SPEC Hashing: Similarity Preserving algorithm for Entropy-based Coding

Ruei-Sung Lin David A. Ross Jay Yagnik
Google Inc.
Mountain View, CA 94043
{rslin, dross, jyagnik}@google.com

Abstract

Searching approximate nearest neighbors in large scale high dimensional data set has been a challenging problem. This paper presents a novel and fast algorithm for learning binary hash functions for fast nearest neighbor retrieval. The nearest neighbors are defined according to the semantic similarity between the objects. Our method uses the information of these semantic similarities and learns a hash function with binary code such that only objects with high similarity have small Hamming distance. The hash function is incrementally trained one bit at a time, and as bits are added to the hash code Hamming distances between dissimilar objects increase. We further link our method to the idea of maximizing conditional entropy among pair of bits and derive an extremely efficient linear time hash learning algorithm. Experiments on similar image retrieval and celebrity face recognition show that our method produces apparent improvement in performance over some state-of-the-art methods.

1. Introduction

With the advance of Internet, we are inundated with an abundance of data of images, documents, music, videos, etc. As the size of the data continues to grow, the density of similar objects in the data space also increases. These objects are likely to have similar semantics. As a result, inferences based on nearest neighbors can be more reliable than ever before.

In this paper, we describe a new learning-based hashing algorithm for nearest neighbor search in high dimensional feature space. Our nearest neighbors are objects with similar semantics. The trained hash function map objects to binary vectors such that the neighboring objects have small Hamming distances between their codes, while irrelevant objects have large distances. Therefore, we can use these binary vectors for fast semantic nearest-neighbor retrieval. Learning our hash function takes time linear to the data size and is fast. This makes our algorithm feasible to tasks with an evolving dataset, in which periodically updating or retraining the hash function is required.

Searching nearest neighbors in sublinear time has been an ongoing research. Traditional methods such as the KD-tree [1] works well on data with limited feature dimensionality, but become linear time search as dimensionality grows. Recently, Locality Sensitive Hashing (LSH) [2, 3] has been successfully applied to datasets with high dimensional features. It uses random projections to map objects from feature space to bits, and treats these bits as keys for multiple hash tables. As a result, the collision of similar samples in at least one hash bucket has high probability. This randomized algorithm has tight asymptotic bound, and provides the foundation to a number of follow-up works.

Parameter sensitive hashing [10] is one such extension. It chooses a set of weak binary classifiers to generate bits for the hash keys. The classifiers are selected according to the criteria that nearby objects are more likely to have the same class label than more distant objects. Ke et. al. [4] adopt a similar idea, and formulate the learning problem within the boosting framework. A major drawback of this type of approach is the requirement of evaluation on object pairs, which has size quadratic to the number of objects. Hence, its scalability to larger scale dataset is limited.

Salakhutdinov et. al. [9] use restricted Boltzmann machines (RBM) to learn the hash function, and show that the learned hash codes preserve semantic similarity in Hamming space. This approach is then applied to the task of similarity search in millions of images [11]. Training RBM is a computationally intensive process that makes it very costly to re-train the hash function when data evolve. Spectral Hashing [14] takes a completely different approach to generate hash code. It first rotates the feature space to statistically orthogonal axes using PCA. Then, a special basis function is applied to carve each axis independently to generate hash bits. As a result, bits in the hash code are all independent, which leads to a compact representation with short code length. Experiments in [14] show it outperforms RBM and the boosting approach. Spectral Hashing is developed on the assumption that objects are spread in an Euclidean

space with a particular distribution; either uniform or Gaussian. This is seldom true in real world data set. Raginsky et. al. [7] introduce a distribution free coding method. It maps features to a low dimensional space using random Fourier features [8] and then randomly quantizes these projected features into a binary code. This simple encoding scheme has good convergence properties and is shown to outperfrom spectral hashing empirically.

