Weighted Hashing for Fast Large Scale Similarity Search

Qifan Wang
Computer Science Department
Purdue University
West Lafayette, IN 47907, US
wang868@purdue.edu

Dan Zhang
Facebook Incorporation
Menlo Park, CA 94025, US
danzhang@fb.com

Luo Si
Computer Science Department
Purdue University
West Lafayette, IN 47907, US
lsi@purdue.edu

ABSTRACT

Similarity search, or finding approximate nearest neighbors, is an important technique for many applications. Many recent research demonstrate that hashing methods can achieve promising results for large scale similarity search due to its computational and memory efficiency. However, most existing hashing methods treat all hashing bits equally and the distance between data examples is calculated as the Hamming distance between their hashing codes, while different hashing bits may carry different amount of information. This paper proposes a novel method, named Weighted Hashing (WeiHash), to assign different weights to different hashing bits. The hashing codes and their corresponding weights are jointly learned in a unified framework by simultaneously preserving the similarity between data examples and balancing the variance of each hashing bit. An iterative coordinate descent optimization algorithm is designed to derive desired hashing codes and weights. Extensive experiments on two large scale datasets demonstrate the superior performance of the proposed research over several state-of-the-art hashing methods.

Categories and Subject Descriptors

H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval

Keywords

Hashing, Similarity Search

1. INTRODUCTION

Similarity search, also known as approximate nearest neighbor search, has many applications such as content-based image retrieval, similar document detection and collaborative filtering. Many real world applications of similarity research need to process a huge amount of data within a high-dimensional space to answer a query. Simple similarity search methods in the original high-dimensional space do not scale up due to the excessive cost in storage and processing when the data size grows. Recently, hashing methods [5, 10] have been successfully used for approximating nearest neighbor search due to its fast query speed and low storage cost. The basic idea of hashing is to design compact binary codes in a low-dimensional space for data points and preserve their similarities. More specifically, each data point will be hashed/mapped into a compact binary code, and similar points in the original feature space should be hashed into close points in the binary hashing code space. Hashing methods have become a promising and popular choice for efficient similarity search due to its two main advantages, reduced storage cost and fast query time.

Locality-Sensitive Hashing (LSH) [1] is one of the most commonly used data-independent hashing methods. It utilizes random linear projections, which are independent of training data, to map data points from a high-dimensional feature space to a low-dimensional binary space. Another class of hashing methods are called data-dependent methods, whose projection functions are learned from training data. These data-dependent methods include spectral hashing (SH) [10], principal component analysis based hashing (PCAH) [4], self-taught hashing (STH) [12] and iterative quantization (ITQ) [2]. SH learns the hashing codes based on spectral graph partitioning and forcing the balanced and uncorrelated constraints into the learned codes. PCAH utilizes principal component analysis (PCA) to learn the projection functions. STH combines an unsupervised learning step with a supervised learning step to learn effective hashing codes. ITQ learns an orthogonal rotation matrix to refine the initial projection matrix learned by PCA so that the quantization error of mapping the data to binary codes is minimized. Compared with the data-independent methods, these data-dependent methods generally provide more effective hashing codes.

Hashing methods generate promising results by successfully addressing the storage and search efficiency challenges. However, most existing hashing methods treat all hashing bits equally and the similarity between data examples is represented by the Hamming distance between their hashing codes. But different hashing bits carry different amount of information (e.g., [3]), where the hashing bit with larger variance contains more information. Therefore, it is unreasonable to use a same weight for different hashing bits.

This paper proposes a novel hashing method, named Weighted Hashing (WeiHash), to assign different weights on different hashing bits to capture their importance. The hashing codes and their corresponding weights are jointly learned in a unified framework by simultaneously preserving the similarities between data examples and balancing the variance of each hashing bits. An iterative coordinate descent algorithm is designed as the optimization procedure.

2. WEIGHTED HASHING

This section first states the problem setting of WeiHash. Assume there are total n training data examples, denoted as: $\boldsymbol{X} = \{x_1, x_2, \dots, x_n\} \in \boldsymbol{R}^{n\times m}$, where m is the dimensionality of the feature. The main purpose of weighted hashing is to map these training examples to the optimal binary hashing codes $\boldsymbol{Y} = \{y_1, y_2, \dots, y_n\} \in \{-1, 1\}^{n\times k}$ through a hashing function $f : \boldsymbol{R}^m \to \{-1, 1\}^k$, such that the similarities among data examples in original feature space are preserved in the hashing codes with appropriate weights assigned to hashing bits. Here k is the number of hashing bits and $y_j = f(x_j)$.

