# Fast Classification with Binary Prototypes

**Kai Zhong**[†] **Ruiqi Guo**[‡] **Sanjiv Kumar**[‡]
**Bowei Yan**[†] **David Simcha**[‡] **Inderjit S. Dhillon**[†]
[†] University of Texas at Austin, TX, USA and [‡] Google Research, NY, USA

## Abstract

In this work, we propose a new technique for *fast* k-nearest neighbor (k-NN) classification in which the original database is represented via a small set of learned binary prototypes. The training phase simultaneously learns a hash function which maps the data points to binary codes, and a set of representative binary prototypes. In the prediction phase, we first hash the query into a binary code and then do the k-NN classification using the binary prototypes as the database. Our approach speeds up k-NN classification in two aspects. First, we compress the database into a smaller set of prototypes such that k-NN search only goes through a smaller set rather than the whole dataset. Second, we reduce the original space to a compact binary embedding, where the Hamming distance between two binary codes is very efficient to compute. We propose a formulation to learn the hash function and prototypes such that the classification error is minimized. We also provide a novel theoretical analysis of the proposed technique in terms of Bayes error consistency. Empirically, our method is much faster than the state-of-the-art k-NN compression methods with comparable accuracy.

## 1 Introduction

The classical k-Nearest Neighbor (kNN) classfication technique is attractive due to its simplicity and interpretability. Moreover, when the number of labels is large, the performance of kNN-based classification is very competitive. Theoretically, a kNN classifier asymptotically converges to Bayes optimal classifier as the number of data points approaches infinity [7]. Even though a classical kNN classifier does not need a training phase[1], it requires a huge amount of time and storage for predicting labels of new data points when databases are large. Besides, its performance highly depends on the quality of the distance metric.

To speed up the search process, several approximate nearest neighbor search techniques have been proposed, including tree-based techniques [3, 4, 23, 29], hashing/binary embedding techniques [11, 18, 21, 24, 28, 30], and vector quantization [13]. Tree based methods perform recursive partitioning of the data via a tree data structure to reduce the number of data points to be searched. Binary embedding techniques represent data points as compact binary codes, allowing fast computation of pairwise distances. Vector quantization techniques quantize the data points such that both the storage of the database and the time to calculate the distance are reduced. Most such techniques are traditionally applied in an unsupervised setting.

For fast kNN classification, an alternative strategy has been explored where the database is replaced by a smaller set of prototypes. At prediction time, a query is matched against this prototype set, leading to fast classification. The idea behind these *neighbor compression* techniques is that for classification, not all of the database items are needed. The aim is to remove the redundant points to reduce the search complexity. In some cases, removing redundant points also improves the generalization performance of the kNN classifier by getting rid of noisy points. There are two main approaches to neighbor compression: 1. Prototype Selection, where a subset of the database is chosen as the prototype set, and 2. Prototype Generation, where a set of new points (which are not part of the database) is used as the prototype set.

There exist extensive studies on techniques to generate a representative training set with reduced sample

Proceedings of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS) 2017, Fort Lauderdale, Florida, USA. JMLR: W&CP volume 54. Copyright 2017 by the author(s).

[1]Sometimes training is used to learn appropriate features or distance metric.

size. [8, 25] provide a thorough survey and comparison of different prototype generation/selection algorithms. These can be classified into four types based on the mechanisms adopted: class relabeling, centroid based, space splitting and positioning adjustment. However techniques for learning the prototype set do not scale well with the database size. More recently, [16] proposed a data compression method, Stochastic Neighbor Compression(SNC). The prototypes, or reference vectors, are initialized from a random subset of the training set, and then optimized by minimizing the training classification error.

In all of the above prototype selection/generation works, prototypes are selected or built in the same space as the input data. To retain accuracy similar to a kNN classifier applied to the whole database, the number of prototypes must be large. For instance in SNC, the authors show that they need to retain about 1% or more of the database points to achieve high accuracy. Hence, for a database containing hundreds of millions of points, the prototype set must contain millions of points. kNN classification even in this reduced set can be costly, especially for high dimensional data. In this work, we propose to learn a reduced prototype set, where each prototype is a compact binary code such that distances calculated in the binary space capture the similarity between points in the original input space. We also show that simultaneously learning the reduced set and the binary coding is superior to converting the prototypes learned by other techniques into binary codes in a post-hoc manner.