In this paper we present a new hashing algorithm. It uses the affinity matrix of the training set as the target, and learns a hash function such that the Hamming distances correlate to the similarities specified in the affinity matrix. Our hash function is a collection of bit functions, and learning is a incremental process that incrementally selects bit functions to expand the hash code. We show that our algorithm adds bits to the hash function to increase the Hamming distances between dissimilar objects while keeping the Hamming distances between similar objects small. This algorithm has quadratic computational complexity, so we propose two approximate linear time solutions. These algorithms are based on similar heuristics and can be related to maximizing conditional entropy between the new bit function and the existing bits in the hash function, which, therefore, minimizes the mutual information between bits. As a result the learned hash function generates compact hash codes. Our experiments indicate that our algorithm outperforms Spectral Hashing. Our method is related to work on metric learning [3, 16], but differs in that we commit a priori to using Hamming distance, then seek to learn an Hamming embedding that preserves pairwise similairies.

The remainder of the paper is organized as follows. In section 2, we formalize the learning problem, and describe our incremental learning algorithm. The proposed algorithm has quadratic complexity, so in section 3 we present two approximate linear time algorithms and describe their connections to the idea of maximizing conditional entropy. In section 4, we describes the implementation details of our two fast learning algorithms, which followed by the experiments and the conclusion.

2. Similarity-Preserving Hashing

We define our learning problem as follows. Given training set $\{x_i\}_i$ and affinity matrix S of this set, we want to learn a similarity preserving hash function. This hash function maps objects from the feature space to a Hamming space such that only objects with high similarity measures will have small Hamming distances.

We denote B_T as a T-bit hash function. In our model, it is a collection of T binary functions: $B_T(x) = \{b_1(x), b_2(x), \ldots, b_T(x)\}$ with $b_k(x) \in \{0, 1\}$. $d_k(i, j)$ is the distance based on b_k. $d_k(i, j) = 1$, if $b_k(x_i) \neq b_k(x_j)$. Otherwise, $d_k(i, j) = 0$. $H_T(i, j)$ is the Hamming distance between two hash codes generated by B_T. Therefore, $H_T(i, j) = \sum_{k=1}^{T} d_k(i, j)$. In the affinity matrix, S_{ij} is the semantic similarity between object i and j, $S_{ij} \geq 0$. $S_{ij} = 0$ indicates that pair (i, j) are dissimilar. We expect that every object is only related to a small number of objects. Therefore, S is a sparse matrix.

We formulate the hash learning problem as a distribution learning process proposed in [5]. First, S is normalized to $\sum_{i,j} S_{ij} = 1$, and treated as our target distribution. We then define another distribution $W^{(T)}$ using Hamming distance H_T:

$$W_{ij}^{(T)} = \frac{1}{Z_T} e^{-\lambda H_T(i,j)}$$

where $Z_T = \sum_{i,j} e^{-\lambda H_T(i,j)}$.

By making distribution $W^{(T)}$ close to the target distribution, objects with large similarity values will have relatively small hamming distances, and vice versa. As a result, we can learn hash function B_T by minimizing the Kullback-Leibler divergence $KL(S||W^{(T)})$. Because S is fixed, this is equivalent to minimizing the cross entropy:

$$\begin{aligned} \min_{B_T} J_T &= -\sum_{i,j} \lambda S_{ij} \log W_{ij}^{(T)} \\ &= \lambda \sum_{i,j} S_{ij} H_T(i, j) + \log \sum_{k,l} e^{-\lambda H_T(k,l)} \end{aligned}$$

Without loss of generality and to facilitate the description of our method, we set λ to be 1 in the following derivations.

2.1. Incremental Learning

Directly optimizing (1) is a challenging task, especially when B_T has a large hypothesis space. Here we adopt an alternative approach. By factorizing (1) into a recursive equation, we present an sub-optimal algorithm that incrementally learns the hash function one bit at a time.

Denote $sum(H_k) = \sum_{i,j} e^{-H_k(i,j)}$ and $cut_S(b_l) = \sum_{i,j} S_{ij} d_l(i, j) = \sum_{i,j;b_l(i)\neq b_l(j)} S_{ij}$. This choice of namings will be explained further in the following section. We can rewrite J_T as:

$$\begin{aligned} J_T &= \sum_{i,j} S_{ij} \sum_{t=1}^{T} d_t(i, j) + \log sum(H_T) \\ &= \sum_{t=1}^{T} cut_S(b_t) + \log sum(H_T) \\ &= J_{T-1} + cut_S(b_T) + \log sum(H_T) - \log sum(H_{T-1}) \end{aligned} \tag{1}$$

