Pattern Recognition Letters 33 (2012) 271–277

Contents lists available at SciVerse ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Sparse spectral hashing

Jian Shao, Fei Wu *, Chuanfei Ouyang, Xiao Zhang

College of Computer Science, Zhejiang University, China

ARTICLE INFO

Article history:
Received 28 March 2011
Available online 9 November 2011
Communicated by J. Laaksonen

Keywords:
Semantic hashing
Sparse principal component analysis
Laplacian eigenmap
AdaBoost

ABSTRACT

A better similarity index structure for high-dimensional feature datapoints is very desirable for building scalable content-based search systems on feature-rich dataset. In this paper, we introduce sparse principal component analysis (Sparse PCA) and Boosting Similarity Sensitive Hashing (Boosting SSC) into traditional spectral hashing for both effective and data-aware binary coding for real data. We call this Sparse Spectral Hashing (SSH). SSH formulates the problem of binary coding as a thresholding a subset of eigenvectors of the Laplacian graph by constraining the number of nonzero features. The convex relaxation and eigenfunction learning are conducted in SSH to make the coding globally optimal and effective to datapoints outside the training data. The comparisons in terms of F1 score and AUC show that SSH outperforms other methods substantially over both image and text datasets.

1. Introduction

Fast finding the similar datapoints to a given query from a large scale database is critical to content-based information retrieval. Given a query, the naive solution to accurately find the examples that are most similar to the query is to search over all the datapoints in database and sort them according to their similarity to the query. However, this becomes prohibitively expensive when the scale of the database is very large. To speed up the process of finding relevant datapoints for a query, indexing techniques are necessarily conducted to organize the datasets. However, some studies pointed out that many of the index structures have an exponential dependence (in space or time or both) upon the number of dimensions, therefore even a simple brute-force, linear-scan approach may be more efficient than an index-based search in high-dimensional settings (Berchtold et al., 2000). Moreover, an excellent index structure should guarantee that the similarity of two datapoints in index space keeps consistency with their similarity in the original data space (Salakhutdinov and Hinton, 2007).

Recently, Locality Sensitive Hashing (LSH) and its variations have been proposed as indexing approaches for approximate nearest neighbor search (Datar et al., 2004; Lv et al., 2007). The basic idea in LSH is to use a family of locality preserving hash functions to hash similar datapoints in the high-dimensional space into the same bucket with high probability than non-similar datapoints. To conduct similarity search, LSH hashes a query datapoint into a bucket and take the datapoints in the bucket as the retrieved candidates. A final step is then performed to rank the candidate data by calculating their distance to the query, or just return several candidate datapoints randomly. As shown by Salakhutdinov and Hinton (2009), Weiss et al. (2009), LSH could be unstable and may lead to extremely bad result for a small number of bits due to its randomized approximate similarity search. Therefore, the number of hash bits required may be large in some cases in order to faithfully maintain the input distance in LSH.

Unlike those approaches which randomly project the input data into an embedding space such as LSH, some machine learning approaches were recently implemented to generate more compact and accurate binary codewords for data indexing, such as Restricted Boltzmann Machine (RBM) in semantic hashing (Salakhutdinov and Hinton, 2009) and Parameter Sensitive Hashing (PSH) in pose estimation (Shakhnarovich et al., 2003). These approaches attempt to elaborate appropriate hash functions to optimize an underlying hashing objective. For examples, RBM outputs a binary code for each data so that similar data will have similar binary codewords by training a deep graphical model. PSH utilizes a Boosting method on LSH to classify data and several weak learners decide the binary code together. However, RBM has an extremely complex model which means the parameters of RBM is very difficult to determine. Meanwhile, both of the mentioned machine learning approaches suffer from the inaccuracy of the datapoints outside the training set.

Spectral Hashing (Weiss et al., 2009) (SH) formalizes the problem of semantic hashing as a form of graph partitioning and designs an efficient eigenvector solution for graph partition in order to generate the best binary code for data indexing. SH finds a projection from Euclidean space to Hamming space and guarantees that the datapoints which are close in Euclidean space will be mapped to similar binary codewords. In order to avoid NP-hard problem, spectral relaxation is implemented in SH to obtain a

* Corresponding author.
E-mail addresses: jshao@cs.zju.edu.cn (J. Shao), wufei@zju.edu.cn (F. Wu), chfouyang@gmail.com (C. Ouyang), cherry.zhang.hz@gmail.com (X. Zhang).