Supervised learning of binary codes has been proposed previously [15, 19, 21, 26]. The goal in such learning, for instance, can be to map the original data points into binary codes such that neighbors or points that have the same labels have small Hamming distance, and vice-versa. However, simultaneous learning of the reduced prototype set and the binary embedding function is challenging and has not yet been attempted. Moreover, we provide guarantees of convergence to the Bayes optimal classifier which were not provided for previous binary embedding techniques even with the full dataset.

In this paper, we propose a new technique called *Binary Neighbor Compression* (BNC) which learns both the prototypes and the binary hash functions that convert an input point into a binary code. This is achieved by minimizing a margin-based loss, which is a surrogate to the 0-1 loss. We provide Bayes consistency guarantees for our method when the optimal solution is obtained by minimizing a surrogate loss. In particular, when the number of prototypes $m$ is proportional to the number of data points $N$, i.e., $m = \beta N$ for any constant $\beta \in (0, 1]$, and the number of bits of the binary code $r$ satisfies $r = O(N^{\frac{1}{16d}})$, where $d$ is the data dimension, then the generalization error of the solution will converge to the Bayes error as $N$ goes to infinity. We hope this novel Bayes consistency analysis can provide insights for other analyses with binary embedding techniques.

The paper is organized as follows: In Section 2, we formulate the learning problem and the relaxed objective function. In Section 3, we propose a two-phase algorithm to learn the binary prototypes and the hash functions. Section 4 provides the Bayes consistency guarantees. In Section 5, we show our experimental results.

## 2 Binary Neighbor Compression (BNC)

Given $N$ training data points, $X = [\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_N] \in \mathbb{R}^{d\times N}$, and their corresponding labels $[y_1, y_2, \cdots, y_N]$ for $y_i \in \mathcal{Y}$, where $\mathcal{Y}$ is the set of labels, our goal is to find $m$ binary prototypes, each of length $r$, denoted as $B = [\boldsymbol{b}_1, \boldsymbol{b}_2, \cdots, \boldsymbol{b}_m] \in \{-1, 1\}^{r\times m}$ with labels $[z_1, z_2, \cdots, z_m]$, such that the test points can be classified accurately using kNN on the prototypes. For simplicity, we fix the labels of the prototypes and choose the number of prototypes for each label based on the proportion of each class in the dataset. We define a set of $r$ hash functions, i.e., the mapping from input space to $r$-bit binary codes as $g : \mathbb{R}^d \to \{-1, 1\}^r$. We use $\rho_H(\boldsymbol{b}_1, \boldsymbol{b}_2)$ to denote the Hamming distance between two binary codes, $\boldsymbol{b}_1$ and $\boldsymbol{b}_2$. We would like to learn a mapping $g$ and a prototype set $B$ such that for as many data points as possible, $\min_{j:z_j=y_i} \rho_H(g(\boldsymbol{x}_i), \boldsymbol{b}_j) < \min_{j:z_j\neq y_i} \rho_H(g(\boldsymbol{x}_i), \boldsymbol{b}_j)$, where the left term is the smallest distance between a hashed data point and the prototypes with the same label, and the right term is that with a different label.

In this work, we focus on linear hash functions, i.e., $g(\boldsymbol{x}) = \text{sign}\,(W\boldsymbol{x})$, where $W := [\boldsymbol{w}_1, \boldsymbol{w}_2, \cdots, \boldsymbol{w}_r]^T \in \mathbb{R}^{r\times d}$.

