Smart Hashing Update for Fast Response

Qiang Yang, Longkai Huang, Wei-Shi Zheng*, and Yingbiao Ling
School of Information Science and Technology, Sun Yat-sen University, China
Guangdong Province Key Laboratory of Computational Science
mmmyqmmm@gmail.com,hlongkai@gmail.com,
wszheng@ieee.org(corresponding author),issLYB@mail.sysu.edu.cn

Abstract

Recent years have witnessed the growing popularity of hash function learning for large-scale data search. Although most existing hashing-based methods have achieved promising performance, they are regarded as passive hashing and assume that the labelled pairs are provided in advance. In this paper, we consider updating a hashing model upon gradually increased labelled data in a fast response to users, called smart hashing update (SHU). In order to get a fast response to users, SHU aims to select a small set of hash functions to relearn and only updates the corresponding hash bits of all data points. More specifically, we put forward two selection methods for performing efficient and effective update. In order to reduce the response time for acquiring a stable hashing code, we also propose an accelerated method to further reduce interactions between users and the computer. We evaluate our proposals on two benchmark data sets. Our experimental results show it is not necessary to update all hash bits in order to adapt the model to new input data, and our model obtains better or similar performance without sacrificing much accuracy against the batch mode update.

1 Introduction

Recently, owing to the dramatically increasing in the scale and dimension of real-world data, hashing has become an important method for expediting similarity computation and search, which has been applied to large-scale vision problems including objection recognition [Torralba *et al.*, 2008a], image retrieval [Xu *et al.*, 2011], local descriptor compression [Strecha *et al.*, 2012], image matching [Strecha *et al.*, 2012], etc. In many conditions, hashing using only dozens of bits per image allows search into a collection of millions of images in a constant time [Torralba *et al.*, 2008b; Wang *et al.*, 2012].

So far, hashing techniques are categorized into two groups: data independent and data dependent methods. Most early exploration of hashing works uses random projections to construct randomized hash functions. One of the most well-known representatives is Locality-Sensitive Hashing (LSH) [Gionis *et al.*, 1999]. Its variants have been widely developed to accommodate more metrics including ℓ_p distance for $p \in (0, 2]$ [Datar *et al.*, 2004], Maharanees distance [Kulis *et al.*, 2009], and kernel similarity [Liu *et al.*, 2012]. LSH-based methods generally require long codes to achieve good precision [Dasgupta *et al.*, 2011; Gorisse *et al.*, 2012]. However, long codes result in low recall rate since the collision probability that two codes fall into the same hash bucket decreases exponentially as the code length grows.

In contrast to the data independent hashing methods, data dependent hashing has recently gotten attraction. Data dependent hashing learns a compact set of hash bits for high-dimensional data points, so that nearest neighbour search can be accomplished in a constant computational complexity as long as the neighbourhood of a point is well preserved in the coding space. In addition, compact codes are especially beneficial for saving storage particularly when the database is very large. To design efficient and effective compact hashing codes, a great number of methods have been developed such as Spectral Hashing (SH) [Weiss *et al.*, 2008], Anchor Graph Hashing (AGH) [Liu *et al.*, 2011], Binary Reconstruction Embedding (BRE) [Kulis and Darrell, 2009], etc. Usually, data dependent hashing methods are categorised into three categories: unsupervised (e.g., [Weiss *et al.*, 2008]), semi-supervised(e.g., [Wang *et al.*, 2012]), and supervised methods (e.g., [Mu *et al.*, 2010; Liu *et al.*, 2012]).

The above works are considered as passive hashing learning in [Zhen and Yeung, 2012], which assumes labelled point pairs are provided in advance. Since they conduct learning once and for all, they cannot efficiently adapt to the change of training data in order to tackle large scalability (e.g. large scale image search problems). This is important for some applications whenever (online) interaction between users and system is needed in order to make system optimise to specific users' requirement. In this paper, we consider proposing smart ways to update an existing hashing learning model in order to obtain a fast response to users' feedback (i.e. the new labelled data by users). More specifically, we assume a set of hash functions are given, which are generated or learned from an existing training set consisting of a small set of labelled data. Our objective is to select the most useful hash functions to update so as to make the whole hashing model promptly

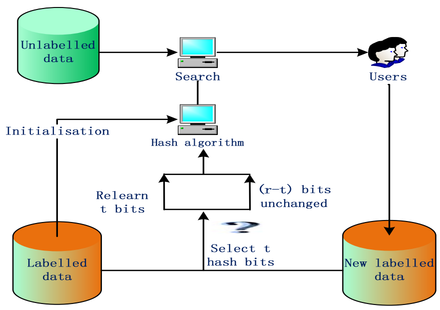


Figure 1: Smart Hashing update for Fast Response.

adapt to the new labelled data. In this work, we have introduced two strategies to select those hash functions.