Let $cut(H_k, b_l) = \sum_{i,j;b_l(x_i)\neq b_l(x_j)} e^{-H_k(i,j)} = \sum_{i,j;d_l(i,j)=1} e^{-H_k(i,j)}$. We can derive:

$$\sum_{\substack{i,j \\ d_l(i,j)=0}} e^{-H_k(i,j)} = sum(H_k) - cut(H_k, b_l)$$

Using this property and $H_T(i,j) = H_{T-1}(i,j) + d_T(i,j)$, $sum(H_T)$ can be factorized as:

$$sum(H_T) = \sum_{\substack{i,j \\ d_T(i,j)=0}} e^{-H_{T-1}(i,j)} + \sum_{\substack{i,j \\ d_T(i,j)=1}} e^{-(H_{T-1}(i,j)+1)} = sum(H_{T-1}) - cut(H_{T-1}, b_T) + e^{-1} cut(H_{T-1}, b_T) \quad (2)$$

Putting (1) and (2) together, we get:

$$L_T = J_T - J_{T-1} = cut_S(b_T) + \log\left(1 - (1 - e^{-1})\frac{cut(H_{T-1}, b_T)}{sum(H_{T-1})}\right)$$

L_T represent the 'improvement' from adding binary function b_T to the hash function B_{T-1}. If L_t is negative, adding b_T is favorable because it further reduces the cross entropy defined in (1).

Based on this result, we can learn the hash function by incrementally selecting new binary function to expand the hash code. Our learning algorithm is formalized as follows:

1. Start with $t = 0$, initialize a empty hash function B_o.
2. Find binary function b_{t+1} that minimizes:

$$\min_{b_{t+1}} L_{t+1} = cut_S(b_{t+1}) + \log\left(1 - (1 - e^{-1})\frac{cut(H_t, b_{t+1})}{sum(H_t)}\right) \quad (3)$$

3. Set $B_{t+1} = \{B_t, b_{t+1}\}$ and increment t by one. Repeat step 2 until either the desired code length is reached or no candidate for b_{t+1} has negative L_{t+1}.

2.2. Analysis

According to (3), binary function b_{t+1} must induce small $cut_S(b_{t+1})$ and large $cut(H_t, b_{t+1})$. This can be reasoned as follows. $cut_S(b_{t+1}) = \sum_{i,j;b_{t+1}(i)\neq b_{t+1}(j)} S_{ij}$ is the total loss of assigning similar objects to different binary code in b_{t+1}. This term is minimized when similar objects are assigned the same binary code. In fact, taken in isolation, $cut_S(b_{t+1})$ can be trivially minimized by assigning all objects the same label, collapsing all Hamming distances to zero.

On the other hand, in order to have large value in $cut(H_t, b_{t+1}) = \sum_{i,j;b_{t+1}(x_i)\neq b_{t+1}(x_j)} e^{-H_t(i,j)}$, b_{t+1} should assign different codes to as many pairs (i,j), especially those with small Hamming distance $H_t(i,j)$. We refer this countervailing force as the Hamming-distance spread measured by $cut(H_t, b_{t+1})/sum(H_t)$. The larger its value is, the better the spread.

Combining these two, our algorithm incrementally adds bits to the hash function to increase the Hamming distance between dissimilar objects while keeping the Hamming distance between similar objects small.

3. Approximate Algorithms

In Section 2, we describe our approach to incrementally learn a hash function that preserves similarities between objects. Our proposed algorithm is simple, but exactly computing (3) has a inherited quadratic cost. It requires computing and constantly updating $H_t(i,j)$ for every possible (i,j) as t increases. This makes the hash learning algorithm intractable for large datasets.

In this section, we introduce two fast linear time approximate algorithms, neither of which compute H_t. Instead, the first uses the property $H_t(i,j) = \sum_{k=1}^{t} d_k(i,j)$, and measures b_{t+1} against each d_k separately. In contrast, the second evaluates the conditional entropy of bit b_{t+1} with each of the previously-learned bits. Before describing our approximate algorithms, we first look at the computation of (3) for the case of two bit hash code, that is $b_{t+1} = b_2$.