Since both $g(\boldsymbol{x})$ and $b_j$ are binary codes with values $\pm 1$,

$$\rho_H(g(\boldsymbol{x}), \boldsymbol{b}_j) = \frac{1}{4}\|g(\boldsymbol{x}) - \boldsymbol{b}_j\|^2 = \frac{1}{2}(r - g(\boldsymbol{x})^T\boldsymbol{b}_j).$$

Given $W$ and $B$, we define a scoring function for each data pair $\{\boldsymbol{x}, y\}$ as

$$\begin{aligned}
&\phi_{W,B}(\boldsymbol{x}, y)\\
:=&2(\min_{j:z_j\neq y} \rho_H(\text{sign}\,(W\boldsymbol{x}), \boldsymbol{b}_j)\\
&- \min_{j:z_j=y} \rho_H(\text{sign}\,(W\boldsymbol{x}), \boldsymbol{b}_j))\\
=&\max_{j:z_j=y} \left(\text{sign}\,(W\boldsymbol{x})^T \boldsymbol{b}_j\right) - \max_{j:z_j\neq y} \left(\text{sign}\,(W\boldsymbol{x})^T \boldsymbol{b}_j\right)
\end{aligned} \tag{1}$$

The prediction for a given $\boldsymbol{x}$ is $\hat{y} = \arg\max_y \phi_{W,B}(\boldsymbol{x}, y)$. The generalization error can be formulated as,

$$R(\phi_{W,B}) := \mathbb{E}[\mathbb{1}[\phi_{W,B}(\boldsymbol{x}, y) < 0]]. \tag{2}$$

The training error is

$$\hat{R}(\phi_{W,B}) = \hat{\mathbb{E}}[\mathbb{1}[\phi_{W,B}(\boldsymbol{x}, y) < 0]],$$

where we denote

$$\hat{\mathbb{E}}[h(\boldsymbol{x}, y)] := \frac{1}{N}\sum_{i=1}^{N} h(\boldsymbol{x}_i, y_i),$$

for some function $h$. Since both the indicator function $\mathbb{1}[\cdot]$ and sign $(\cdot)$ are not differentiable, we replace sign $(\cdot)$ by $\tanh(\gamma(\cdot))$ and $\mathbb{1}[\cdot]$ by $\Phi(\cdot)$ (defined in Eq. (5)). $\Phi$ is also used by [14] to prove the Bayes consistency of 1-NN classifier. Note that if $u$ ($u \neq 0$) is fixed, then $\lim_{\gamma\to\infty} \tanh(\gamma u) = \text{sign}\,(u)$ and $\lim_{\alpha\to 0,\xi\to 0} \Phi(u) = \mathbb{1}[u < 0]$.

Now we formulate our problem,

$$\begin{aligned} \min_{W,B}\ & F(W, B) := \frac{1}{N}\sum_{i=1}^{N} \Phi(\tilde{\phi}_{W,B}(\boldsymbol{x}_i, y_i)), \\ \text{s.t.,}\ & \|\boldsymbol{w}_k\| = 1, \forall k \in [r] \text{ and } B \in \{-1, 1\}^{r\times m}, \end{aligned} \tag{3}$$

where

$$\begin{aligned} \tilde{\phi}_{W,B}(\boldsymbol{x}, y) := & \max_{j:z_j=y} \left(\tanh(\gamma W\boldsymbol{x})^T \boldsymbol{b}_j\right) \\ & - \max_{j:z_j\neq y} \left(\tanh(\gamma W\boldsymbol{x})^T \boldsymbol{b}_j\right) - \nu, \end{aligned} \tag{4}$$

$$\Phi(u) = \begin{cases} 1 & \text{if } u \le \alpha(1-\xi), \\ 0 & \text{if } u \ge \alpha, \\ (\alpha - u)/(\alpha\xi) & \text{otherwise.} \end{cases} \tag{5}$$

Here $\gamma$ is used to approximate sign $(\cdot)$ by $\tanh(\cdot)$, $\{\alpha, \xi\}$ are for the approximation of $\mathbb{1}[\cdot]$ by $\Phi(\cdot)$ and $\nu$ ensures $\tilde{\phi}_{W,B} \neq 0$ when $\gamma$ is large. The constraint $\|\boldsymbol{w}_k\| = 1$ is used with the hope that when $\gamma \to \infty$, $\tanh(\gamma \boldsymbol{w}_k^T \boldsymbol{x}) \to \text{sign}\left(\boldsymbol{w}_k^T \boldsymbol{x}\right)$ because $\boldsymbol{w}_k$ is not independent of $\gamma$. In Sec. 4, we will show that the classifier formed by the optimal solution of the formulation (3) is Bayes consistent. The above constants and constraints are essential to our consistency analysis.

