and $\sum_{(i,j)>(i,k)}$ means to sum over all ranked tuples. In this case, the definition of C in equation (4) would become:

$$C = \lambda K_{P\times P} + \sum_{(i,j)>(i,k)} X_j X_j^T + X_k X_k^T - 2(X_j + X_k) X_i^T \qquad (22)$$

Note here sample i does not necessarily come from training sets. For example, in the scenario of relevance feedback, sample i can be novel queries, and sample j and k could be some retrieved samples for the query.

3. SCALABILITY

In the above method, the bottleneck of computation is W and $K_{P\times N}(D - W)K_{P\times N}^\top$. Since W is $N \times N$ matrix, when we have large scale data set, what may consist of millions of samples, W and $K_{P\times N}(D - W)K_{P\times N}^\top$ would be every expensive to compute, with a time complexity of N^2 and PN^2 respectively.

3.1 Sparse Representation of W

One way to overcome the computation complexity is to use a sparse W.

Sometimes a sparse W can be obtained with supervised information. For example, in the task of near-duplicate text (or image) detection, for every sample in the training set, we only need to consider the similarity between one sample and its near-duplicates, and hence W is very sparse. Another example is multi-class classification task, such as news topic categorization or image object categorization, only samples in the same class would be considered to compute similarity.

More generally, one can always obtain sparse W by sampling a small subset of training samples to compute similarity matrix.

If W is sparse, even when N is very large, we can still compute $K_{P\times N}(D - W)K_{P\times N}^\top$ directly.

3.2 Low Rank Representation of W

Another approach to overcome the computation complexity is to apply a low rank representation/approximation for W to handle huge data set. First, assume W (or $\frac{W+W^T}{2}$ if W is not symmetric) can be computed or approximated as

$$W = RQR^\top$$

where R is a $N \times L$ matrix and Q is $L \times L$ matrix. Usually $L << N$. In our experiments, L is set to be $O(P)$.

This low rank approximation is often possible. For instance, if the data are vectors, W can be defined as inner products $W = XX^\top$, and hence $R = X$ and $Q = I$. More generally, when other similarities such as some kinds of feature similarities or kernel similarities is used, often Nyström algorithm [21] can be applied to get a low rank approximation for W, such that $R = W_{N\times L}$ and $Q = W_{L\times L}^{-1}$. (if $W_{L\times L}$ is not invertible, use the pseudo inverse). Here $W_{N\times L}$ is the similarity matrix between N samples and L selected samples in Nyström algorithm, and $W_{L\times L}$ is the similarity matrix among L selected samples.

Using the aforementioned approximation $W = RQR^\top$,

$$K_{P\times N} W K_{P\times N}^\top = (K_{P\times N} R) Q (K_{P\times N} R)^\top \qquad (23)$$

$$K_{P\times N} D K_{P\times N}^\top = \sum_{j=1}^{N} d_j K_j K_j^\top \qquad (24)$$

where $d = W1 = RQ(R^\top 1)$. Therefore, C can be computed as

$$C = (\sum_{j=1}^{N} d_j K_j K_j^\top) - (K_{P\times N}R)Q(K_{P\times N}R)^\top + \lambda K_{P\times P}. \tag{25}$$

which involves small matrix sizes only.

3.3 Time Complexity Analysis

The detailed description and time complexity analysis for our algorithm are provided in Table 2 and 3. To maintain a sublinear time online searches, P are usually chosen as a number much smaller than N. Note that T_K is the time complexity to compute the kernel between two samples and N_S is the number of non-zero elements in sparse W.

If W is sparse, we can compute C from (4), and the time complexity is $O(PN_S)$; otherwise if W is not sparse, we can compute C from (25) with a complexity of $O(P^2N + PLN)$. If W is sparse, $N_S << N^2$, and usually we can assume $N_S = O(NP)$. Moreover, L are often chosen as a number close to P. So in both cases, the complexity to compute C is $O(P^2N)$, which is much faster than $O(PN^2)$, the original complexity if without any speedup. Note that in step 2 of Table 2, we also eed to compute W. However, the time complexity of obtaining a sparse W or computing R and Q is negligible compared to the complexity of step 1 and 2 in Table 2.