0167-8655/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2011.10.018

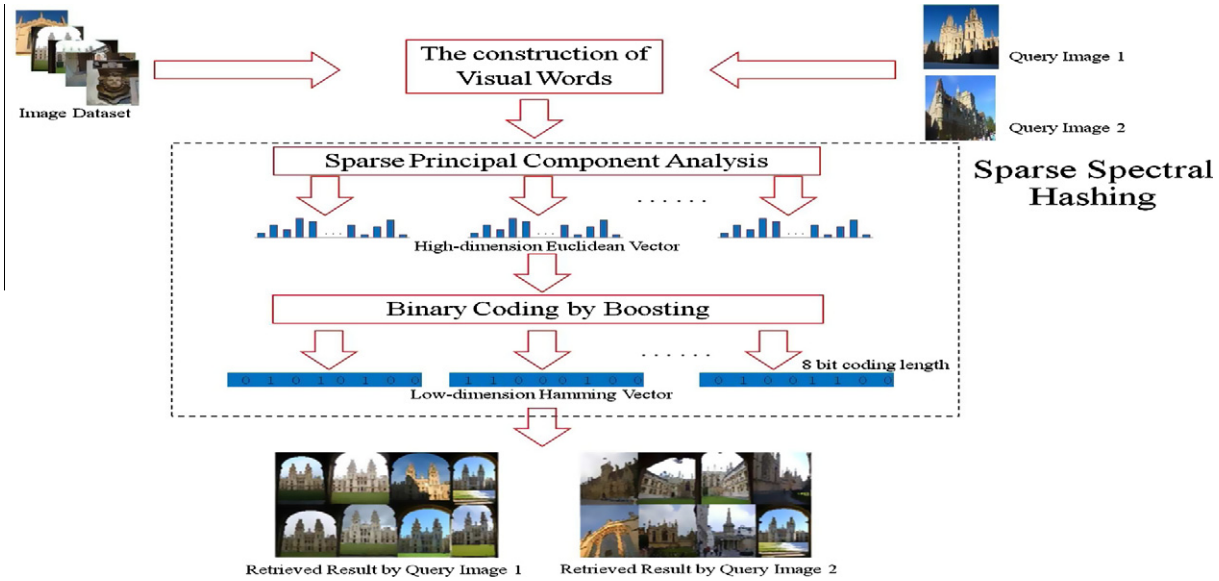

Fig. 1. Flowchart of sparse spectral hashing.

number of eigenvectors with minimal eigenvalue from Laplacian matrix after a Principal Component Analysis (PCA) step. However, SH suffers from the fact that the data-obvious binary coding is not suitable for many datasets because of the assumption of multi-dimensional uniform distribution. Moreover, It is very important to only select discriminative features for indexing (Fei et al., 2010). Spectral Hashing cannot actually remove the affect of over-complete features due to the implementation of PCA. There are many of approaches for feature selection, for examples, Bhattacharyya has proposed a second order programming formulation for feature selection (Bhattacharyya, 2004).

This paper focuses on both *effective* and *data-aware* hashing for the indexing of large scale database, which means we tend to design compact binary codes for a large dataset of any kind so that semantically similar datapoints are mapped to similar codes within a short Hamming distance. We introduce Sparse Principal Component Analysis (Sparse PCA) and Boosting Similarity Sensitive Coding into Spectral Hashing, and propose a new indexing method *sparse spectral hashing* (**SSH**) (Fig. 1 is an example for image retrieval).

2. Sparse spectral hashing

In this section, we first give the notation used and then the formulation of sparse spectral hashing as well as its solution are discussed.

2.1. Notation and motivation

Assume that we have a collection of N d-dimensional datapoints in Euclidean space $\{(\boldsymbol{x}_i) \in \mathbb{R}^d : i = 1, 2, \ldots, N\}$, where $\boldsymbol{x}_i$ represents the feature vector for the ith datapoint, d represents the dimension of the features from the training data. Θ is an efficient indexing function to map each $\boldsymbol{x}_i$ from d-dimensional Euclidean space to m-dimensional Hamming space $\boldsymbol{y}_i$, we define Θ as follows:

$$\Theta : \boldsymbol{x}_i \in \mathbb{R}^d \to \boldsymbol{y}_i \in \{-1, 1\}^m \tag{1}$$

The indexing function Θ defined above have following good characteristics: (1) Θ is a semantic hashing function. That is to say, if the distance between the ith datapoint $\boldsymbol{x}_i$ and the jth datapoint $\boldsymbol{x}_j$ is small in term of Euclidean distance in $\mathbb{R}^d$ space, their distance in term of Hamming distance in $\{-1,1\}^m$ space is also small; (2) Θ tends to generate compact binary code $\boldsymbol{y}_i$, which means we can get the best evaluation with a small number of bits. Therefore, we need to remove the affect of redundancy. For each datapoint, only a few features are implemented to index this datapoint. (3) The coding is data-aware. Our algorithm get certain adjustment for different datasets.

In the following sections, the N d-dimensional real vectors can also be written as $\boldsymbol{X} \in \mathbb{R}^{N \times d}$ and their corresponding N m-dimensional binary coded datapoints are written as $\boldsymbol{Y} \in \{-1,1\}^{N \times m}$. It should be pointed out here that the ith row in $\boldsymbol{X}$ and the ith row in $\boldsymbol{Y}$ respectively denote the ith original datapoint and its corresponding binary coded counterpart.

2.2. Sparse PCA by convex relaxation

According to Weiss et al. (2009), a good code should be efficient and similarity preserving, which means that each bit of code has a probability of being -1 or 1 equally and uncorrelated to each other. SH formalizes such requirements being a good code as a particular problem of thresholding a subset of eigenvectors of the Laplacian graph as follows (Weiss et al., 2009):

$$\begin{aligned} \text{minimize}: \quad & \sum_{i,j=1}^{N} \boldsymbol{W}(i,j)\|\boldsymbol{y}_i - \boldsymbol{y}_j\|^2 \\ \text{subject to}: \quad & \boldsymbol{y}_i \in \{-1,1\}^m \\ & \sum_{i=1}^{N} \boldsymbol{y}_i = 0 \\ & \tfrac{1}{n}\sum_{i=1}^{N} \boldsymbol{y}_i \boldsymbol{y}_i^T = \boldsymbol{I} \end{aligned} \tag{2}$$