## 3 Algorithm

Although the objective function in Eq. (3) is continuous in $\boldsymbol{x}$, it is difficult to optimize because 1) the objective function is highly non-convex; 2) $B$ takes discrete values. In this section, we discuss the techniques for relieving these difficulties.

First, we relax the non-convex surrogate loss $\Phi(\cdot)$ to the convex hinge loss, which is inspired by recent works on large margin nearest neighbor classification [22, 27]. Second, we relax $B$ to be box-constrained continuous values, i.e., $\tilde{B} \in [-1, 1]^{r\times m}$, where $\tilde{B}$ is the relaxed version of $B$. During the prediction phase, we still use $B = \text{sign}\left(\tilde{B}\right)$. Therefore, we consider minimizing the following surrogate objective for training,

$$\begin{aligned} \min_{W,\tilde{B}} \frac{1}{N}\sum_{i=1}^{N} \Bigg[ & \alpha - \max_{j:z_j=y_i} \left(\tanh(\gamma W\boldsymbol{x}_i)^T \tilde{\boldsymbol{b}}_j\right) \\ & + \max_{j:z_j\neq y_i} \left(\tanh(\gamma W\boldsymbol{x}_i)^T \tilde{\boldsymbol{b}}_j\right)\Bigg]_+ \\ & + \lambda \sum_{k\in[r]} (\|\boldsymbol{w}_k\|^2 - 1)^2, \\ \text{s.t.}\ \ & W \in \mathbb{R}^{r\times d}, \tilde{B} \in [-1, 1]^{r\times m}, \end{aligned} \tag{6}$$

where $[z]_+ = \max\{z, 0\}$ and $\tilde{\boldsymbol{b}}_j$ is $j$-th column of $\tilde{B}$. $\alpha$ absorbs $\nu$ in Eq. (4). The hard constraint $\|\boldsymbol{w}_k\| = 1$ is replaced by the continuous regularization term $(\|\boldsymbol{w}_k\|^2 - 1)^2$.

Next we propose a two-phase algorithm with a special initialization step to optimize this non-convex function. We will show in Sec. 5 that all of these techniques improve the empirical performance of the algorithm.

### 3.1 Initialization

We initialize $W$ with a random Gaussian matrix. It is well known that binary hashing based on random projections, i.e., $\boldsymbol{b} = \text{sign}\,(W\boldsymbol{x})$, where $W$ is a random Gaussian matrix, approximates the Euclidean distance between samples when the number of bits is large and the samples are normalized to unit $L_2$ norm [12]. For initialization of prototypes, we can randomly sample binary codes from the existing hashed data points as initial prototypes. However, good prototypes are typically not a random subset. In real world data, points tend to form local clusters. A better choice of prototypes is to pick representatives of these clusters. Therefore we first do k-means clustering on the data points in each class separately, and then initialize the prototypes as the centers of the clusters. By this construction, each k-means cluster only contains data points from a single class. Let $\mathcal{C}_{y,p}$ denote the $p$-th cluster of Class $y$ and $m_y$ be the number of clusters belonging to Class $y$. Let $\boldsymbol{c}_{y,p}$ be the center of cluster $\mathcal{C}_{y,p}$. Now we show the proposed initialization ensures zero training error when the data is well-behaved.

**Proposition 1.** *Assume that: 1) The magnitude of the data points is equal to* 1*. 2) The clusters from different classes are separated by a margin* $\mu$*. Formally* *speaking, for any data point* $\boldsymbol{x} \in \mathcal{C}_{y,p}$*, assume*

$$\|\boldsymbol{x} - \boldsymbol{c}_{y',p'}\| - \|\boldsymbol{x} - \boldsymbol{c}_{y,p}\| > \mu, \tag{7}$$

*for any* $y' \neq y$ *and any* $p' \in [m_{y'}]$*.*

*Then if* $r \geq \frac{8\log(2mN/\delta)}{\mu^2}$*, w.p.* $1-\delta$*, any data point* $\boldsymbol{x}_i \in \mathcal{C}_{y,p}$ *satisfies*

$$\begin{aligned}&\rho_H(sign(W\boldsymbol{x}_i), sign(W\boldsymbol{c}_{y,p}))\\ <&\rho_H(sign(W\boldsymbol{x}_i), sign(W\boldsymbol{c}_{y',p'}))\end{aligned} \tag{8}$$