Our work is closely connected to active learning, which aims to automatically/smartly select points for users to label and update learning system based on existing and the new labelled data. Active hashing (AH) learning [Zhen and Yeung, 2012] is a recent active learning method for hashing. Comparing to active learning, we assume those selected points have already been labelled by users and we focus on the latter procedure for updating the learning system. Rather than conducting a batch mode update as in active learning (e.g. AH), we select a small set of hash functions to update. Although the selection idea is used in ours and active learning, they work in different dimensions and can be combined in future for developing a complete active system for getting more scalable for large scale computing.

Our proposal is not restricted to particular hashing learning models and can be applied generally to supervised and semi-supervised hash learning methods. In this work, we have applied proposed strategies to a supervised hashing model—KSH [Liu *et al.*, 2012] and a semi-supervised model—S3PLH [Wang *et al.*, 2012] in Sect. 3. We will empirically show our proposal is efficient, and low computational and space complexity without sacrificing much accuracy.

2 Approach

2.1 Problem Statement

We aim to select a small set of hash functions to update in a smart way in order to obtain a fast response (as shown in Figure 1) to users' new labelled data without sacrificing much accuracy. Suppose before each interaction with users, there are r hash functions $h_k = sgn(w_k x)$ ($1 \leq k \leq r, w_k$ stands for the projection vector for the kth hash function) learned from a small set of l labelled data $\chi_l = \{x_1, \cdots, x_l\}$, which is a part of the big data set $\chi_n = \{x_1, \cdots, x_n\}(n \gg l)$ with the remaining $(n - l)$ data points unlabelled. Then, after each interaction, new labels for p unlabelled points are assigned by users and the labelled set becomes $\chi_{l+p} = \{x_1, \cdots, x_l, x_{l+1}, \cdots, x_{l+p}\}$. Therefore, the labelled data becomes $(l + p)$ points. After each interaction, we would update the notation by $l \leftarrow l + p$.

2.2 Proposed Strategies

Due to large scalability for computation, updating all hash functions as data increases becomes infeasible to gain a fast response to a user (as shown in our experiments). Our main idea is to develop a bit-wise hash update strategy in order to update one bit or a small number of bits of hash codes at a time, and we call it as Smart Hashing Update (SHU). Algorithm 1 gives an overview of our solution. The challenge of this idea is how to select hash bits to update, in other words, how to select a set of hash functions to update.

Algorithm 1: Smart Hashing Update

Input: Training set $\chi = \{x_i \in R^d\}_{i=1}^n$, pairwise label matrix $S \in R^{l \times l}$ on initial l labelled samples $\chi_l = \{x_i\}_{i=1}^l$, hashing method f, smart hashing update selection strategy SHU, maximum interaction T, hash bit length r, and the number t of hash functions to be updated.

Learn r hash functions on χ_l using hashing method f;

for $k = \{1, \ldots, T\}$ **do**

 1)obtain p new labelled data χ_p;
 2)update χ_l and S, and $l \leftarrow l + p$;;
 3)obtain hash codes of all χ_l;
 4)select t hash functions according to SHU;
 5)exclude the effect of the other $(r - t)$ hash functions on S and χ_l;
 6)relearn the selected t hash functions on updated S and χ_l;

end

Output: r hash functions $\{h_k(x) = sgn(x^T w_k)\}_{k=1}^r$ and hash codes for all data $H = \{code_r(x_i)\}_{i=1}^n$

In the following, from different perspectives, we mainly propose two strategies for selecting hash bits to update. Before detailing our proposals, we describe random selection principle first, which is a natural way for selection.

Random Selection

Random strategy selects $t(1 \leq t \leq r)$ hash functions randomly from r hash functions. Then based on the updated data and similarity in each iteration, the t hash functions are retrained or relearned. Although it is the simplest one, we show in the experiment that such a random strategy is not effective. More important, it is unstable, which means that sometimes it may improve the performance and sometimes it could also degrade the performance.

Consistency-based Selection

Hashing methods are to map similar data points onto the same hash codes and make them fall into the same bucket. Relying on this perspective, we propose a selection strategy that investigates the consistency of hashing codes of data from the same class. We call it as consistency-based selection strategy.

Therefore, the core idea here is to select the t hash functions which lead to in-consistent hashing coding within a class. To be detail, let us suppose that there are c classes in total, the hash bit length is r, and the number of data points is n. In addition, the label information of labelled data is stored in matrix L. Let $h_j(x_i)$ indicate the jth hash code of point x_i.