where $\boldsymbol{W} \in \mathbb{R}^{N \times N}$ is the similarity matrix and $\boldsymbol{W}(i,j) = exp(-\|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2/\sigma^2)$. $\boldsymbol{y}_i \in \{-1,1\}^m$ guarantees that the indexing code is binary, $\sum_{i=1}^{N} \boldsymbol{y}_i = 0$ guarantees that each bit has 50% to be -1 or 1 when we choose 0 as a threshold, and $\frac{1}{n}\sum_{i=1}^{N} \boldsymbol{y}_i \boldsymbol{y}_i^T = \boldsymbol{I}$ guarantees that the bits are uncorrelated to each other. $\|\boldsymbol{y}_i - \boldsymbol{y}_j\|$ is Hamming distance.

Though the solution of formula (2) is NPC problem, by introducing Laplacian matrix and removing the constraint $\boldsymbol{y}_i \in \{-1,1\}^m$, (Weiss et al., 2009) turned it into graph partition so that the solution is simply the m eigenvectors of Laplacian matrix $\boldsymbol{L}$ with minimal eigenvalue, where $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{W}$ and $\boldsymbol{D}$ is a diagonal matrix

with its entries $\boldsymbol{D}(i,i) = \Sigma_{j=1}^{N} \boldsymbol{W}(i,j)$. As a result, the solution of formula (2) is transformed into Laplacian eigenmap dimension reduction and then PCA is directly utilized to make the dimensions uncorrelated with each other by SH Weiss et al. (2009).

However, PCA suffers from the fact that its Principal Components (PCs) is lacking of sparseness, i.e., all the data coordinates participate in the linear combination (Zou et al., 2006). Usually, the features used to represent the dataset is often *over-complete*. Intuitively, given an datapoint, only a few features are enough to represent this datapoint. Take image as an example, color-related visual words could be salient features to represent an image of rainbow and shape-related visual words are better to distinguish an image of car from others.

As a result, we introduce a sparse factor into SH, make its PCs be obtained with sparse loadings, so that the linear combination of only several features is used during indexing. We call this approach Sparse Spectral Hashing (SSH).

Suppose the sparse loadings of the Laplacian $\boldsymbol{L}$ is $\boldsymbol{p}$ which is a d-dimension vector. We can penalize the cardinality of $\boldsymbol{p}$ from formula (2), then we have following optimized problem:

$$\begin{aligned} &\text{minimize}: \ \boldsymbol{p}^T\boldsymbol{L}\boldsymbol{p} + \rho\mathbf{Card}^2(\boldsymbol{p}) \\ &\text{subject to}: \boldsymbol{p}^T\mathbf{1} = 0 \\ &\qquad\qquad \boldsymbol{p}^T\boldsymbol{p} = 1 \end{aligned} \tag{3}$$

Where $\mathbf{Card}(\boldsymbol{p})$ is the cardinality of $\boldsymbol{p}$ and the parameter $\rho > 0$ controls the magnitude of the penalty, i.e. the sparsity of the loadings. This problem of formula (3) is also NP-hard. Fortunately, according to DAspremont et al. (2004), we can find a convex relaxation of formula (3) and get its semidefinite relaxation as follows:

$$\begin{aligned} &\text{minimize}: \ \text{trace}(\boldsymbol{L}\boldsymbol{P}) + \rho\mathbf{1}^T|\boldsymbol{P}|\mathbf{1} \\ &\text{subject to}: \boldsymbol{p}^T\mathbf{1} = 0 \end{aligned} \tag{4}$$

Where $\boldsymbol{P} = \boldsymbol{p}\boldsymbol{p}^T$. $|\boldsymbol{P}|$ is the matrix whose elements are the absolute values of the elements of $\boldsymbol{P}$. This is a semidefinite program and can be solved by an iteration approach (DAspremont et al., 2004).

Until now, we can transform the datapoints in training data from d-dimensional Euclidean space to m-dimensional binary Hamming space by thresholding the eigenvectors. However, it is necessary to deal with the problem of *out of sample*, that is to say, how to encode the datapoints out of training set.

Some solutions of out-of-sample have been proposed in past years such as (Bengio et al., 2004). The basic idea of those extensions is to transform eigenvectors to *eigenfunctions*. By the assumption that each $\boldsymbol{x}_i$ is embedded in a manifold subspace and the probability distribution of $\boldsymbol{x}_i$ in the embedded manifold is multidimensional uniform distribution, the *eigenfunctions* of the weighted Laplace–Beltrami operators defined over the embedded manifold is learned in (Weiss et al., 2009) to resolve the problem of out of samples. Belkin and Niyogi (2008) gave out a theoretical proof that the discrete Laplacian defined by n data sampled from an embedded manifold subspace converges to a eigenfunctions as the number of data go to infinity.

The same approach is used in (Weiss et al., 2009) to work out the problem of out-of-sample.

2.3. The data-oblivious binary coding

Given N training images $\boldsymbol{X} \in \mathbb{R}^{N\times d}$, this section will discuss how to find an indexing function Θ, so that Θ transforms $\boldsymbol{X}$ in d-dimensional Euclidean space to $\boldsymbol{Y} \in \{-1,1\}^{N\times m}$ in m-dimensional Hamming space. The Θ can be learned by following two steps.