3.1. Case with 2-bit Hash Code

When $t + 1 = 2$, there exists a efficient algorithm to compute (3). According to (1),

$$\min_{b_2} L_2 = cut_S(b_2) + \log sum(H_2) - \log sum(H_1) \quad (4)$$

Denote N the number of total training objects and N_1 the number of objects with $b_1(x) = 1$. It can be proved that,

$$sum(H_1) = \sum_{\substack{i,j \\ b_1(i)=b_1(j)}} e^0 + \sum_{\substack{i,j \\ b_1(i)\neq b_1(j)}} e^{-1} = N_1^2 + (N - N_1)^2 + 2N_1(1 - N_1)e^{-1}$$

Now let N_{11} be the number of samples with $b_1(x) = 1$ and $b_2(x) = 1$, $N_{11} \leq N_1$. Similarly N_{10} is the number of samples with $b_1(x) = 0$ and $b_2(x) = 1$. $sum(H_2)$ can be computed using only N, N_1, N_{11} and N_{10}:

$$\begin{aligned} sum(H_2) &= \sum_{\substack{i,j \\ H_2(i,j)=0}} e^0 + \sum_{\substack{i,j \\ H_2(i,j)=1}} e^{-1} + \sum_{\substack{i,j \\ H_2(i,j)=2}} e^{-2} \\ &= N_{11}^2 + (N_1 - N_{11})^2 + N_{10}^2 + (N_0 - N_{10})^2 + \\ &\quad 2e^{-1}\left(N_{11}(N_1 - N_{11}) + N_{10}(N_0 - N_{10})\right) + \\ &\quad 2e^{-1}\left(N_{11}N_{10} + (N_1 - N_{11})(N_0 - N_{10})\right) + \\ &\quad 2e^{-2}\left(N_{11}(N_0 - N_{10}) + N_{10}(N_1 - N_{11})\right) \end{aligned}$$

Using the equations above, we can compute L_2 without explicitly computing $H_2(i,j)$. In addition, because it only takes linear time to get the counts of N_1, N_{11} and N_{10}, this method is a linear time algorithm.

Equation (4) can also be written as:

$$\min_{b_2} L_2 = cut_S(b_2) + \log\left(1 - (1-e^{-1})\frac{cut(H_1, b_2)}{sum(H_1)}\right) \quad (5)$$

As is pointed out in Section 2.2, $cut(H_1, b_2)/sum(H_1)$ is the Hamming distance spread that b_2 induces to H_1. Increasing the spread reduces L_2.

3.2. Approximate Solution: SPEC-Spread Algorithm

According to (3), selecting b_{t+1} depends on $cut(H_t, b_{t+1})/sum(H_t)$ which has quadratic computational complexity. Here we propose an approximate algorithm that avoids this computation. Our algorithm is based on the result for two-bit hash code, and measure b_{t+1} against every bit in H_t separately.

For notational convenience, we rewrite $sum(H_t)$ as $sum(B_t)$. This is valid because Hamming distance H_t is determined by hash function B_t. Similarly, we denote $sum(\{b_k, b_l\}) = \sum_{i,j} e^{-(d_k(i,j)+d_l(i,j))}$ as the sum of the 2-bit hash function $\{b_k, b_l\}$. Given current hash function $B_t = \{b_1, \ldots, b_t\}$, our algorithm decomposes B_t into a set of 1-bit hash functions and measures the improvement b_{t+1} induces on each of these hash function. We select b_{t+1} that minimizes $\hat{L}_{t+1}$:

$$\hat{L}_{t+1} = \max_{b_k \in B_t} \Big\{ cut_S(b_{t+1}) + \log sum(\{b_k, b_{t+1}\}) - \log sum(\{b_k\}) \Big\} \quad (6)$$

Applying (5) to (6), we get

$$\hat{L}_{t+1} = cut_S(b_{t+1}) + \log\left(1 - \min_{b_k \in B_t} \frac{cut(d_k, b_{t+1})}{sum(\{b_k\})}\right) \quad (7)$$

$cut(d_k, b_{t+1})/sum(\{b_k\})$ is the measurement of Hamming-distance spread that b_{t+1} induces on one of these 1-bit hash functions. By applying $\min_{b_k \in B_t}$ we get a lower bound on the Hamming-distance spread. Therefore, $\min_{b_k \in B_t} cut(d_k, b_{t+1})/sum(\{b_k\})$ is a heuristic approximation to $cut(H_t, b_{t+1})/sum(H_t)$. Knowing that b_{t+1} induces certain amount of Hamming-distances spread on any of the binary function in B_T, we expect b_{t+1} to induce good quality spread on Hamming distance H_t, which is the sum of all these one-bit functions.