As shown in Table 2 and 3, the complexity to obtain the optimized kernel hash functions and compute hash codes for training samples is $O(PNT_K + P^2N + NMP)$ and the complexity to compute hash codes for one novel query sample is $O(PT_K + PM)$. Our complexity is actually the same as that of kernelized locality sensitive hashing. In practice, to index a data set consisting of around 1 million samples, spectral hashing, kernelized locality sensitive hashing, and our algorithm cost about the same amount of time, i.e., several hours.

The space complexity of our method is $O(NP)$, no matter we use a sparse W or a low rank approximation for W.

4. EXPERIMENTS

4.1 Discussion on experiment setup

As shown in equation (2), our algorithm needs to select a set of landmark samples. These landmark samples, for example, can be a subset randomly chosen from the original training data, some "basis" vectors generated by projections like PCA, or some cluster centers.

Our algorithm only involves one parameter: λ as in (2). Though in some preliminary experiments, we found the performance can indeed be improved by carefully selecting the parameter λ. However tuning the parameter needs extra time. To reduce the learning time especially on the large scale data set, in the following experiments, we set $\lambda = 0$. As shown in the experiment results, such simplified method performs well, for example, better than other state-of-the-art methods.

We compare our algorithm with several state-of-the-art methods including locality sensitive hashing (LSH), spectral hashing (SH), and kernelized locality sensitive hashing (KLSH). All algorithms are compared using the same number of hash bits. For the latter two, We used the codes provided by the original authors, which can be downloaded from Internet. For LSH, We use our own implementation according to [5], which is reported as one of the best variation for LSH. For a fair comparison, we always use the same number of landmark samples and the same kernel for our method and KLSH.

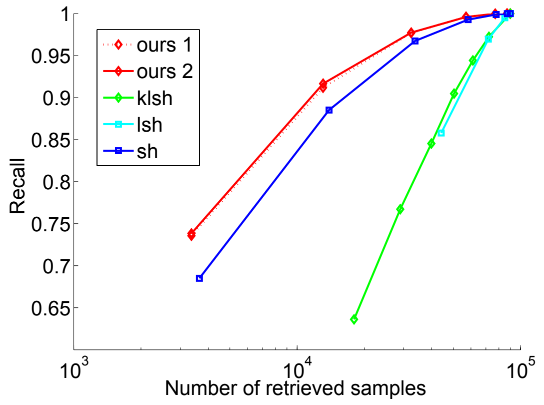
To evaluate the above approximate methods, we need to obtain groundtruth of true nearest neighbors for each query sample. One way to define groundtruth near neighbors is to choose the top samples that has highest feature similarities to the query. Another way is to use side information other than features. For example, in the task of near-duplicate detection, near-duplicate groundtruth are known and can be used as groundtruth for nearest neighbors.

4.2 Near-duplicate detection on Photo Tourism image patch data set - 100K samples

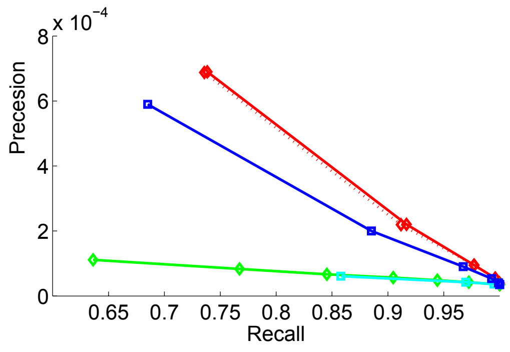
The first data set we use is Photo Tourism image patch set[25]. In our experiment, we use 100K patches, which are extracted from a collection of Notre Dame pictures. $10K$ patches are randomly chosen as queries, the rest $90K$ are used as training set to learn the optimized kernel hashing function. For each patch, 512 dimension gist features [18] are extracted. The task on this data set is to identify the neighbors, i.e., near-duplicate patches, in the training set for each query patch. The groundtruth neighbors are defined based on the $3D$ information. More specifically, for each patch in the data set, there is a label provided to describe its $3D$ position. Patches with the same label are defined as groundtruth neighbors, which are near duplicate image patches of the same place of Notre Dame, with variations in lighting, camera viewpoints, and etc.