*for any* $y' \neq y$ *and any* $p' \in [m_{y'}]$*. In other words, the training error will be zero w.p.* $1-\delta$*.*

**Remark:** The above proposition assumes the data points are separable by the classes, therefore, we can obtain zero training error w.h.p.. This assumption is too strong in real-world settings. So we treat this proposition as a sanity check for our initialization method. Note that our main theorem (Theorem 1) is independent of Proposition 1, so it does not require this assumption. We defer the proof to Appendix A. Also, we will show in Section 5.1 that this k-means initialization empirically works better than random initialization.

### 3.2 Phase 1: Learn linear transformation and relaxed binary prototypes

To learn the parameters $W$ and $\tilde{B}$ from loss function in (6), alternating minimization is a possible choice. However, since each alternating step doesn't have a closed-form solution, we need to introduce an inner loop to iteratively solve $W$ or $\tilde{B}$ when fixing the other. Hence, we directly minimize the loss function (6) over $W$ and $\tilde{B}$ jointly using stochastic gradient descent followed by a projection of $\tilde{B}$ to $[-1,1]^{r\times d}$.

### 3.3 Phase 2: Learn linear transformation with binary prototypes fixed

A caveat of joint learning is that the binarization step in the prediction phase for both $W\boldsymbol{x}$ and $\tilde{B}$ may lead to a noticeable loss in accuracy. Thus we propose to learn $W$ again by fixing $\tilde{B}$ as binary codes after the joint learning step. So, only $\text{sign}(W\boldsymbol{x})$ is approximated by $\tanh(\gamma W\boldsymbol{x})$ in the second phase.

A detailed algorithm for BNC can be found in Algorithm 1. In this algorithm, the compression ratio $\beta$ is $m/N$. The number of prototypes/clusters for each class in the initialization step is set to $\beta N_y$, where $N_y$ is the number of data points in Class $y$.

**Algorithm 1** Stochastic Gradient Descent for Binary Neighbor Compression

```
Require: Dataset $\{\boldsymbol{x}_i, y_i\}_{i=1,2,\dots,N}$, number of bits $r$, the compression ratio $\beta$, and hyper-parameters $\alpha$, $\lambda$ and $\gamma$.
Ensure: $W$ and $B$.
1: Initialize $W$ via random Gaussian distribution.
2: Do k-means clustering for the data points in each class, where the number of clusters for each class is $k_y = \beta N_y$, for $N_y$ being the number of points from Class $y$.
3: Form the centers of the clusters into a matrix $C$, where the columns of $C \in \mathbb{R}^{d\times m}$ are the centers of the clusters $\boldsymbol{c}_{yk}$ ($k$-th cluster of Class $y$) from all the classes.
4: Set the binary prototypes to be $\tilde{B} = B = \text{sign}(WC)$.
5: for $t = 0, 1, 2, \cdots, T1$ do //Phase 1: Jointly learn $W$ and $\tilde{B}$
6:     Calculate the gradient of $W$ and $\tilde{B}$ for Eq.(6) using a small batch.
7:     Update $W$ and $\tilde{B}$ by their gradients.
8:     Project $\tilde{B}$ to the cube $[-1,1]^{r\times m}$.
9: end for
10: $B = \text{sign}\left(\tilde{B}\right)$.
11: for $t = T1+1, \cdots, T2$ do //Phase 2: Learn $W$ when fixing $B$
12:     Calculate the gradient of $W$ for Eq.(6) using a small batch.
13:     Update $W$ by its gradient
14: end for
```