3.3. Connection with Minimal Conditional Entropy: SPEC-Entropy Algorithm

We notice that conditional entropy $H(b_l|b_k)$ has a strong correlation with the Hamming distance spread $cut(d_k, b_l)/sum(d_k)$. As a matter of fact, the binary function b_l that maximizes $H(b_l|b_k)$ will also be the maximal solution to $cut(d_k, b_l)/sum(\{b_k\})$.

Based on this observation, we develop another heuristic based approximate algorithm that uses negative minimal conditional entropy to approximate the log term in (7):

$$\min_{b_{t+1}} \tilde{L}_{t+1} = cut_S(b_{t+1}) - \eta \min_{b_k \in B_t} H(b_{t+1}|b_k) \quad (8)$$

For a given b_{t+1}, $\min_{b_k \in B_t} H(b_{t+1}|b_k)$ is the lower bound on the conditional entropies between b_{t+1} and each of the binary function in B_t. Minimizing the negative of this bound, in (8), equates with maximizing the minimal conditional entropy. This can be further explained using mutual information.

Let $I(b_{t+1}, b_k)$ be the mutual information between b_{t+1} and b_k. Because $H(b_{t+1}|b_k) = H(b_{t+1}) - I(b_{t+1}, b_k)$, we can rewrite (8) as:

$$\min_{b_{t+1}} \tilde{L}_{t+1} = cut_S(b_{t+1}) - \eta H(b_{t+1}) \quad + \eta \max_{b_k \in B_t} I(b_{t+1}, b_k) \quad (9)$$

According to this equation, binary function b_{t+1} should have small $cut_S(b_{t+1})$, large bit entropy $H(b_{t+1})$ and small mutual information with each of the binary functions in B_t, which is measured by the upper bound $\max_{b_k \in B_t} I(b_{t+1}, b_k)$. As a result of this minimal mutual information constraint, we expect the learned hash function to produce compact codes.

Using N, N_1, N_{11}, N_{10} defined in Section 3.1, and setting $N_0 = N - N_1$, $N_{01} = N_1 - N_{11}$, and $N_{00} = N_0 - N_{10}$, conditional entropy $H(b_l|b_k)$ can be computed in linear time as follows:

$$H(b_l|b_k) = -\frac{N_{11}}{N}\log\frac{N_{11}}{N_1} - \frac{N_{01}}{N}\log\frac{N_{01}}{N_1} - \frac{N_{10}}{N}\log\frac{N_{10}}{N_0} - \frac{N_{00}}{N}\log\frac{N_{00}}{N_0}. \quad (10)$$

4. Implementation

In this work, following [10, 4], we use decision stumps as the binary functions for the hash code. A decision stump performs binary classification by thresholding one input feature value. Stumps can be evaluated quickly, which is ideal for nearest-neighbor applications. When training, the hypothesis space of decision stumps, which we'll call $\mathcal{H}$, is bounded. Specifically, for dataset with N objects and M feature dimensions, the number of hypotheses is

$|\mathcal{H}| = MN$. Using this property together with the special structure of our two hashing algorithms defined in (6) and (8), we can further reduce learning time using dynamic programming.

Let $h \in \mathcal{H}$ denote one of the stumps. Because S in fixed, for each h we can pre-compute $cut_s(h)$ and simply look up, rather than recompute, the value during the learning process. Repeatedly evaluating $\max_{b_k \in B_t} sum(\{b_k, h\})$ in (6) is particularly expensive, and this cost grows as t increases. However, by using the property

$$\max_{b_k \in \{B_{t+1}\}} sum(\{b_k, h\}) = \max\left(\max_{b_k \in B_t} sum(\{b_k, h\}), sum(\{b_{t+1}, h\})\right) \quad (11)$$

for each stump h, we can store the value of $\max_{b_k \in B_t} sum(\{b_k, h\})$, and update it using the recurrence above each time a new binary function is added to the hash function. This reduces the per-bit learning time from $O(tMN)$ to $O(MN)$. A similar trick can be applied to compute $\min_{b_k \in B_t} H(h|b_k)$ in (8).