In some papers, performance is measured in terms of recall rate for the results with a hamming distance from the query smaller than a threshold. However, this kind of evaluation may be biased sometimes. In an extreme case, if one hashing algorithm maps all data into the same code, then all training samples would have a 0 hamming distance to any query, and hence have a recall of 1 all the time. This hashing algorithm would be "misstated" as the best, but indeed it is one of the worst cases. In our experiments, we report recall rate, i.e., percentage of groundtruth neighbors found, together with the number of retrieved samples within a hamming distance from the query smaller than a threshold. This evaluation metric contains information related to represent both search quality (recall rate) and search time (number of retrieved examples). Moreover, the precision-recall curve is also reported . The experiment results of our algorithm compared to locality sensitive hashing, spectral hashing, and kernelized locality sensitive hashing are shown in Figure 1. As we can see, with a similar number of retrieved samples, our algorithm achieved significantly higher recall than all the other three methods. With the same recall rate, the precision of our method is often several times higher than that of SH, hundreds of times higher than LSH and KLSH.

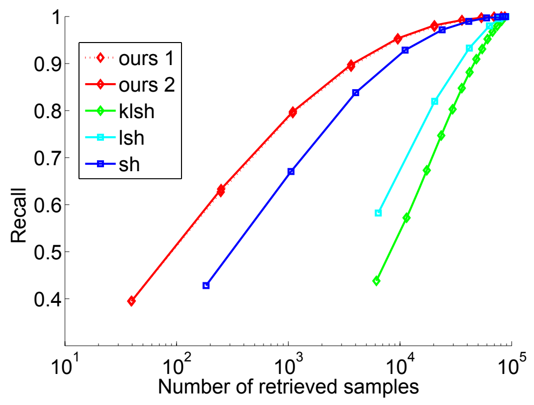
It is a little surprising that our method can create significantly more efficient codes than spectral hashing in such vector data, considering both solutions are designed with similar objectives, e.g., balanced efficient codes. One possible reason is that spectral hashing assumes a uniform distribution for the data points, which may not be true in this data set. Moreover, another limitation of spectral hashing is that its similarity matrix W has to be fixed as $W_{ij} =$



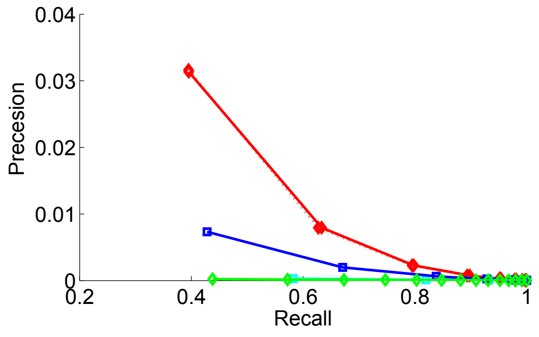
(a) 8 bits hash code



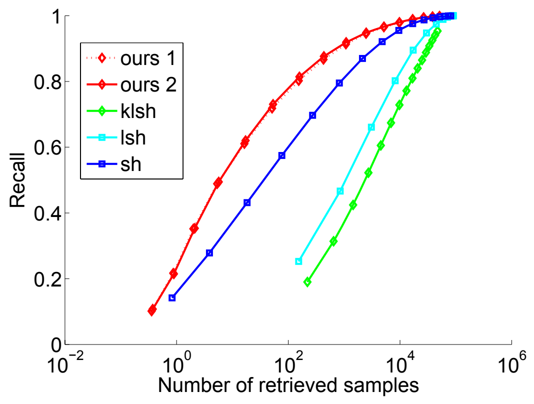
(d) Corresponding precision-recall curve for (a)