2.1 Objective Function

2.1.1 Similarity Preservation

One of the key problems in hashing methods is similarity preserving, which indicates that similar data examples should be mapped to similar hashing codes. Most existing hashing methods [3, 10, 12] define the similarity between two hashing codes accordingly to their Hamming distance. The Hamming distance between two binary codes y_i and y_j is given by the number of different bits between them, which can be calculated as $\frac{1}{4}\|y_i - y_j\|^2$. However, the Hamming distance may not accurately represent the similarity between two hashing codes since different hashing bits carry different amount of information. Therefore, we propose to use the weighted Hamming distance to capture the code similarity as $\frac{1}{4}\|w\cdot(y_i - y_j)\|^2$, where w is a $k\times 1$ weight vector indicating the importance of different bits and $\cdot$ is the element-wise vector multiplication.

To measure how well the similarity between data examples is represented by the binary hashing codes, one natural way is to minimize the weighted Hamming distance as follow:

$$\sum_{i,j=1}^{n} \boldsymbol{S}_{ij}\|w\cdot(y_i - y_j)\|^2 \tag{1}$$

Here, $\boldsymbol{S}$ is the similarity matrix which is calculated based on the original features. To meet the similarity preservation criterion, we seek to minimize this quantity, because it incurs a heavy penalty if two similar examples are mapped far away. There are many different ways to define the similarity matrix $\boldsymbol{S}$. In this paper, we adopt the local similarity due to its nice property in many machine learning applications [9, 12]. In particular, the corresponding similarities are computed by Gaussian functions, *i.e.*, $\boldsymbol{S}_{ij} = e^{-\frac{\|x_i - x_j\|^2}{\sigma_{ij}^2}}$, where σ_{ij} is determined automatically by local scaling [11].

By introducing a diagonal $n \times n$ matrix $\boldsymbol{D}$, whose entries are given by $\boldsymbol{D}_{ii} = \sum_{j=1}^{n}\boldsymbol{S}_{ij}$ and a diagonal $k \times k$ matrix $\boldsymbol{W}$ with diagonal elements to be the weight vector w, Eqn.1 can be rewritten as:

$$tr\left(\boldsymbol{W}\boldsymbol{Y}^T(\boldsymbol{D}-\boldsymbol{S})\boldsymbol{Y}\right) = tr\left(\boldsymbol{W}\boldsymbol{Y}^T\boldsymbol{L}\boldsymbol{Y}\right) \tag{2}$$

where $\boldsymbol{L}$ is the graph *Laplacian* and $tr()$ is the matrix trace. By minimizing this term, the similarity between different data examples can be preserved in the learned hashing codes.

2.1.2 Variance-Weight Proportion Constraint

Different weights are assigned to different hashing bits in this work to better measure the similarity between hashing codes. This is also observed in previous research work that different hashing bits carry different amount of information and the hashing bit with larger variance should contain more information. Therefore, more weight should be assigned to the hashing bit with larger variance and we propose to follow the proportion principle that the weight of a hashing bit should be proportional to its variance denoted as:

$$\frac{w^p}{var(y^p)} = \frac{w^q}{var(y^q)} \quad \forall\ p,\ q \tag{3}$$

where w^p and $var(y^p)$[1] are the weight and variance of the *p-th* hashing bit.

2.1.3 Orthogonality Constraint

In order to maximize the performance of the hashing codes, many existing hashing methods [9, 10, 12] enforce hard orthogonality constraint among hashing bits, *e.g.*, $\frac{1}{n}\boldsymbol{Y}^T\boldsymbol{Y} = \boldsymbol{I}$. The hard orthogonality constraint forces the hashing bits to be uncorrelated with each other. However, these constraints may sometimes be problematic, since most of the variance is contained in a few top projections for many real-world datasets. But the orthogonality constraints often force hashing methods to choose some directions with low variance progressively, which may substantially reduce the effectiveness of hashing codes. Therefore, instead of imposing hard orthogonality constraint on hashing bits, we impose a soft orthogonality term as follows:

$$\|\frac{1}{n}\boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{I}\|_F^2 \tag{4}$$

where $\|\|_F$ is the matrix *Frobenius* norm. The above soft orthogonality constraint has certain tolerance to non-orthogonality, which allows the learning approach to choose appropriate projection directions.