5. Experiments

We evaluate the performance of our two approximate learning algorithms on two tasks: retrieving semantically similar images from the LabelMe image database, and performing nearest-neighbor recognition of celebrity face images. Results for the algorithm described in section 2.1 are not presented, since the algorithm is too inefficient for large-scale experimentation; our most efficient implementation of it has per-bit learning time $O(MN^2)$.

5.1. LabelMe: Semantically-Similar Image Retrieval

The ability to quickly retrieve visually or semantically similar images from a large collection given a query image is becoming increasingly important in many visual search systems. (For example `http://similar-images.googlelabs.com/`.) Many sophisticated image similarity measures can be expensive to compute, thus prompting the interest in hashing-based approximations [11].

The first experimental dataset, used in [11, 14], consists of approximately 13,500 image thumbnails from LabelMe dataset. Each image is represented using a 512-dimensional Gist feature vector. As in [14], ground truth similarity is obtained by calculating the L2 distance between these Gist vectors, and thresholding the values. The dataset was divided into a training set containing 80% of the samples, and a test set containing the remainder. After training, hash codes were computed for all samples. For each test sample, the nearest neighbors (based on Hamming distance between codes) were found from amongst the training samples, and performance was evaluated by measuring the precision and recall.

Performance is compared to two baseline algorithms. The first is the state of the art *Spectral Hashing* [14], as described earlier. The second is a simple yet effective technique, which we will refer to as *PCA Hashing* [12, 13]. PCA Hashing computes a k-bit hash code by projecting each sample to the k principal components of the training set, then binarizing the coefficients, by setting each to 1 if it exceeds the average value seen for the training set, and 0 otherwise. We also tried applying our algorithms after first transforming the input Gist values using PCA.

The results are displayed in Figure 1. Examination of the precision-recall curve (top left) indicates that each of the SPEC algorithms outperforms Spectral Hashing and PCA Hashing by a generous margin. To create the plot, the hashing algorithm were used to obtain 64-bit codes for the all of the samples. Then for each test sample we located all samples within a fixed Hamming radius and, based on the ground truth similarity information, calculated the precision and the recall. The results are averaged over all test samples, for a range of different Hamming radiuses.

To help interpret the relative behaviour of the hashing algorithms, precision and recall are also plotted separately, as they vary by Hamming radius, in Figure 1 (bottom left) and (bottom right). The plots show that PCA Hashing favours increased precision at the cost of significantly decreased recall, resulting in poor overall performance. Spectral Hashing, SPEC-Spread, and SPEC-Entropy all take more balanced approaches. Interestingly, pre-processing the input using PCA has a very dramatic effect on SPEC-Entropy, greatly reducing the precision and increasing the recall, but the effects on SPEC-Spread are more muted.

Another measure of performance is the precision amongst the top nearest neighbors as the number of bits in the hash code is allowed to vary. The result of this metric, for the top 15 neighbors, is plotted in Figure 1 (top right). Here we can see that for short codes, less than 20 bits, Spectral Hashing outperforms the SPEC algorithms on the standard features, but again the SPEC algorithms dominate for longer codes. Interestingly, PCA-preprocessing of the inputs does seem to be advantageous in this setting, particularly for shorter codes. This makes intuitive sense, since PCA has the effect of concentrating the largest dimensions of data variability in a few input features.

5.2. Celebrity Face Recognition

When performing large scale nearest neighbor face recognition, the computational cost of comparing a test face to a gallery of known faces can be considerable. One way to optimize the search for the nearest-neighboring face in the gallery is to convert all facial feature vectors to binary hash