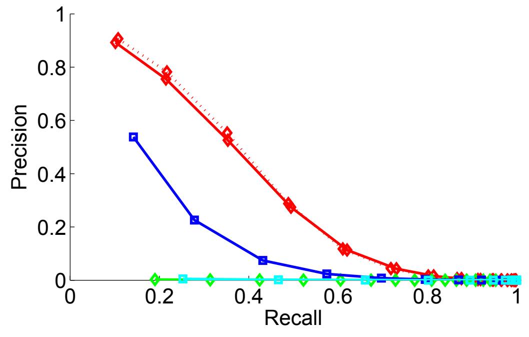
(b) 16 bits hash code



(e) Corresponding precision-recall curve for (b)



(c) 32 bits hash code



(f) Corresponding precision-recall curve for (c)

Figure 1: Near-duplicate search results on Photo Tourism data set for several hashing algorithms with 8 bits, 16 bits and 32 bits hash code. lsh: locality sensitive hashing; klsh: kernelized locality sensitive hashing; sh: spectral hashing; In (a), (b) and (c), the horizontal axis is the number of retrieved samples on average. The vertical axis is the recall rate, i.e. the portion of groundtruth neighbors covered by the retrieved samples. In the first implementation of our algorithm, denoted as "ours 1", 500 landmark samples are generated by PCA projections, while in the second implementation of our algorithm, denoted as "ours 2" and also KLSH, 500 landmark samples are randomly chosen from original training set. Linear kernel (inner product) is used in our algorithm and KLSH. For the convenience of readers, the corresponding precision-recall curves for (a), (b) and (c) are shown in (d), (e) and (f) respectively. For most cases, with the same recall, our precision is hundreds of times higher than LSH or KLSH, and several times higher than SH. Graphs are best viewed in color.

$\exp(-||X_i - X_j||^2/\sigma^2)$, which may not be suitable for the task here either. On the contrary, the similarity matrix W used in our algorithm here is defined as the label consistency directly, i.e., $W_{ij} = 1$, if the ith patch and jth patch in the training set have the same label, namely near-duplicate pairs; otherwise $W_{ij} = 0$. This demonstrates a unique strength of our algorithm, unlike most existing hashing algorithms that can only preserve some kinds of feature similarity, our algorithm can preserve various type similarities other than feature similarity, or more specifically any kinds of similarities represented by a real matrix W. This property is one of the keys to the superior performance achieved by our algorithm. This is confirmed by the results shown in Figure 2, in which label similarity and feature similarity are directly compared.

Sensitivity to landmark points

Since our algorithm needs some landmark points, which might be randomly chosen. So a reasonable concern is that how sensitive the performance are affected by the landmark points. Two strategies of choosing landmark samples are actually shown in Figure 1: random selection and deterministic generation via PCA. As we can see, these two strategies provide almost the same result, which demonstrates that our algorithm is quite robust to landmark samples. Actually, from our preliminary observation which is not reported here, changing the number of landmark points within a wide range, for example from 200 to 1000, only affects the performance slightly.

4.3 KNN Classification on biological data with graph kernel - 4K samples

NCI1[27] is a biological data set with 4K samples. Each sample in this data set is a compound represented via a graph, with a label to show whether or not the compound is active in an anti-cancer screen (http://pubchem.ncbi.nlm.nih.gov). Though the data set is not large, however, the kernel similarities between samples are very expensive to compute. For example, even some state-of-the-art graph kernels methods have to take several seconds or minutes to compute the kernel similarity between a single pair[26]. So approximate nearest neighbor search via hashing on this median-size data set is still an important problem. More details on the data and the graph kernel can be found in [26].

Due to the non-vector data type, LSH and spectral hashing can not be applied to this graph data. So we can only compare our algorithms with the KLSH method. The experiments are repeated 5 times. In each time, 90% of the data are chosen as the training set, the other 10% are used as test queries, and KLSH and our algorithm use the same number of randomly chosen landmark points. For each test query sample, we find its top k nearest neighbors based on Hamming distance to the query bits. The label of the query sample is predicted by the majority of labels from top k-nearest neighbors. In table 4, the average accuracy over 5 runs is shown. We can see that our method is clearly better than the KLSH method, especially when k is small ($10\% - 20\%$ performance gain).