Step 1 : The calculations of m sparse PCs via convex relaxation by formula (4). This step will map $\boldsymbol{X} \in \mathbb{R}^{N\times d}$ into $\boldsymbol{B} \in \mathbb{R}^{N\times m}$.

Given the covariance matrix $\boldsymbol{\Sigma}$ of $\boldsymbol{X}$ ($\boldsymbol{\Sigma} \in \mathbb{R}^{d\times d}$), a d-dimensional sparse PC $\boldsymbol{p}$ can be obtained according to convex relaxation mentioned before. Then the covariance matrix $\boldsymbol{\Sigma}$ is updated as following:

$$\boldsymbol{\Sigma} := \boldsymbol{\Sigma} - (\boldsymbol{p}^T\boldsymbol{\Sigma}\boldsymbol{p})\boldsymbol{p}\boldsymbol{p}^T \tag{5}$$

The update of $\boldsymbol{\Sigma}$ in formula (5) is repeated by m times and convex relaxation is implemented to each updated $\boldsymbol{\Sigma}$. By this way, we can get m sparse PCs and construct a matrix $\boldsymbol{M} \in \mathbb{R}^{d\times m}$ whose columns are the sparse PCs. The matrix $\boldsymbol{B}$ can be defined by matrix multiplication as $\boldsymbol{B} = \boldsymbol{X}\boldsymbol{M}$.

Step 2 : The mapping of $\boldsymbol{B}$ in Euclidean space to $\boldsymbol{Y}$ in Hamming space.

We denote the jth ($j = 1,\ldots,m$) column of matrix $\boldsymbol{B}$ as $\boldsymbol{B}_{(:,j)}$. Obviously, the cardinality of $\boldsymbol{B}_{(:,j)}$ is N. For each $\boldsymbol{B}_{(:,j)}$, a function δ_j^k is defined as follows (Weiss et al., 2009):

$$\delta_j^k = 1 - e^{-\frac{\varepsilon^2}{2}\left|\frac{k\pi}{\boldsymbol{B}_{(:,j)}^{max} - \boldsymbol{B}_{(:,j)}^{min}}\right|^2} \tag{6}$$

where $k = 1,\ldots,m$, $\boldsymbol{B}_{(:,j)}^{max}$ and $\boldsymbol{B}_{(:,j)}^{min}$ are respectively the maximum and minimum values in $\boldsymbol{B}_{(:,j)}$, and ε is a constant. Thus, the number of δ_j^k generated by each $\boldsymbol{B}_{(:,j)}$ is m, and the number of δ_j^k ($k = 1,\ldots,m$; $j = 1,\ldots,m$) in total is $m \times m$. After ranking those δ_j^k, we only choose the first m minimum ones denoted as $\left\{\delta_1^{min},\ldots,\delta_m^{min}\right\}$. Given the ith image $\boldsymbol{x}_i \in \mathbb{R}^d$ in training set, assume its corresponding binary code is $\boldsymbol{y}_i \in \{-1,1\}^m$ and the value after mapping is $\boldsymbol{z}_i \in (-1,1)^m$. The jth binary value $\boldsymbol{z}(i,j)$ in $\boldsymbol{z}_i$ can be calculated as:

$$\boldsymbol{z}(i,j) = \Theta\left(\delta_j^{min}, \boldsymbol{B}(i,t)\right) = \sin\left(\frac{\pi}{2} + \frac{k\pi}{\boldsymbol{B}_{(:,t)}^{max} - \boldsymbol{B}_{(:,t)}^{min}}\boldsymbol{B}(i,t)\right) \tag{7}$$

where δ_j^{min} is the jth minimum value in $\left\{\delta_1^{min},\ldots,\delta_m^{min}\right\}$ and assume that δ_j^{min} is generated by k and the t column of $\boldsymbol{B}$, $\boldsymbol{B}(:,t)^{max}$ and $\boldsymbol{B}(:,t)^{min}$ are the maximum and minimum values in $\boldsymbol{B}(:,t)$, $i = (1,\ldots,N)$ and $j = (1,\ldots,m)$. After obtaining $\boldsymbol{z}(i,j)$, we can simply take a threshold to generate binary code $\boldsymbol{y}(i,j)$.

2.4. Binary coding by boosting in SSH

Theoretically, zero is the appropriate threshold for each $\boldsymbol{z}(i,j)$ to quantize data from high-dimensional Euclidean space to low-dimensional Hamming space since the mapping function in formula (7) is a sine function whose range is $[-1,1]$. However, real world data does not always have uniform distribution and the *data-oblivious* binary coding is apparently very restrictive. As a result, a *data-aware* threshold learned from dataset is necessary and much more flexible to improve the hashing performance. Similar like (Shakhnarovich et al., 2003), here we introduce AdaBoost into SSH in order to find a more practical threshold for the quantization of binary code, which is called AdaSSH.