Since each hash bit has only two values, namely $\{1, -1\}$[1], we can just count the number of -1s and 1s for a specific class at each hash bit. Let $num(k, j, -1)$ and $num(k, j, 1)$ stand for the number of -1s and 1s of kth class at jth hash bit respectively:

$$\begin{array}{l} num(k,j,-1) = \sum_{i=1}^{l} \{h_j(x_i) = -1 \&\& L(x_i) = k\} \\ num(k,j,1) = \sum_{i=1}^{l} \{h_j(x_i) = 1 \&\& L(x_i) = k\} \end{array} \tag{1}$$

where l is the number of labelled points. Then, we define $Diff(k, j)$ which can indicate the number of data points of class k whose jth hash bit differs from the majority of the class as follows:

$$Diff(k,j) = \min\{num(k,j,-1), num(k,j,1)\} \tag{2}$$

The consistency-based selection is thus to select t hash functions according to the expectation of $Diff(k, j)$ over all classes. In this paper, we estimate the expectation $Diff\_mean$ below

$$Diff\_mean(j) = mean(Diff(:,j)) = \frac{1}{c}\sum_{k=1}^{c} Diff(k,j) \tag{3}$$

where $Diff\_mean(j)$ is the mean value of the jth hash bit in $Diff(k, j)$ across all classes.

As we wish that all similar data points should be mapped to the same hash code for each bit, and thus we aim to minimize $Diff\_mean$ as follows

$$arg \min_{h \in H} Diff\_mean \tag{4}$$

Then, this would suggest that we should update hash bits which correspond to the largest $Diff\_mean(j)$ and relearn the corresponding hash functions. Algorithm 2 shows the procedure of this strategy.

Algorithm 2: Consistency-based Selection Algorithm

Input: Hash codes H of l labelled data, a label matrix L defined on the l data, the number of classes c, hash bit length r, and the number t of hash bits to be updated.
for $k = \{1, \ldots, c\}$ **do**
  for $j = \{1, \ldots, r\}$ **do**
    Compute $num(k, j, -1)$ and $num(k, j, 1)$;
    if $num(k, j, -1) < num(k, j, 1)$ **then**
      $Diff(k, j) = num(k, j, -1)$;
    else
      $Diff(k, j) = num(k, j, 1)$;
    end
  end
end
$Diff\_mean(j) = \frac{1}{c}\sum_{k=1}^{c} Diff(k, j)$;
sort $Diff\_mean$;
obtain the indexes of t larger elements;
Output: t indexes of hash functions

[1]another hash codes are $\{0, 1\}$, but 0 and 1 hash codes can be transferred into -1 and 1 codes through $2h - 1$

Similarity-based Selection

In the previous section, we introduce the use of consistency of hashing codes within a class for selecting hash functions to update. It, however, does not explicitly measure the similarity relations between either intra-class or inter-class data. Hence for optimising those relations, we propose another selection strategy called similarity-based selection.

We first define the similarity S_{ij} between data below:

$$S_{ij} = \begin{cases} 1, & x_i \text{ and } x_j \text{ are similar} \\ -1, & x_i \text{ and } x_j \text{ are not similar} \end{cases} \tag{5}$$

where two points from the same class are similar, the ones from different classes are not similar, S is a similarity matrix consisting of entries S_{ij}.

It is expected the learned hash codes can preserve the similarity among data points. One can evaluate the effect of learned hash codes using the following objective function as used in KSH [Liu *et al.*, 2012]:

$$\min_{H_l \in \{-1,1\}^{l \times r}} Q = \left\| \frac{1}{r} H_l H_l^T - S \right\|_F^2 . \tag{6}$$

where H_l is a hash code matrix where each row stands for the hash code of a labelled data.

Our idea to select the hash functions for update is to select those who contribute less for preserving the similarity. In order to do so, we define a residue similarity matrix when the kth hash bit is not used as follows

$$R_k = \left\| rS - H_{r-1}^k {H_{r-1}^k}^T \right\|_F^2 \tag{7}$$

where H_{r-1}^k is the hash code matrix of labelled data χ_l when excluded the kth hash bit (i.e. the kth column of H_l is deleted) and $\|.\|_F$ is the Frobenius norm.

R_k suggests how much similarity is preserved when all hash bits are used except the kth one. The larger R_k is, the more contribution the kth hash bit makes to reconstructing S, which in other words has a great effect on the approximation. Thus, the kth hash bit corresponding to the largest R_k should remain unchanged, namely, we should choose the hash bit corresponding to the smallest R_k for update. Hence, our objective function for choosing hash bits to update comes as follows

$$\min_{k \in \{1,2,\ldots,r\}} R_k = \left\| rS - H_{r-1}^k {H_{r-1}^k}^T \right\|_F^2 \tag{8}$$

We also detail this strategy in Algorithm 3.