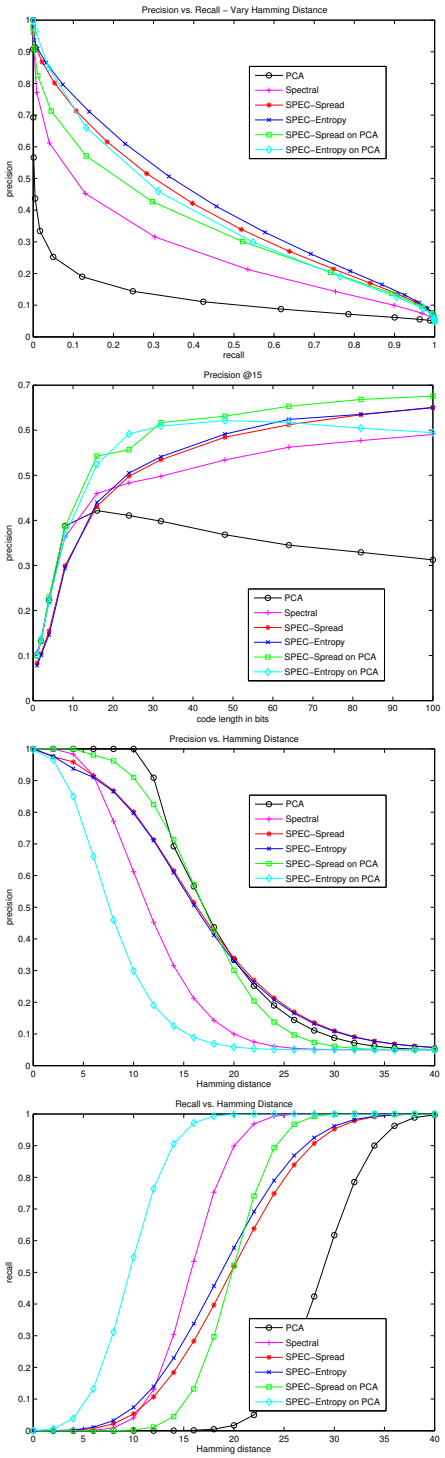

Figure 1. A comparison of hashing algorithms on the LabelMe image retrieval task. (top left) This plot shows the precision vs. recall for the different hashing algorithms, using 64-bit hash codes. (top right) A plot of precision within the top 15 nearest neighbors averaged over test images, as the code length in bits increases from 2 to 100 bits. (bottom left) Precision vs. Hamming radius and (bottom right) Recall vs. Hamming radius plotted separately for 64-bit codes.

codes. Then, assuming Hamming distance between codes preserves semantic similarity—faces of the same subjects map to nearby codes—quick retrieval of a small collection of likely candidates is possible.

To evaluate the feasibility of learning such hashing functions, we collected a set of approximately 276,436 face images, each labelled with the name of the celebrity it depicts, using the method of [15, 17]. Each face is represented using a vector of 1000 real-valued features, obtained by applying Gabor filters at various facial landmark points, then performing LDA-style dimensionality reduction.

The dataset contained 3703 celebrities, each with between 5 and 500 faces. The celebrities were split into two sets: a training set of 2001 celebrities, and a held-out set of 1702, with no intersection between the two. Each of these sets were further subdivided into a gallery, containing 80% of the faces, and a test set, containing the remaining 20%. We trained the hash function using the top performing algorithm from the LabelMe experiment, SPEC-Entropy, on the gallery portion of the training celebrities (totalling 174,747 faces from 2001 celebrities), and computed hash codes for all faces. Our gallery set includes the training set and an additional 45,933 faces from the 1702 held-out celebrities not in the training set.

For this experiment the ground truth similarity matrix was determined from the label information attached to each face. Specifically, two faces labelled with different celebrity names were assigned zero similarity. For faces belonging to the same celebrity, the similarity score was computed by the state of the art Neven vision face recognition system [6]. Finally, to compensate for label noise, small Neven similarities were truncated to zero. It is important to note that the target similarity matrix in this experiment is *semantic* (do two faces belong to the same person?) rather than metric (e.g. L2 distance between feature vectors).

63,398 test faces (44,212 faces from training celebrities and 11,544 faces from hold-out celebrities) were recognized by returning the label of the nearest gallery sample, based on the Hamming distances between hash codes, and recognition accuracy was averaged across all testing samples. The baseline Neven Vision face recognition system on the test set was able to have accuracy 96.22% on training celebrities and 96.23% on hold-out celebrities. The results of our model with 1200 bits had accuracy 95.76% on training celebrities and 95.59% on hold-out celebrities. The detailed results are shown in figure 2.

Thus it is possible using our algorithm to achieve parity in recognition performance using a fraction of the number of bits used by the original feature vector. This provides benefits in terms of reduced storage, as well as greatly reducing the cost of nearest neighbor lookups from a large gallery.