2.2 Optimization Algorithm

The proposed unified framework combines the above three components as follows:

$$\begin{aligned} &\min_{\boldsymbol{Y},\boldsymbol{W}} tr\left(\boldsymbol{W}\boldsymbol{Y}^T\boldsymbol{L}\boldsymbol{Y}\right) + \alpha\|\frac{1}{n}\boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{I}\|_F^2 \\ s.t.\quad &\boldsymbol{Y} \in \{-1,1\}^{n\times k},\ \ \boldsymbol{Y}\boldsymbol{1} = 0 \quad \sum_p w^p = 1 \\ &\frac{w^p}{var(y^p)} = \frac{w^q}{var(y^q)}\ \forall\ p,\ q \end{aligned} \tag{5}$$

The bit balance constraint $\boldsymbol{Y}\boldsymbol{1} = 0$ requires each bit to have equal chance to be 1 or -1 and $\sum_p w^p = 1$ is the weight normalization constraint.

2.2.1 Relaxation

Directly minimizing the objective function in Eqn.5 is intractable because of the discrete constraint. Therefore, we propose to relax this constraint and drop the bit balance constraint $\boldsymbol{Y}\boldsymbol{1} = 0$ first (we will discuss the constraint later). However, even after the relaxation, the objective function is still difficult to optimize since $\boldsymbol{Y}$ and $\boldsymbol{W}$ are coupled together and it is non-convex with respect to $\boldsymbol{Y}$ and $\boldsymbol{W}$ jointly.

[1] $var(y^p) = \frac{1}{n}\sum_{j=1}^{n}(y_j^p - mean(y^p))^2$

We propose to use a coordinate descent algorithm [7] for solving this relaxed optimization problem by iteratively optimizing the objective with respect to $\boldsymbol{Y}$ and $\boldsymbol{W}$. In particular, after initializing $\boldsymbol{W}$, the relaxed problem can be solved by doing the following two steps iteratively similar as [8], until convergence.

Step 1: Fix $\boldsymbol{W}$, optimize w.r.t. $\boldsymbol{Y}$:

$$\min_{\boldsymbol{Y}} tr\left(\boldsymbol{W}\boldsymbol{Y}^T\boldsymbol{L}\boldsymbol{Y}\right) + \alpha\|\frac{1}{n}\boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{I}\|_F^2 \tag{6}$$

The above problem is still non-convex. However, the objective function is differentiable with respect to $\boldsymbol{Y}$ and the gradient of Eqn.6 can be calculated as follows:

$$\partial\frac{Eqn(7)}{\boldsymbol{Y}} = 2\boldsymbol{S}\boldsymbol{Y}\boldsymbol{W} + \frac{4\alpha}{n}\boldsymbol{Y}(\frac{1}{n}\boldsymbol{Y}^T\boldsymbol{Y} - \boldsymbol{I}) \tag{7}$$

With this obtained gradient, L-BFGS Quasi-Newton method is applied to solve this optimization problem.

Step 2: Fix $\boldsymbol{Y}$, solve for $\boldsymbol{W}$:

We can obtain the close form solution of $\boldsymbol{W}$ as:

$$w^p = \frac{var(y^p)}{\sum_{t=1}^k var(y^t)} \tag{8}$$

where $var(y^p)$ can be directly computed from the hashing codes $\boldsymbol{Y}$. By solving Eqns.6 and 8 iteratively, the optimal values of $\boldsymbol{Y}$ and $\boldsymbol{W}$ can be obtained.

2.2.2 Binarization

After obtaining the optimal solution for the relaxed problem, we need to binarize them to obtain binary hashing codes that satisfy the relaxed constraints. The binary hashing codes for the training set can be obtained by thresholding $\boldsymbol{Y}$. It was pointed out in [4] and [9] that desired hashing codes should also maximize the entropy to ensure efficiency. Following the maximum entropy principle, a binary bit that gives balanced partitioning of the whole dataset should provides maximum information. Therefore, we set the threshold for binarizing the *p-th* bit to be the median of y^p. In particular, if *p-th* bit of y_j is larger than median value, y_j^p is set to +1, otherwise y_j^p is set to -1. In this way, the binary code achieves the best balance and the bit balance constraint $\boldsymbol{Y}\boldsymbol{1} = 0$ in Eqn.5 can also be satisfied.

2.2.3 Hashing Function

A linear hashing function is utilized to map data examples to the binary hashing codes as:

$$y_j = f(x_j) = \boldsymbol{H}x_j \tag{9}$$

where $\boldsymbol{H}$ is a $k \times m$ parameter matrix representing the hashing function. Then the optimal hashing function can be obtained by minimizing $\|\boldsymbol{Y} - \boldsymbol{H}\boldsymbol{X}\|^2$.