Unlike the random selection strategy, similarity-based selection method prefers to select the bits to update which contribute less to approximating S. Therefore, we can see that, every time it minimises the distance between hash codes inner products $H_l H_l^T$ and rS. Thus, this method is more stable and reliable than random method. Additionally, compared with consistency-based strategy, similarity-based strategy updates the hash functions in order to make them more effective for preserving the similarity between data.

Algorithm 3: Similarity-based Selection Algorithm

Input: Hash code matrix H of l labelled data, a pairwise similarity matrix S defined on the l data, hash bit length r, and the number t of hash bits to be updated.

for $j = \{1, \ldots, r\}$ **do**

Get the hash code matrix H_{r-1}^{k} when excluding the jth bit;

$R(j) = \left\| rS - H_{r-1}^{k} {H_{r-1}^{k}}^{T} \right\|_F^2$;

end

obtain the indexes corresponding to t smallest $R(j)$;

Output: t indexes of hash functions

2.3 Accelerated Hashing Update

Since at each update iteration of the above strategies we only utilise the labelled data provided by users and the labelled data points could still very few, this may limit the speed of the algorithms to get a stable performance as shown in our experiments, resulting in more interactions between users and the system.

In order to accelerate the interaction, we explore the usefulness of the unlabelled data around the labelled data provided by users at each iteration. More specifically, we developed K-nearest neighbours (KNN)[Coomans and Massart, 1982] based on hashing update strategy as follows. For each selected point x_i labelled by users, we use Hamming distance among the hash codes learned in last iteration to obtain the K nearest neighbours for x_i. In this work, we simply set K to 1, namely the nearest neighbour, as the nearest neighbour obtained by Hamming distance is more likely to share the same label as x_i. Then for x_i, we assume its nearest neighbour is similar to x_i and thus we can compute the similarity between x_i and its nearest neighbour using Eq. (5). In some aspect, this can be viewed as deriving some pseudo labels for some unlabelled data, resulting in a better learned model and thus reducing interactions as shown in Sect.3. In this work, we particularly develop an accelerated version for the similarity-based selection strategy.

2.4 Update for the Selected Hash Functions

After selecting the t hash bits to update, we now detail how to update them based on existing supervised or semi-supervised hash models. In this paper, we particularly combine the proposed strategy (SHU) with a supervised hashing model—KSH [Liu *et al.*, 2012] and a semi-supervised one—S3PLH [Wang *et al.*, 2012].

We first retain the residual unselected $(r-t)$ hash functions at the current round. Then, for the selected t hash functions, we just describe hash update at one round. Since the unselected $(r-t)$ bits remain unchanged, we just exclude the effects that the residual $(r-t)$ hash bits have on the similarity S upon all labelled data after users' action as follows:

$$S_t = rS - H_{r-t}H_{r-t}^{T} \quad (9)$$

where H_{r-t} stands for the hash code matrix of currently available labelled data excluding the t selected hash bits and S_t represents the similarity that the selected t hash bits should preserve. Then, we relearn the t hash functions simply through substituting the similarity matrix S in KSH and S3PLH with our updated similarity matrix S_t.

3 Experiments

3.1 Datasets

We evaluate our methods on two benchmark data sets: MNIST[2], and 20 Newsgroups[3]. On all data sets, we use the ground-truth class labels to simulate users' label information. MNIST consists of $70K$ handwritten digit samples from digit 0 to 9, each of which is an image of size 28×28 pixels yielding a 784-dimensional vector. In experiment, the entire dataset is randomly partitioned into two parts: a training data set consisting of $69K$ samples and a testing set consisting of $1K$ samples. Newsgroups is a text dataset, which contains $18,846$ documents distributed across 20 categories. We randomly selected $1K$ documents as the testing set and the rest are used as the training set. In order to reduce the computational complexity, we first reduce the dimensionality of data from 26,214 dimension to 1000 dimension by using PCA.

3.2 Protocols and Methods

Protocols

We evaluate the algorithms in terms of computational time and accuracy. Therefore, we follow two protocols, namely mean precision on all test data for evaluating accuracy and average response time (namely the training time for each update and the retrieval time, where the latter is the same for all methods) for comparing computational time. For all datasets, to perform real time search, we adopt Hamming ranking and select the top 500 as nearest neighbours for evaluation, which is commonly followed in [Liu *et al.*, 2012; Xu *et al.*, 2011]. In order to guarantee the stability, we run each algorithm 10 times with random division of training and testing sets. During all experiments, we fix the length of hash bits to be 24. We run all methods on the windows server with eight 3.4GHz Intel Core CPUs and 48GB memory.