We get a matrix $\boldsymbol{Z}$ whose element in ith row and jth column is $\boldsymbol{z}(i,j)$. Each column of $\boldsymbol{Z}$ is taken as a weak learner, so that we can transfer the encode problem into a classifier problem. Take the threshold of the jth column of $\boldsymbol{Z}$ as T_j, then we can get the binary code $\boldsymbol{y}(i,j)$ according to the following rule:

$$\boldsymbol{y}(i,j) = \begin{cases} +1, \textbf{if } \boldsymbol{z}(i,j) \leqslant T_j \\ -1, \text{otherwise} \end{cases} \tag{8}$$

Then the classification implied by formula (8) is $\hat{f}(u,v) = \boldsymbol{y}(u,j)\boldsymbol{y}(v,j)$, which predicts whether the uth image and the vth image is similar or not by the jth column. The optimal threshold T_j for this column can be found in the following steps:

Step 1 : we get each image pairs from dataset. As we have N images in the dataset, we can get N^2 pairs. For each pair, we can assign it a label f which is 1 or -1 determined by that whether the two images in the pair are near neighbors or not. Then we get N^2 triads which can be denoted by $(\boldsymbol{z}(u,j), \boldsymbol{z}(v,j), f(u,v))$, where $\boldsymbol{z}(u,j)$, $\boldsymbol{z}(v,j)$ denotes the mapping value of the uth and vth image in the jth column and $f(u,v)$ is the corresponding label.

Step 2 : we try each potential value as the threshold to calculate the error rate, then take the value related to the minimum error rate as the threshold of the current dimension.

Algorithm 1 gives out the procedure

Algorithm 1. Threshold learning by boosting

Input N^2 triads $(\boldsymbol{z}(u,j),\boldsymbol{z}(v,j),f(u,v))$
1: Empty array A and $T_n = 0$, which means the number of negative pairs
2: ***For*** each triad
  if $\boldsymbol{z}(u,j) > \boldsymbol{z}(v,j)$ ***then*** $l_1 = 1$
  else if $\boldsymbol{z}(u,j) < \boldsymbol{z}(v,j)$***then*** $l_1 = -1$
  else $l_1 = 0$
  end if
  $l_2 = -l_1$
  $A = A \cup \{(\boldsymbol{z}(u,j),l_1,f(u,v)), (\boldsymbol{z}(v,j),l_2,f(u,v))\}$
  $T_n = T_n + 1$ **if** $f(u,v) == -1$
End For
3: Sort A increasingly by $\boldsymbol{z}$ which has $2N^2$ elements now
$s_p = s_n = 0$
$c_b = T_n$, $T_j = min(\boldsymbol{z}) - eps$
4: ***For*** $k = 1:2N^2$
  $(\boldsymbol{z},l,f) = A[k]$
  if $f == 1$ ***then***
    $s_p = s_p - l$
  else if $f == -1$ ***then***
    $s_n = s_n - l$
  end if
  $c = T_n - s_n + s_p$
  if $c < c_b$ ***then***
    $c_b = c, T_j = \boldsymbol{z}$
  end if
End for
Output T_j, which is the threshold of the jth column of $\boldsymbol{Z}$

Table 1
The summary of average performance on Paris in term of F1 and AUC over different dimensional vectors with different length of code (m).

m	F1				AUC			
	300	500	800	1000	300	500	800	1000
2	0.1355	0.1636	0.1757	0.1924	0.5369	0.5339	0.5311	0.5343
4	0.1356	0.1638	0.1757	0.1924	0.5370	0.5345	0.5311	0.5344
8	0.1357	0.1652	0.1779	0.1926	0.5378	0.5411	0.5382	0.5347
16	0.2148	0.2218	0.2495	0.2507	0.7383	0.6465	0.6909	0.6631

Repeat the algorithm 1 and we can get the threshold for each column of matrix $\boldsymbol{Z}$. Therefore the learned thresholds are conducted together to generate binary code matrix $\boldsymbol{Y}$ later.

2.5. The description of SSH

Algorithm 2 gives out the detailed description of our proposed SSH.

Algorithm 2. Sparse spectral hashing

Input The training set $\boldsymbol{X}$ and length of code m
1: Calculate covariance matrix $\boldsymbol{\Sigma}$ of $\boldsymbol{X}$
2: ***For*** $i = 1:m$
  calculate the sparse PC $\boldsymbol{p}_i$ of $\boldsymbol{\Sigma}$
  update $\boldsymbol{\Sigma}$ by formula (5)
End For
3: Get matrix $\boldsymbol{M}$ whose ith column is $\boldsymbol{p}_i$
$\boldsymbol{B} = \boldsymbol{XM}$.
4: Calculate $m \times m$ δ of matrix $\boldsymbol{\Sigma}$ with formula (6) (set ϵ as 1) and sort with ascending order
5: Map $\boldsymbol{B}$ into $\boldsymbol{Z}$ by formula (7)
6: Determine threshold of each column in $\boldsymbol{Z}$ by algorithm 1 to get binary code $\boldsymbol{Y}$
Output The binary code for each image

For the datapoints outside the training set such as a query, the step 3 and 5 in Algorithm 2 are implemented to generate its binary code with m bits.

After the binary code of each datapoint is obtained, the Hamming distance between codes is used here to measure the similarity of datapoints.

3. Experiments

In SSH, we can use zero as a threshold to transform data from high-dimensional Euclidean space to low-dimensional hamming space, or we can use Adaboost to learn a threshold for the generation of binary codes. In our experiments, we call the threshold is learned by Adaboosting in SSH as AdaSSH. Here, we report the comparisons of our proposed SSH/Adaboost with SH (Weiss et al., 2009), LSH (Gionis et al., 1999), E2LSH (Datar et al., 2004), RBM (Salakhutdinov and Hinton, 2009) and PSH (Shakhnarovich et al., 2003). Additionally, we compare all the algorithms with a baseline method, in which we first use PCA to reduce the dimensions of data in original space to the target subspace, then find the nearest neighbors (similar items) in subspace based on the the threshold which is set as 1.5% of the average Euclidean distance of all data in subspace. This baseline method is called Baseline in all experiments.