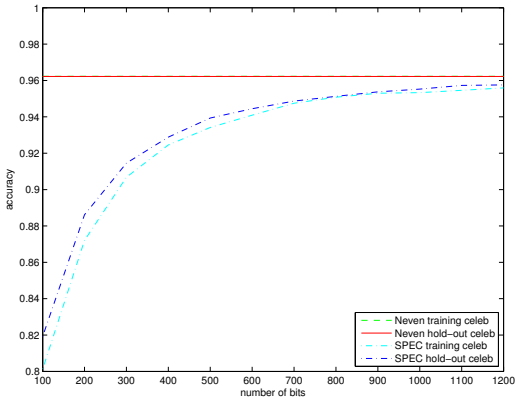

Figure 2. Celebrity face recognition result.

6. Conclusion

In this paper, we presented two efficient algorithms for learning hash functions that produce binary codes. We first described a general incremental learning algorithm that learns the hash function bit-by-bit. Binary functions that assign similar objects bit code and induce large Hamming distance spread are selected and incrementally added to the hash function. This method takes quadratic learning time, so we proposed two approximate linear time algorithms.

We first presented an heuristic approximate algorithm that still favors the selection of binary function will large Hamming distance spread, but takes only linear time. We related this approximate approach to the conditional entropy between pair of bits, and derived our second approximate algorithm. We showed that the second method picks binary functions that minimizes the Hamming distances between similar objects with the constraint that the new bit function should has low mutual information with any of the bit in the existing hash function. This ensures that the hash learning will produce compact hash code.

Experiments on similar image retrieval and celebrity face recognition indicates that our method produce comparable, or superior, results to some of the state-of-the-art methods.

References

[1] J. Bentley. K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 187–197. ACM, 1990.

[2] M. Datar and P. Indyk. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational Geometry*. ACM Press, 2004.

[3] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Computer Vision and Pattern Recognition*, 2008.

[4] Y. Ke, D. Hoiem, and R. Sukthankar. Computer vision for music identification. In *Computer Vision and Pattern Recognition*, 2005.

[5] M. Meila and J. Shi. Learning segmentation by random walks. In *In Advances in Neural Information Processing Systems*, pages 873–879. MIT Press, 2001.

[6] P. Phillips, W. Scruggs, A. O'Toole, P. Flynn, K. Bowyer, C. Schott, and M. Sharpe. FRVT 2006 and ICE 2006 Large-Scale Results. Technical Report NISTIR 7408, National Institute of Standards and Technology, Gaithersburg, MD 20899, March 2007. http://www.frvt.org/.

[7] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Advances in Neural Information Processing Systems 22*. MIT Press, Cambridge, MA, 2009.

[8] A. Rahimi and B. Recht. Random features for large scale kernel machines. In *Advances in Neural Information Processing Systems 22*. MIT Press, Cambridge, MA, 2009.

[9] R. Salakhutdinov and G. Hinton. Semantic hashing. In *SIGIR workshop on Information Retrieval and applications of Graphical Models*. 2007.

[10] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. In *International Conference on Computer Vision*, 2003.

[11] A. Torralba, R. Fergus, and Y. Weiss. Small code and large image databases for recognition. In *Computer Vision and Pattern Recognition*, 2008.

[12] B. Wang, Z. Li, and M. Li. Efficient duplicate image detection algorithm for web images and large-scale database. Technical report, Microsoft Research, 2005.

[13] X.-J. Wang, L. Zhang, F. Jing, and W.-Y. Ma. Annosearch: Image auto-annotation by search. In *IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 1483–1490, 2006.

[14] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*. MIT Press, Cambridge, MA, 2008.

[15] J. Yagnik and A. Islam. Learning people annotation from the web via consistency learning. In *Proceedings of the international workshop on Workshop on multimedia information retrieval*, pages 285–290, 2007.

[16] L. Yang, R. Jin, R. Sukthankar, and F. Jurie. Unifying discriminative visual codebook generation with classifier training for object category recognition. In *Computer Vision and Pattern Recognition*, 2008.

[17] M. Zhao, J. Yagnik, H. Adam, and D. Bau. Large Scale Learning and Recognition of Faces in Web Videos. In *FG*, 2008.