Methods

We develop a set of smart hashing update (SHU) algorithms by applying our strategies to a supervised algorithm—KSH [Liu *et al.*, 2012] and a semi-supervised algorithm—S3PLH [Wang *et al.*, 2012]. We detail the notations of these algorithms as follows.

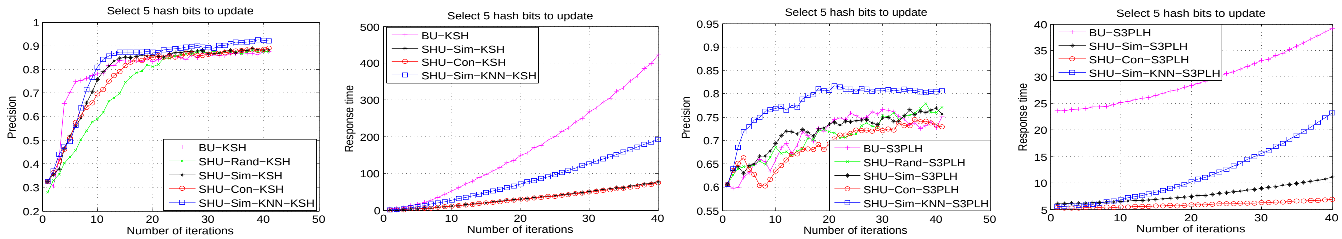
1) KSH based SHU. KSH based smart hashing update (SHU) methods are based on the supervised algorithm—KSH[4] and only select a small subset of hash functions to update. Based on different selection strategies, we develop SHU-Rand-KSH, SHU-Con-KSH, and SHU-Sim-KSH, which are based on random selection, consistency-based selection strategy and similarity-based strategy, respectively. SHU-Sim-KNN-KSH is our accelerated method based on similarity-based selection. Unless otherwise stated, we select 5 hash functions to update in SHU-Rand-KSH, SHU-Con-KSH, SHU-Sim-KSH and SHU-Sim-KNN-KSH. In order to compare these methods, we also compare the batch update approach directly using KSH itself, denoted as BU-KSH.

2) S3PLH based SHU. S3PLH based smart hashing update (SHU) methods are based on the semi-supervised

[2]http://yann.lecun.com/exdb/mnist/

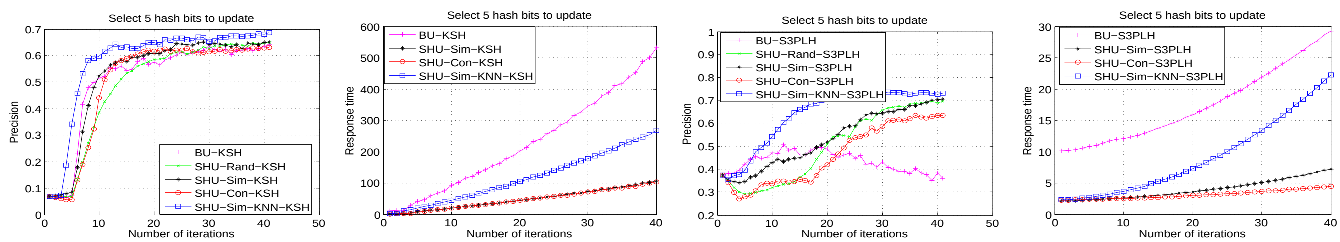
[3]http://qwone.com/∼jason/20Newsgroups/

[4]http://www.ee.columbia.edu/∼wliu/



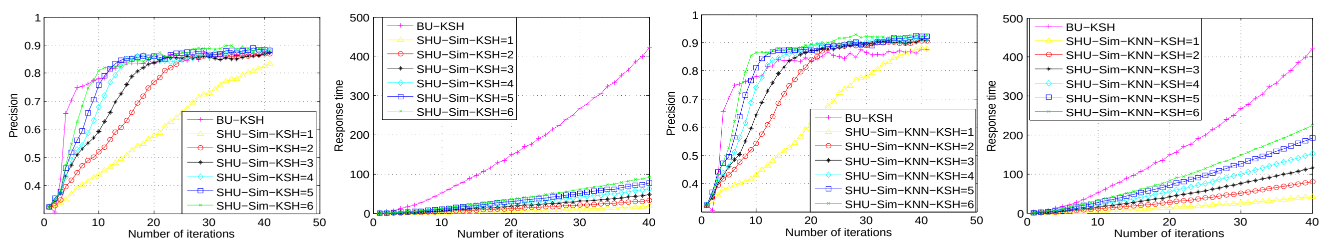
(a) Precision of KSH based SHU (b) Response time of KSH based SHU (c) Precision of S3PLH based SHU (d) Response time of S3PLH based SHU

Figure 2: Precision (@top 500) and Response time comparison on MNIST, when 5 hash bits are selected to update. 2(a) and 2(b) show the performance of KSH based SHU, and 2(c) and 2(d) display the performance of S3PLH based SHU.