In each experiment, the dataset is hashed to Hamming space by each hashing method respectively, the hash result of a query (outside the dataset) is taken as the center of a Hamming ball, a certain Hamming distance is taken as the radius. All the datapoints of the

Table 2
The summary of average performance for the compared algorithms and baseline on Paris in term of F1 and AUC with different length of code (m).

m	F1								AUC							
	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline
2	0.1355	0.1886	0.1335	0.1372	0.1356	0.1906	0.1356	0.1355	0.5369	0.6833	0.5275	0.5443	0.5370	0.6255	0.5370	0.5365
4	0.1356	0.1886	0.1214	0.0922	0.1356	0.0909	0.1356	0.1354	0.5370	0.6833	0.4775	0.5025	0.5370	0.4201	0.5370	0.5360
8	0.1357	0.2011	0.1265	0.1333	0.1356	0.3320	0.1358	0.1348	0.5378	0.7027	0.4983	0.5653	0.5370	0.6596	0.5379	0.5332
16	0.2148	0.2873	0.1209	0.0004	0.1356	0.0094	0.1425	0.1347	0.7383	0.8111	0.4772	0.5325	0.5370	0.5378	0.5648	0.5328

dataset in the Hamming ball are taken as the search result of the corresponding method.

Though usually LSH and E2LSH are used to create a candidate list which is been exhausted search to determine the result, with the sharp rise in the amount of data, this method is getting more and more impractical, so we choose to return all the datapoints in the Hamming ball as the search result.

3.1. Evaluation data set

Two kinds of datasets are used in our experiments:

(1) **Image dataset**. Paris (Philbin et al., 2008), Oxford5k (Philbin et al., 2007), MCG-WEBV (Cao et al., 2009) and USPS-Digit are used in this experiment. We choose SIFT features as local features for image datasets. *K*-means clustering is implemented to generate visual words for each dataset respectively, thus each image is represented by a vector whose elements denote the frequency the corresponding visual word appears in that image.
(2) **Text dataset**. 20 Newsgroups and Reuters-21578 are used in this experiment. We choose tf*idf as local features for text datasets. For each dataset, we do the word segmentation, remove the stop words and calculate the tf*idf of each word in the whole dataset to determine the key words. Then we calculate the tf*idf of key words in each text respectively to represent the corresponding text.

3.2. The evaluation metrics

We calculate the Euclidean distance between every two vectors in a dataset and take 0.015 multiple the average Euclidean distance of the dataset as the threshold of near neighbors, which is also used by Weiss et al. (2009). We choose Euclidean distance instead of groundtruth provided by dataset because this paper concentrates on the projection from the Euclidean space to the Hamming space with similarity preserving.

The F1 score (harmonic mean of precision and recall) and AUC (the area under the ROC curve) are used to measure the performance for image indexing.

3.3. Experimental results

3.3.1. Image dataset

(1) **Paris**. Paris consists of 6412 images from Flickr. We choose 119 images as queries randomly and remain the others as the dataset. Firstly, 300, 500, 800, 1000 visual words are generated alternatively from this dataset, so that each image is represented as a 300, 500, 800, 1000 dimensional vector respectively. Table 1 shows the average performance by SSH in term of F1 and AUC, where *m* is the length of the projected code.

As shown in Table 1, we can see that as we removed the redundancy of the features, most of the performance of the same length of code is similar despite the different dimension. So we just choose 300 as the default dimension of vector for the following datasets.

The performance of seven hashing methods and the baseline method is shown in Table 2 with 300 dimensional vector as the feature representation.

(2) **Oxford5k**. Oxford5k consists of 5062 images including 50 queries. This dataset serves as groundtruth in all experiments. The performance of seven hashing methods and the baseline method is shown in Table 3 with 300 dimensional vector as the feature representation.

(3) **MCG-WEBV**. This dataset consists of 248,887 most viewed videos for every month from December 2008 to November 2009 on YouTube. Each video has several keyframes. As the dataset is extremely large, we randomly choose 4950 images for the database and 203 images as the queries. As this dataset provided 1000 dimensional vectors to represent the keyframes in the dataset,

Table 3
The summary of average performance for the compared algorithms and baseline on Oxford5k in term of F1 and AUC with different length of code (*m*).

m	F1								AUC							
	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline
2	0.0959	0.0821	0.0475	0.0583	0.0560	0.0983	0.0865	0.0549	0.6590	0.6448	0.3933	0.4945	0.4657	0.7054	0.6223	0.4545
4	0.0963	0.1799	0.0469	0.0006	0.0560	0.1589	0.0869	0.0549	0.6601	0.7820	0.3830	0.4266	0.4659	0.8107	0.6236	0.4545
8	0.1345	0.2437	0.0495	0.0026	0.0873	0.1251	0.1038	0.0549	0.7216	0.8026	0.4149	0.4477	0.6654	0.5999	0.6655	0.4545
16	0.1630	0.5274	0.0517	0.0000	0.0560	0.0946	0.2049	0.0549	0.7444	0.7796	0.4383	0.4545	0.4659	0.4852	0.7637	0.4545