4.4 Retrieval on Caltech101 image set with spatial pyramid matching kernel - 10K samples

Caltech101 is an image data set of about 10K samples [23]. In our experiment, local SIFT features [19] are extracted for each image, and Bag of visual words with geometry location are used. In other words, each image is represented by a set of visual words with geometry locations. Spatial pyramid matching (SPM) kernel [24] is used to compute the set similarity between two images. Since this data set consists of non-vector data, only KLSH and our algorithm are applicable. 90% of the data are selected as training samples while the other 10% are used as queries. The groundtruth neighbors for each query is defined as top 300 neighbors in the training set found via linear scan with spatial pyramid matching kernel similarity. KLSH and our algorithm use the same parameters like the number of landmark samples and so on. The similarity matrix for our method is defined as label consistency, i.e., $W_{ij} = 1$, if the ith sample and jth sample have the same object class label; otherwise $W_{ij} = 0$. The precision-recall curves for our algorithm and KLSH are shown in Figure 3, confirming the superiority of our method.

4.5 Retrieval on web images data set - 1M samples

We have downloaded around $1M$ web images from flickr web site [1]. For each image, 512 dimension gist features[18] are extracted. The groundtruth neighbors for each query here is defined as top 1% samples in the training set found via linear scan of inner product. In this case, a factorization for $W = XX^T$ can be used and the tricks described in section 3 can be applied to handle a huge data set of this scale (1 million). Specifically, $R = X$ and $Q = I$ in our experiments.

For our method and KLSH, RBF kernel with the same parameter is used, and moreover, the number of landmark points are set close to the number of feature dimensions, such that KLSH, SH and our algorithm would have almost the same amount indexing time, i.e., several hours by using a regular workstation. The results are shown in Figure 4. Our algorithm provides better results compared to other two methods.

We have also tried other kernels like linear kernel for KLSH and our method. With linear kernel, our method performs comparably or slightly better than KLSH and SH. This result is not shown due to space limit.

5. CONCLUSION

In this paper, we have proposed a novel and effective hashing algorithm that can create compact hash codes for general types of data with any kernel, and can be easily scaled to huge data set consisting of millions of samples.

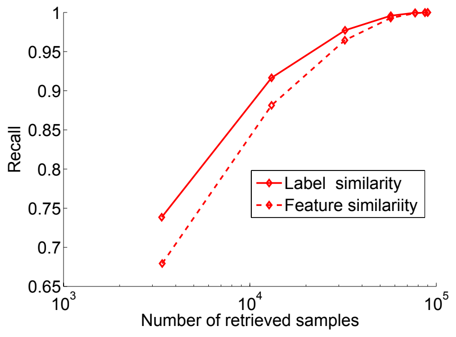
Future works include study of how other large matrix approximation methods can be incorporated with hash function learning and how they will affect the performance of the integrated approach. In addition, we will study how to select suitable kernels and how to fuse multiple kernels for specific tasks.

6. ACKNOWLEDGMENTS

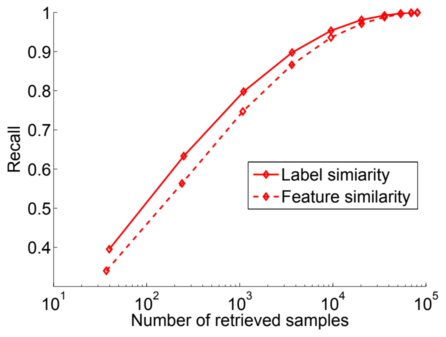
This work is supported in part by National Science Foundation through Grant No CNS-07-51078 and CNS-07-16203, and Office of Naval Research.

We also thank Professor Nenghai Yu and his group for their assistance in providing the web images.