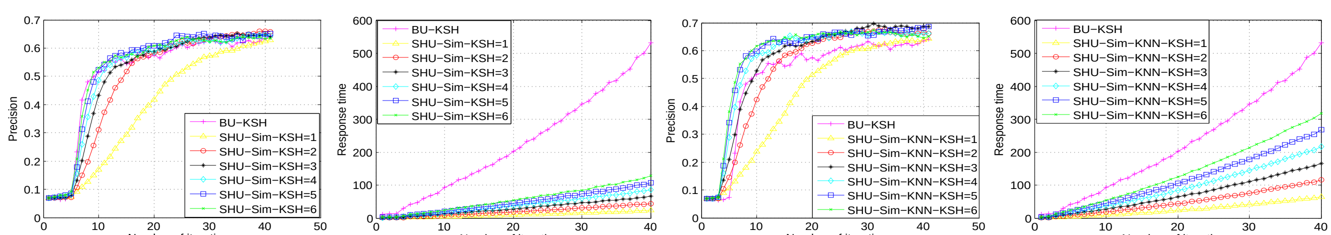
(a) Precision of KSH based SHU (b) Response time of KSH based SHU (c) Precision of S3PLH based SHU (d) Response time of S3PLH based SHU

Figure 3: Precision (@top 500) and Response time comparison on NewsGroups, when choosing 5 hash bits to update. 3(a) and 3(b) show the performance of KSH based SHU, and 3(c) and 3(d) display the performance of S3PLH based SHU.



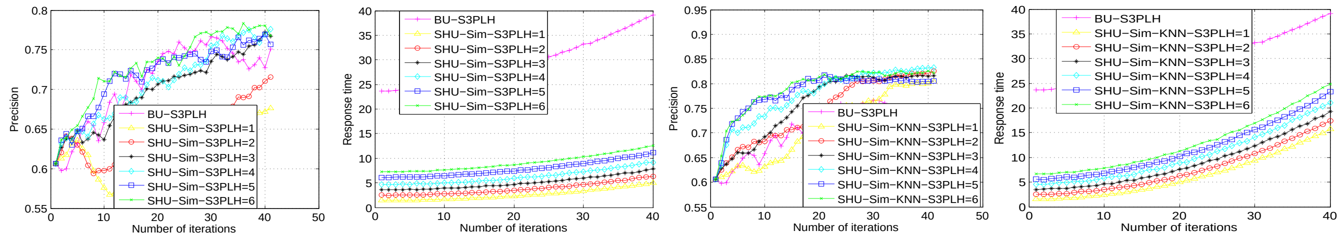
(a) Precision of SHU-Sim-KSH (b) Response time of SHU-Sim-KSH (c) Precision of SHU-Sim-KNN-KSH (d) Response time of SHU-Sim-KNN-KSH

Figure 4: Precision (@top 500) and Response time comparison of SHU-Sim-KSH and SHU-Sim-KNN-KSH with different number of hash bits to update on MNIST.



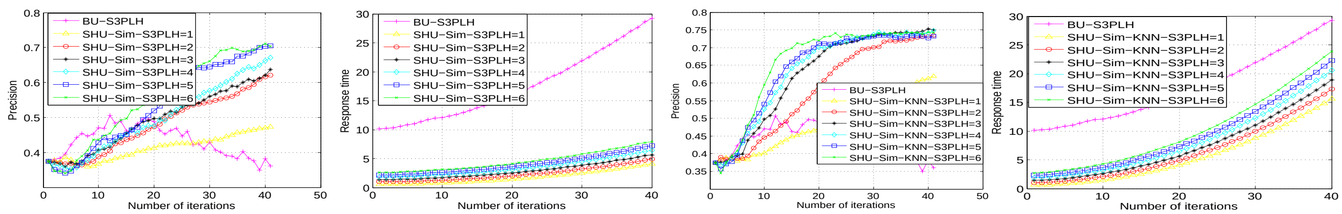
(a) Precision of SHU-Sim-KSH (b) Response time of SHU-Sim-KSH (c) Precision of SHU-Sim-KNN-KSH (d) Response time of SHU-Sim-KNN-KSH

Figure 5: Precision (@top 500) and Response time comparison of SHU-Sim-KSH and SHU-Sim-KNN-KSH with different number of hash bits to update on NewsGroups.