3. EXPERIMENTS

3.1 Datasets and Implementation

NUS-WIDE dataset is created by NUS lab as a benchmark for evaluating image similarity search techniques. It contains total $269,648$ images. 500-dimensional visual features are extracted using a bag-of-visual-word model with local SIFT descriptor. We randomly partition this dataset into two parts, $268K$ as training data examples and around $1K$ for query test. *MIRFLICKR-1M* dataset is collected from Flicker images for image retrieval and similar visual concept detection tasks. This dataset contains 1 million image examples. 512-dimensional GIST descriptors are extracted from these images and are used as image features for learning the binary hashing codes. We randomly choose $990K$ image examples as the training set and $10K$ for testing. The parameters α and γ are tuned by cross validation on the training set. The number of nearest neighbors is fixed to be 7 when constructing the graph *Laplacian* for all experiments. For LSH, we randomly select projections from a Gaussian distribution with zero-mean and identity covariance to construct the hash tables.

3.2 Evaluation Method

The search results are evaluated based on the labeled semantic tags. If an example shares same semantic tag with a query, it is a relevant example, otherwise it is irrelevant. We use several metrics to measure the performance of different methods. For evaluation with *Hamming Ranking*, we calculate the precision at top k that is the percentage of relevant neighbors among the top k returned examples, where we set k to be 200 in the experiments. We also utilize the precision-recall value [9] for evaluation, which is a widely used metric in information retrieval applications. For evaluation with *Hash Lookup*, all the examples within a fixed weighted Hamming distance, r, of the query are evaluated. In particular, following [6] and [10], a weighted Hamming distance $r = 2$ is used to retrieve the neighbors in the case of *Hash Lookup*. The precision of the returned examples falling within weighted Hamming distance 2 is reported. Note that if a query returns no points inside the Hamming ball with weighted Hamming distance 2, it is treated as zero precision.

3.3 Results and Discussion

The proposed WeiHash approach is compared with five different methods, *i.e.*, Spectral Hashing (SH) [10], PCA Hashing (PCAH) [4], Latent Semantic Hashing (LSH) [1], Self Taught Hashing (STH) [12] and Isotropic Hashing (IsoHash) [3]. IsoHash is compared here due to its superior performance over many other hashing methods such as ITQ and KLSH. We evaluate the performance of different methods by varying the number of hashing bits in the range of $\{8, 16, 32, 64, 128\}$.

Two sets of experiments are conducted on each dataset to evaluate the performance of WeiHash. In the first set of experiments, we report the precision values for the top 200 retrieved examples in Fig.1(a)-(b). The precision values for retrieved examples with weighted Hamming distance 2 are reported in Fig.1(c)-(d). From these comparison results, we can see that WeiHash gives the best performance among all hashing methods on both datasets. In Fig.1(c)-(d), the precision values of *Hash Lookup* decrease significantly with the increasing number of hashing bits. This is because when using longer hashing bits, the Hamming space becomes increasingly sparse and very few data points fall within the Hamming ball with radius 2, resulting in many 0 precision queries. However, the precision values of WeiHash are still consistently higher than other methods.

In the second set of experiments, the precision-recall curves with 16 and 32 hashing bits on both datasets are reported in Fig.2. It is clear that among all of these comparison methods, WeiHash shows the best performance. From the reported figures, we can see that LSH does not

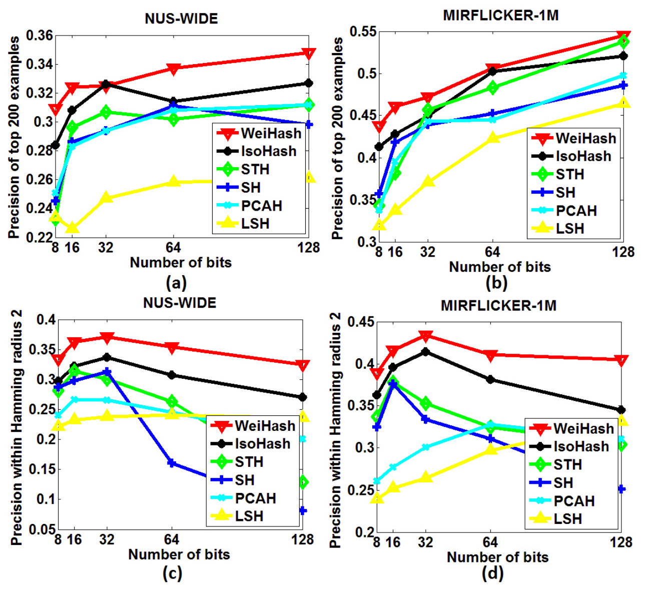

Figure 1: Precision results on two datasets with different hashing bits. (a)-(b): Precision of the top 200 retrieved examples using *Hamming Ranking*. (c)-(d): Precision within Hamming radius 2 using *Hash Lookup*.