## 4 Consistency Analysis

In the analysis, we consider the Bayes consistency for the binary classification case, i.e., $\mathcal{Y} = \{1, -1\}$. First we introduce some notation. In the binary classification case, we define the generalization risk

$$R(f) := \mathbb{E}[\mathbb{1}[yf(\boldsymbol{x}) < 0]]$$

for a scoring function, $f : \boldsymbol{x} \to \mathbb{R}$. Define the Bayes optimal risk,

$$R^* := \inf_f \mathbb{E}[\mathbb{1}[yf(\boldsymbol{x}) < 0]].$$

For the binary classification case, we can reformulate Eq.(1) as $\tilde{\phi}_{W,B}(\boldsymbol{x}, y) = y\tilde{f}_{W,B}(\boldsymbol{x})$, where

$$\begin{aligned}\tilde{f}_{W,B}(\boldsymbol{x}) := &\max_{j:z_j=1}\left(\tanh(\gamma W\boldsymbol{x})^T\boldsymbol{b}_j\right)\\ &- \max_{j:z_j=-1}\left(\tanh(\gamma W\boldsymbol{x})^T\boldsymbol{b}_j\right) - \nu\end{aligned}$$

We also denote

$$\begin{aligned}f_{W,B}(\boldsymbol{x}) := &\max_{j:z_j=1}\left(\text{sign}(W\boldsymbol{x})^T\boldsymbol{b}_j\right)\\ &- \max_{j:z_j=-1}\left(\text{sign}(W\boldsymbol{x})^T\boldsymbol{b}_j\right) - \nu\end{aligned}$$

**Assumption 1.** *Assume the data points are generated from $\mathcal{S}^{d-1}$, i.e., $\|\boldsymbol{x}\| = 1$ and the data distribution satisfies $\mathbb{P}[|\boldsymbol{w}^T\boldsymbol{x}| \le \theta] \le C_1\theta$ for some constant $C_1$, any fixed unit vector $\boldsymbol{w}$ ($\|\boldsymbol{w}\| = 1$) and any $\theta > 0$.*

**Remark:** The above assumption holds when, for example, data points are distributed uniformly on $\mathcal{S}^{d-1}$ (see Lemma 2.2 [6]).

Let $[W^*, B^*]$ be an optimal solution of Eq. (3),

$$[W^*, B^*] := \underset{\|\boldsymbol{w}_k\|=1, B\in\{-1,1\}^{r\times m}}{\arg\min} \frac{1}{N}\sum_{i=1}^{N} \Phi(y_i \tilde{f}_{W,B}(\boldsymbol{x}_i)). \tag{9}$$

The following theorem shows the Bayes consistency of our classifier.

**Theorem 1.** *Let $\beta$ be any constant satisfying $0 < \beta \le 1$. Set $r = C\frac{d^{3/2}\log d}{2^d}N^{\frac{1}{16d}}$, $m = \beta N$ and $\alpha = \nu = 32N^{-\frac{1}{16d^2}}$, $\gamma = C_1(d)N^{\frac{3}{32d}}$, $\xi = C_2(d)N^{-\frac{1}{32d}}$, where $C$ is a constant and $C_1(d)$, $C_2(d)$ depend only on $d$. Then under Assumption 1,*

$$R(f_{W^*,B^*}) \xrightarrow{p} R^* \quad \text{as } N \to \infty, \tag{10}$$

*where $p$ represents convergence in probability.*

**Remark:** Theorem 1 shows that the optimal solution of our model converges to the Bayes optimal classifier as long as the compression ratio is a constant satisfying $0 < \beta \le 1$ and the number of bits satisfies $r = O(N^{\frac{1}{16d}})$. For the compression ratio, if we don't care about the convergence rate, it can be set to be any constant among $(0, 1]$. But a smaller $\beta$ will hurt the convergence rate to some extent (see the probabilities in Lemma 3 and Lemma 4). In practice, given a large number of data points, a small compression ratio suffices for good generalization performance, as shown by our experiments. To the best of our knowledge, this is the first Bayes consistency guarantee for compressed binary embedding techniques. However, this analysis assumes the non-convex objective with discrete values, Eq. (9), to be minimized exactly, which is not guaranteed by our current algorithm. We leave this problem as a future work. Our empirical results show that our method achieves the generalization performance of state-of-the-art methods.