(a) Precision of SHU-Sim-S3PLH (b) Response time of SHU-Sim-S3PLH (c) Precision of SHU-Sim-KNN-S3PLH (d) Response time of SHU-Sim-KNN-S3PLH

Figure 6: Precision (@top 500) and Response time comparison of SHU-Sim-S3PLH and SHU-Sim-KNN-S3PLH with different number of hash bits to update on MNIST.



(a) Precision of SHU-Sim-S3PLH (b) Response time of SHU-Sim-S3PLH (c) Precision of SHU-Sim-KNN-S3PLH (d) Response time of SHU-Sim-KNN-S3PLH

Figure 7: Precision (@top 500) and Response time comparison of SHU-Sim-S3PLH and SHU-Sim-KNN-S3PLH with different number of hash bits to update on NewsGroups.

algorithm—S3PLH[5] and only select a small subset of hash functions to update. Similar to KSH based SHU, based on different selection strategies, we develop SHU-Rand-S3PLH, SHU-Con-S3PLH, and SHU-Sim-S3PLH, which are based on random selection, consistency-based selection strategy and similarity-based strategy, respectively. SHU-Sim-KNN-S3PLH is our accelerated method based on similarity-based selection. Likewise, in order to compare these methods, we also compare the batch update approach directly using S3PLH itself, denoted as BU-S3PLH.

3.3 Results on Datasets

We compare all related methods described above on two datasets: MNIST and NewsGroups. Figures 2(a), 2(c), 3(a) and 3(c) show the accuracy performance of different algorithms based on supervised KSH and semi-supervised S3PLH hashing algorithms respectively when selecting 5 hash functions to update. Figures 2(b), 2(d), 3(b) and 3(d) provide the response time for different techniques. From the results, we can draw the following:

1. Our proposed SHU-Con-KSH, SHU-Sim-KSH, SHU-Con-S3PLH and SHU-Sim-S3PLH can greatly approximate the performance of batch based methods: BU-KSH and BU-S3PLH respectively. Note that all our methods take much less time than the batch based ones. For SHU-Sim and SHU-Con based methods, they only take $1/5$ time or so of baseline methods. All these four methods perform better than the random strategy and this shows that random strategy is not an optimal way and sometimes it is not stable as shown in Figure 3(c).

2. Regarding the accelerated methods—SHU-Sim-KNN based methods, namely SHU-Sim-KNN-KSH and SHU-Sim-KNN-S3PLH, they get two advantages: (1) compared with the SHU-Con and SHU-Sim based methods, it largely reduces the number of interactions and gets very similar accuracy or even better as shown in Figures 2(c) and 3(c), although a little more time is taken for computation for each iteration; (2) compared with batch based methods, it is experimentally confident that it always gets better performance after a few interactions, and thus it can take much less time if it stops earlier. This is more obvious on the Newsgroups dataset. Our accelerated algorithm indeed outperforms all the other methods, especially much better than baseline methods, obtaining 5% and 10% higher precision than BU-KSH and BU-S3PLH respectively on MNIST dataset. On NewsGroups dataset, the performance is much better and gains nearly 5% and 20% higher than BU-KSH and BU-S3PLH respectively. This suggests the unlabelled data around the labelled data provided by users are informative to help obtain improvement. Although the accelerated methods take more time after a certain amount of interactions as shown in Figures 2(b), 2(d), 3(b) and 3(d), they don't need to take more than 10 interactions to gain better and more stable performance than batch based methods. This means that our accelerated methods can stop earlier and thus spend much less as a result, which is significantly useful for reducing the interaction between systems and users.

3. Combining Figure 2 and 3, we can apparently see that similarity-based selection strategy outperforms slightly better than consistency-based selection. The reason is that similarity-based approach does the selection for

[5] http://www.cs.ust.hk/∼dyyeung/paper/publist.html#journal

optimising the relations between data, which consider distinguishing intra-class and inter-class hash codes and cannot be processed by the consistency-based one, albeit both are supervised approaches.