[1]www.flickr.com

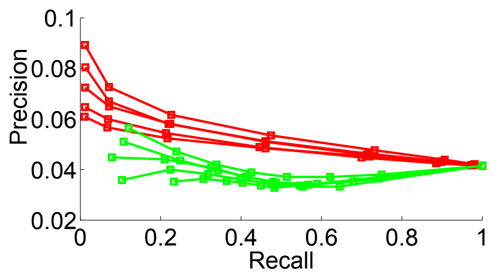


(a) 8 bits hash code

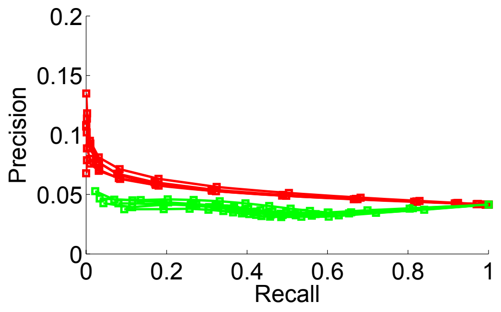


(b) 16 bits hash code

Figure 2: Near-duplicate search using our algorithm with feature similarity and label consistency similarity on Photo Tourism data set. Label consistency similarity helps improve the performance of our algorithm.

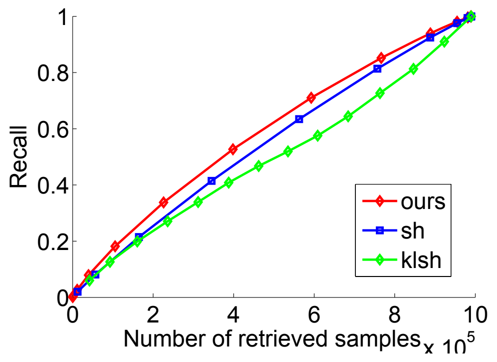


(a) 8 bits hash code

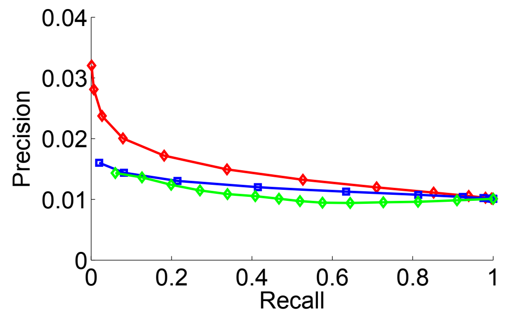


(b) 16 bits hash code

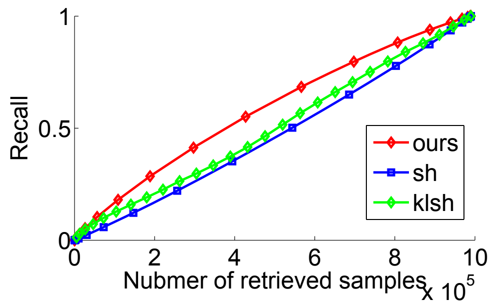
Figure 3: Retrieval results using our algorithm (red lines) and KLSH (gree lines) with spatial pyramid matching kernel on SIFT feauters for Caltech 101 data set. The experiments are repeated 5 times (one line for each). The horizontal axis is the recall rate while the vertical axis is the precision rate.



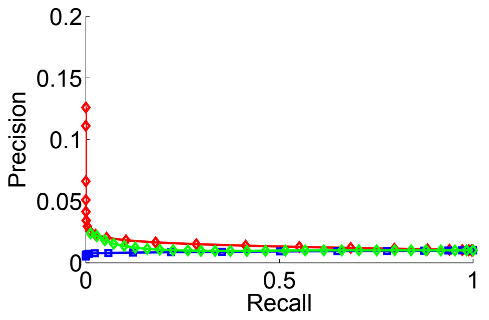
(a) 16 bits hash code



(c) Corresponding precision-recall curve for (a)



(b) 32 bits hash code



(d) Corresponding precision-recall curve for (b)

Figure 4: Search results on flickr data set with 16 and 32 bits hashing. In (a) and (b), the horizontal axis is the number of retrieved samples on average. The vertical axis is the recall rate, i.e. the portion of groundtruth neighbors covered by the retrieved samples. (c) and (d) are the corresponding precision-recall curve for (a) and (b). RBF kernel with the same parameter is used for our method and KLSH.