Table 4
The summary of average performance for seven compared algorithms and baseline on MCG-WEBV in term of F1 and AUC with different length of code (*m*).

m	F1								AUC							
	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline
2	0.2617	0.3435	0.2607	0.2620	0.2677	0.3065	0.2635	0.2596	0.5180	0.6593	0.5152	0.5186	0.5304	0.6069	0.5229	0.5126
4	0.2620	0.3817	0.2629	0.2641	0.2596	0.3117	0.2637	0.2583	0.5184	0.7006	0.5209	0.5247	0.5127	0.6053	0.5231	0.5111
8	0.2687	0.4143	0.2618	0.2647	0.2681	0.2957	0.3171	0.2585	0.5315	0.7249	0.5180	0.5271	0.5316	0.5985	0.6269	0.5137
16	0.3187	0.4378	0.2618	0.2641	0.2676	0.1940	0.1717	0.2577	0.6147	0.7077	0.5180	0.5293	0.5304	0.5647	0.4919	0.5137

Table 5
The summary of average performance for the compared algorithms and baseline on USPS-Digit in term of F1 and AUC with different length of code (*m*).

m	F1								AUC							
	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline
2	0.1175	0.1238	0.0550	0.0498	0.0992	0.0962	0.1176	0.0557	0.7314	0.7358	0.5532	0.5325	0.7473	0.7010	0.7316	0.5617
4	0.1730	0.1847	0.0547	0.0435	0.1879	0.1437	0.1720	0.0556	0.7430	0.7309	0.5477	0.5679	0.7563	0.6494	0.7419	0.5603
8	0.1863	0.2118	0.0546	0.0063	0.1451	0.2132	0.1817	0.0556	0.6222	0.6472	0.5472	0.5671	0.6098	0.6224	0.6217	0.5602
16	0.0234	0.0942	0.0547	0.0000	0.0291	0.1680	0.0209	0.0556	0.5649	0.5791	0.5468	0.5606	0.5661	0.5801	0.5646	0.5606

Table 6
The summary of average performance for the compared algorithms and baseline on 20 Newsgroups in term of F1 and AUC with different length of code (m).

m	F1								AUC							
	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline
2	0.2680	0.3700	0.2615	0.1663	0.2624	0.2791	0.2682	0.2595	0.4849	0.6829	0.4671	0.4562	0.4684	0.5166	0.4852	0.4715
4	0.2739	0.4804	0.2615	0.0952	0.2625	0.3460	0.2744	0.2581	0.4993	0.7746	0.4674	0.4648	0.4686	0.6387	0.5005	0.4731
8	0.3458	0.5026	0.2638	0.0471	0.2626	0.3221	0.3058	0.2601	0.6240	0.7797	0.4734	0.4807	0.4687	0.5884	0.5807	0.4685
16	0.4231	0.5415	0.2615	0.0001	0.2870	0.3347	0.3776	0.2582	0.6591	0.7479	0.4674	0.4660	0.5327	0.5914	0.6309	0.4726

Table 7
The summary of average performance for the compared algorithms and baseline on Reuters-21578 in term of F1 and AUC with different length of code (m).

m	F1								AUC							
	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline
2	0.5345	0.6228	0.4106	0.2147	0.5914	0.4536	0.5297	0.4068	0.7430	0.8368	0.6540	0.5746	0.8159	0.6997	0.7406	0.6463
4	0.4405	0.6508	0.4058	0.0807	0.2982	0.3853	0.4487	0.4058	0.6860	0.8517	0.6437	0.6202	0.6694	0.6188	0.6944	0.6437
8	0.4107	0.6766	0.4106	0.0243	0.1642	0.5571	0.3988	0.4054	0.7038	0.8517	0.6540	0.6510	0.6634	0.7691	0.7044	0.6410
16	0.2917	0.6968	0.4057	0.0000	0.0589	0.4917	0.1904	0.4054	0.6992	0.8435	0.6434	0.6489	0.6556	0.7323	0.6780	0.6386

Table 8
The runtime comparison of eight algorithms on six datasets.

	SSH	AdaSSH	LSH	E2LSH	RBM	PSH	SH	Baseline
Paris	158.667	5127.916	5.444	5.616	7510.571	5075.500	4.353	4.805
Oxford5k	160.025	3222.576	2.528	2.730	4327.635	3072.768	1.966	2.496
MCG-WEBV	197.902	3120.614	6.037	8.409	67081.974	3474.547	6.942	6.053
USPS-Digit	93.990	3205.135	10.827	11.840	2428.315	3175.183	10.405	10.483
20 Newsgroups	197.902	2437.114	3.682	4.508	5048.247	2516.206	3.027	3.401
Reuters-21578	159.463	2439.875	3.572	4.431	2964.146	2512.946	2.948	3.385

we just use it as the local feature representation.The performance of seven hashing methods and the baseline method is shown in Table 4.

(4) **USPS-Digit**. This dataset consists of roughly 10 K samples for digits. We randomly choose 5000 images as the training set and 400 images as queries.The images are represented by 256 dimensional vectors provided by the dataset. The performance of seven hashing methods and the baseline method is shown in Table 5.