In order to see the effect of the parameter t, namely the number of hash functions to update, we also report the performance in this aspect. Due to limitation of the length of the paper, we mainly present the performance of two algorithms: SHU-Sim based and SHU-Sim-KNN based methods in Figures 4 $\sim$ 7. We first introduce the denotation here. For example SHU-Sim-KNN-KSH=3 means three hash bits (t=3) are selected to update in each interaction for method SHU-Sim-KNN-KSH. Figures 4 and 5 show the results of our algorithms based on a supervised algorithm on MNIST and NewsGroups, respectively. We can see that the more hash functions used to update, the better the performance is, but when it is more than 3, the performance tends to be stable and converge in most cases, and little improvement is gained as the number of hash functions to update increases.

In addition, as shown in Figures 4(b), 5(b), 6(b) and 7(b), when the number of selected hash bits increases, the response time only increases a little for SHU-Sim based methods; while for our accelerated methods, the response time increases a little severely as long as the number of selected hash bits to update t increases. However, from Figures 4(a), 4(c), 5(a), and 5(c), the number of interactions that our algorithms need to obtain stable performance decreases when t increases. Hence, there would be tradeoff between the number of selected hash bits to update and the response time.

4 Conclusion and Future Work

We have proposed smart strategies for updating hash functions in order to get a fast response to users' feedback through labelling new data. Our proposed strategies are not limited to certain hashing methods and can be applied to supervised and semi-supervised hashing methods. Particularly, we have applied them to two state-of-the art models and have developed KSH based and S3PLH based smart hashing update methods. Our experiments demonstrate that our proposals achieve better performance in both accuracy and computational time.

Obviously, based on our proposed algorithm, much extra work can be done, for instance, figuring out the optimal number of hash bits to update. And we will do these in future.

Acknowledgements

This research was supported by the National Natural Science of Foundation of China (No.61102111), the NSFC-GuangDong (U1135001), Foundation of China and Royal Society of Edinburgh (NSFCRSE) joint project (No.61211130123), Specialized Research Fund for the Doctoral Program of Higher Education (No.20110171120051), Guangdong Natural Science Foundation (No.S2012010009926), the Fundamental Research Funds for the Central Universities (No.12lgpy28, 2012350003161455) and the Guangdong Provincial Government of China through the Computational Science Innovative Research Team program.

References

[Coomans and Massart, 1982] D. Coomans and D.L. Massart. Alternative k-nearest neighbour rules in supervised pattern recognition. *Analytica Chimica Acta*, 136(0):15 – 27, 1982.

[Dasgupta *et al.*, 2011] A. Dasgupta, R. Kumar, and T. Sarlos. Fast locality-sensitive hashing. In *ACM SIGKDD*, pages 1073–1081, 2011.

[Datar *et al.*, 2004] N. Datar, M.and Immorlica, P. Indyk, and V.S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, pages 253–262, 2004.

[Gionis *et al.*, 1999] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *VLDB*, pages 518–529, 1999.

[Gorisse *et al.*, 2012] D. Gorisse, M. Cord, and F. Precioso. Locality-sensitive hashing for chi2 distance. *TPAMI*, 34(2):402 –409, feb. 2012.

[Kulis and Darrell, 2009] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050, 2009.

[Kulis *et al.*, 2009] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *PAMI*, 31(12):2143 –2157, dec. 2009.

[Liu *et al.*, 2011] W. Liu, J. Wang, S. Kumar, and S.F. Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011.

[Liu *et al.*, 2012] W. Liu, J. Wang, R.G. Ji, Y.G. Jiang, and S.F. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074 –2081, june 2012.

[Mu *et al.*, 2010] Yadong Mu, Jialie Shen, and Shuicheng Yan. Weakly-supervised hashing in kernel space. In *CVPR*, pages 3344 –3351, june 2010.

[Strecha *et al.*, 2012] C. Strecha, A.M. Bronstein, M.M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *PAMI*, 34(1):66 –78, jan. 2012.

[Torralba *et al.*, 2008a] A. Torralba, R. Fergus, and W.T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *PAMI*, 30(11):1958 –1970, nov. 2008.

[Torralba *et al.*, 2008b] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, pages 1 –8, june 2008.

[Wang *et al.*, 2012] Jun Wang, S. Kumar, and Shih-Fu Chang. Semi-supervised hashing for large-scale search. *PAMI*, 34(12):2393 –2406, dec. 2012.

[Weiss *et al.*, 2008] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[Xu *et al.*, 2011] H. Xu, J.D. Wang, Z. Li, G. Zeng, S.P. Li, and N.G.Yu. Complementary hashing for approximate nearest neighbor search. In *ICCV*, pages 1631 –1638, nov. 2011.

[Zhen and Yeung, 2012] Yi Zhen and Dit-Yan Yeung. Active hashing and its application to image and text retrieval. *Data Mining and Knowledge Discovery*, pages 1–20, 2012.