Table 4: Accuray of KNN classification with nearest neighbors obtained by our method and KLSH on NCI biological data set

indexing method	k=3	k=6	k=9	k=12	k=15	k=18	k=21	k=24	k=27	k=30
ours with 16 bits	0.6307	0.6355	0.6506	0.6633	0.6633	0.6628	0.6667	0.6613	0.6628	0.6667
klsh with 16 bits	0.5800	0.5294	0.5796	0.5800	0.6496	0.5698	0.6068	0.5820	0.6131	0.6146
ours with 32 bits	0.7221	0.7134	0.7139	0.7022	0.7158	0.7129	0.7207	0.7148	0.7144	0.7085
klsh with 32 bits	0.5917	0.5518	0.5990	0.6019	0.6350	0.6234	0.6219	0.6253	0.6282	0.6277

7. REFERENCES

[1] J. Freidman, J. Bentley, and A. Finkel. An Algorithm for Finding BestMatches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, page 209-226, 1977.

[2] J. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, page 175-179, 1991.

[3] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *In Proceedings of 30th Symposium on Theory of Computing (STOC)*, 1998.

[4] A. Gionis, P. Indyk, and R.Motwani. Similarity search in high dimensions via hashing. *In Proceedings of the 25th International Conference on Very Large Data Bases*, 1999.

[5] M. Charikar. Similarity search in high dimensions via hashing. *In Proceedings of the ACM Symposium on Theory of Computing*, 2002.

[6] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality sensitive hashing scheme based on p-Stable distributions. *In Proceedings of Symposium on Computational Geometry (SOCG)*, 2004.

[7] K. Grauman and T. Darrell. Pyramid match hashing: sub-Linear time indexing over partial correspondences. *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[8] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[9] R. Salakhutdinov and G. Hinton. Semantic hashing. *In Proceedings of ACM SIGIR Special Interest Group on Information Retrieval*, 2007.

[10] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. *In Proceedings of AI and Statistics*, 2007.

[11] A. Torralba, R. Fergus, and Y.Weiss. Small Codes and Large Image Databases for Recognition. *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[12] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *In Proceedings of Advances in Neural Information Processing Systems*, 2008.

[13] Brian Kulis and Kristen Grauman. Kernelized Locality-Sensitive Hashing for Scalable Image Search. *In Proceedings of 12th International Conference on Computer Vision*, 2009.

[14] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. *In Proceedings of Advances in Neural Information Processing Systems*, 2009.

[15] M. Raginsky and S. Lazebnik. Locality sensitive binary codes from shift-invariant kernels. *In Proceedings of Advances in Neural Information Processing Systems*, 2009.

[16] Kave Eshghi and Shyamsundar Rajaram. Locality sensitive hash functions based on concomitant rank order statistics. *In Proceedings of ACM SIGKDD conference on Knowledge Discovery and Data Mining*, 2008.

[17] Xiaofei He, and Partha Niyogi. Locality Preserving Projections. *In Proceedings of Advances in Neural Information Processing Systems*, 2003.

[18] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal on Computer Vision*, 2001.

[19] D. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal on Computer Vision*, 2004.

[20] Vladimir N Vapnik. *The Nature of Statistical Learning Theory*, 1995.

[21] Christopher Williams , Matthias Seeger. Using the nyström method to speed up kernel machines. *In Proceedings of Advances in Neural Information Processing Systems*, 2001.

[22] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. *In Proceedings of Advances in Neural Information Processing Systems*, 2001.

[23] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. *In Proceedings of CVPR Workshop of Generative Model Based Vision*, 2004.

[24] S. Lazebnik, C. Schmid and J. Ponce. Beyond bags of features, spatial pyramid matching for recognizing natural scene categories. *In Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 2006.

[25] N. Snavely, S. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3D. *In Proceedings of SIGGRAPH*, 2006.

[26] Nino Shervashidze, Karsten M. Borgwardt. Fast subtree kernels on graphs. *In Proceedings of Advances in Neural Information Processing Systems*, 2009.

[27] N. Wale and G. Karypis. Comparison of descriptor spaces for chemical compound retrieval and classification. *In Proceedings of ICDM* , 2006.