perform well in most cases. This is because the LSH method is data-independent and may lead to inefficient codes in practice. For SH and STH, although these methods try to preserve the similarity between data examples in their learned hashing codes, they treat each bit equally and preserve the example similarity represented by standard Hamming distance while our WeiHash approach assigns different weights to different hashing bits based on their variance to capture the importance of each bit and the weighted Hamming distance is utilized for similarity preservation. IsoHash achieves better performance than SH and STH since it somehow tries to balance the variance along different projection dimensions. However, hashing bits are still treated equally whereas in WeiHash, the optimal hashing codes and weights are jointly learned to achieve better hashing performance.

4. CONCLUSION

This paper proposes a novel Weighted Hashing method to assign different weights to different hashing bits. The hashing codes and their corresponding weights are jointly learned in a unified framework by simultaneously preserving the similarities between data examples and balancing the variance of each hashing bit. Two large scale image datasets have been used to demonstrate the advantage of the proposed research over several state-of-the-art hashing methods.

5. ACKNOWLEDGMENTS

This work is partially supported by NSF research grants IIS-0746830, CNS-1012208 and IIS-1017837. This work is also partially supported by the Center for Science of Information (CSoI), an NSF Science and Technology Center, under grant agreement CCF-0939370.

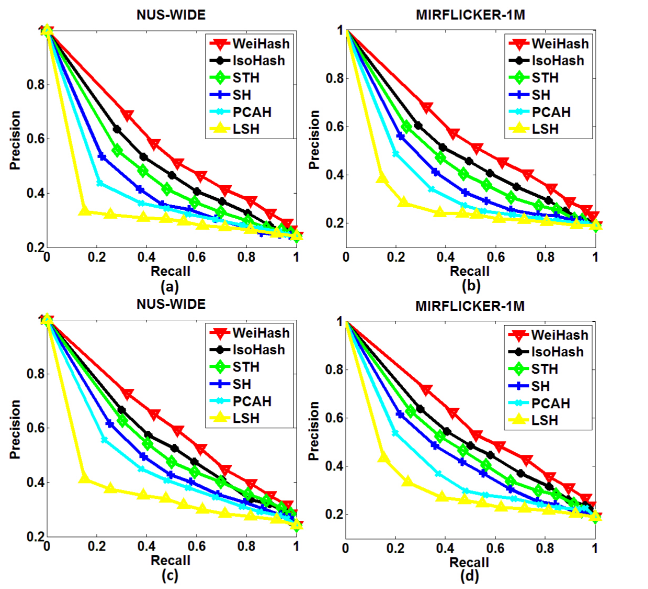

Figure 2: Precision-Recall behavior on two datasets. (a)-(b): Precision-Recall curve with 16 hashing bits. (c)-(d): Precision-Recall curve with 32 hashing bits.

6. REFERENCES

[1] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, 2004.
[2] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, pages 817–824, 2011.
[3] W. Kong and W.-J. Li. Isotropic hashing. In *NIPS*, pages 1655–1663. 2012.
[4] R.-S. Lin, D. A. Ross, and J. Yagnik. Spec hashing: Similarity preserving algorithm for entropy-based coding. In *CVPR*, pages 848–854, 2010.
[5] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *Int. J. Approx. Reasoning*, 50(7):969–978, 2009.
[6] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for large-scale search. *IEEE Trans. Pattern Anal. Mach. Intell.*, 34(12):2393–2406, 2012.
[7] Q. Wang, L. Si, and D. Zhang. A discriminative data-dependent mixture-model approach for multiple instance learning in image classification. In *ECCV (4)*, pages 660–673, 2012.
[8] Q. Wang, L. Tao, and H. Di. A globally optimal approach for 3d elastic motion estimation from stereo sequences. In *ECCV (4)*, pages 525–538, 2010.
[9] Q. Wang, D. Zhang, and L. Si. Semantic hashing using tags and topic modeling. In *SIGIR*, pages 213–222, 2013.
[10] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.
[11] L. Zelnik-Manor and P. Perona. Self-tuning spectral clustering. In *NIPS*, 2004.
[12] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *SIGIR*, pages 18–25, 2010.