3.3.2. Text dataset

(1) **20 Newsgroups**. This dataset consists of 20 topics, 18846 articles. After preprocessing, 11315 valid articles are chosen to be presented into vectors (some articles are too short to use). We choose 4500 articles randomly as database and other 100 articles as queries. The performance of seven hashing methods and the baseline method is shown in Table 6 with 300 dimensional vector as the feature representation.

(2) **Reuters-21578**. This dataset consists of 21578 articles in 135 categories. We choose 4500 articles randomly as database and other 100 articles as queries. The performance of seven hashing methods and the baseline method is shown in Table 7 with 300 dimensional vector as the feature representation.

We can see that the performance achieved by AdaSSH is almost superior to those other semantic hashing methods. The indexing by machine learning approaches (such as AdaSSH, SSH, SH and RBM) is better than the indexing by random projection (such as LSH and E2LSH). All indexing algorithms are generally better than the baseline method. From the experiments, we can know that the threshold in Spectral Hashing is very important since the performance of AdaSSH is better than the performance of SSH.

3.3.3. Runtime

In this section, we compare the runtime complexity of all eight algorithms on the six datasets. The runtime complexity means the time an indexing algorithm is used to generate the 2, 4, 8, 16-bit binary codes on a dataset. Each algorithm is run three times on each dataset, and the average runtime is recorded. The unit is second. Table. 8 shows the results.

From Table 8, we can know that RBM is the slowest algorithm, and the two algorithms AdaSSH and PSH which both use Boosting are obviously slower than the others. Boosting take a lot of time to determine the threshold. Though SSH is slower than LSH, E2LSH and SH, the results of SSH are generally better than the rest.

4. Conclusion

We have extended spectral hashing to sparse spectral hashing for effective and data-aware image indexing. In order to obtain globally optimal solution of sparse PCA, convex relaxation is introduced in this paper. The experimental results show that the performance achieved by SSH is better than those of others.

Acknowledgement

This work was partially supported by the National Natural Science Foundation of China (No. 61070068, No. 60833006), 973 Program (2009CB320801).

References

Belkin, M., Niyogi, P., 2008. Towards a theoretical foundation for Laplacian-based manifold methods. J. Comput. Syst. Sci. 74, 1289–1308.

Bengio, Y., Paiement, J., Vincent, P., Delalleau, O., Le Roux, N., Ouimet, M., 2004. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In: Proc. Advances in Neural Information Processing Systems (NIPS), p. 177.

Berchtold, S., Bohm, C., Jagadish, H., Kriegel, H., Sander, J., 2000. Independent quantization: An index compression technique forhigh-dimensional data spaces. In: Proc. Internat. Conf. on Data Engineering (ICDE), pp. 577–588.

Bhattacharyya, C., 2004. Second order cone programming formulations for feature selection. J. Mach. Learn. Res. 5, 1417–1433.

Cao, J., Zhang, Y., Song, Y., Chen, Z., Zhang, X., Li, J., 2009. MCG-WEBV: A Benchmark Dataset for Web Video Analysis. Technical Report ICT-MCG-09-001. Institute of Computing Technology.

DAspremont, A., Ghaoui, L.E., Jordan, M.I., Lanckriet, G.G., 2004. A direct formulation for sparse PCA using semidefinite programming. In: Proc. Advances in Neural Information Processing Systems (NIPS).

Datar, M., Immorlica, N., Indyk, P., Mirrokni, V., 2004. Locality-sensitive hashing scheme based on p-stable distributions. Proc. Annals of Symposium on Computational Geometry. ACM, pp. 253–262.

Fei, W., Yahong, H., Qi, T., Yueting, Z., 2010. Multi-label Boosting for Image Annotation by Structural Grouping Sparsity. In: Proc. 2010 ACM Internat. Conf. on Multimedia (ACM Multimedia), pp. 15–24.

Gionis, A., Indyk, P., Motwani, R., 1999. Similarity search in high dimensions via hashing. Proc. 25th Internat. Conf. on Very Large Data Bases. Morgan Kaufmann Publishers Inc., pp. 518–529.

Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K., 2007. Multi-probe LSH: Efficient indexing for high-dimensional similarity search. In: Proc. Internat. Conf. on Very Large Data Bases (VLDB), pp. 950–961.

Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A., 2008. Lost in quantization: Improving particular object retrieval in large scale image databases. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 1–8.

Philbin, J., Chum, O., Isard, M., Sivic, J., Zisserman, A., 2007. Object retrieval with large vocabularies and fast spatial matching. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition, pp. 1–8.

Salakhutdinov, R., Hinton, G., 2007. Learning a nonlinear embedding by preserving class neighbourhood structure. In: AI and Statistics.

Salakhutdinov, R., Hinton, G., 2009. Semantic hashing. Int. J. Approx. Reason. 50, 969–978.

Shakhnarovich, G., Viola, P., Darrell, T., 2003. Fast pose estimation with parameter-sensitive hashing, in: Proceedings of IEEE International Conference on Computer Vision (ICCV), p. 750.

Weiss, Y., Torralba, A., Fergus, R., 2009. Spectral hashing, in: Proceedings of Advances in Neural Information Processing Systems (NIPS), pp. 1753–1760.

Zou, H., Hastie, T., Tibshirani, R., 2006. Sparse principal component analysis. J. Comput. Graph. Stat. 15, 265–286.