**Proof Sketch.** The key idea of this proof follows from the fact that when $r$ and $m$ are large enough, the classifier $\tilde{f}_{W^*,B^*}$ can classify as well as any 2-Lipschitz function, which is a Bayes consistent classifier (see Lemma 5 in [14]). Specifically, we bound the difference between the generalization error of the Bayes optimal classifier and our classifier by introducing some intermediate terms, each of which can be bounded. Let

$$R(f_{W^*,B^*}) - R^* \tag{11a}$$
$$= \mathbb{E}[\mathbb{1}[y f_{W^*,B^*}(\boldsymbol{x}) < 0]] - \mathbb{E}[\mathbb{1}[y \tilde{f}_{W^*,B^*}(\boldsymbol{x}) < 0]] \tag{11b}$$
$$+ \mathbb{E}[\mathbb{1}[y \tilde{f}_{W^*,B^*}(\boldsymbol{x}) < 0]] - \mathbb{E}[\Phi(y \tilde{f}_{W^*,B^*}(\boldsymbol{x}))] \tag{11c}$$
$$+ \mathbb{E}[\Phi(y \tilde{f}_{W^*,B^*}(\boldsymbol{x}))] - \hat{\mathbb{E}}[\Phi(y \tilde{f}_{W^*,B^*}(\boldsymbol{x}))] \tag{11d}$$
$$+ \hat{\mathbb{E}}[\Phi(y \tilde{f}_{W^*,B^*}(\boldsymbol{x}))] - \hat{\mathbb{E}}_\beta[\Phi(y f^*_{2\alpha}(\boldsymbol{x}))] \tag{11e}$$
$$+ \hat{\mathbb{E}}_\beta[\Phi(y f^*_{2\alpha}(\boldsymbol{x}))] - \mathbb{E}[\Phi(y f^*_{2\alpha}(\boldsymbol{x}))] \tag{11f}$$
$$+ \mathbb{E}[\Phi(y f^*_{2\alpha}(\boldsymbol{x}))] - \mathbb{E}[\mathbb{1}[y f^*_{2\alpha}(\boldsymbol{x}) < \alpha]] \tag{11g}$$
$$+ \mathbb{E}[\mathbb{1}[y f^*_{2\alpha}(\boldsymbol{x}) < \alpha]] - R^*, \tag{11h}$$

where $\hat{\mathbb{E}}_\beta[h(\boldsymbol{x}, y)] := \sum_{i\in\Omega_\beta} \frac{1}{|\Omega_\beta|} h(\boldsymbol{x}_i, y_i)$ for a random subset $\Omega_\beta \subset [N]$ with size $|\Omega_\beta| = \beta N$, $f^*_{2\alpha} := \arg\min_{f\in\mathcal{F}_2} \mathbb{E}[\mathbb{1}[yf(\boldsymbol{x}) < \alpha]]$ and $\mathcal{F}_2$ is the family of 2-Lipschitz functions.

The first term Eq. (11b), which is due to the approximation of $\tanh(\cdot)$ to sign $(\cdot)$, can be bounded when $\gamma \to \infty$ under Assumption 1. The second term Eq. (11c) and the last-but-one term Eq. (11g) are less than zero by the definition of $\Phi$. The third term Eq. (11d) is the difference between the empirical loss and the population loss, which can be bounded by Rademacher complexity, since the loss function $\Phi$ and the classifier $\tilde{f}_{W^*,B^*}$ are Lipschitz functions. The fourth term Eq. (11e) is the difference between the loss of our classifier and the loss of the best 2-Lipschitz classifier, which can be bounded when $r$ and $m$ are large enough. The fifth term Eq. (11f) is the empirical error and the population error of a given classifier, thus can be bounded by Hoeffding inequality. The last term Eq. (11h) is the difference between the best 2-Lipschitz classifier and the Bayes optimal classifier, which has been studied in Lemma 5 in [14]. The details of these bounds can be found from Lemma 1 to Lemma 5 below. Finally, by setting proper values for the hyper-parameters, Eq. (10) will hold. The detailed proof about how to set the hyperparameters can be found in Appendix A.2.

The following lemmata provide bounds for most terms in Eq. (11). Their proofs can be found in the appendix.

**Lemma 1** (Bound for Eq. (11b))**.** *Let $\gamma \ge \frac{C_1 r}{2\delta}\log(\frac{32r}{\nu})$. Then under Assumption 1,*

$$|\mathbb{E}[\mathbb{1}[y f_{W^*,B^*}(\boldsymbol{x}) < 0]] - \mathbb{E}[\mathbb{1}[y \tilde{f}_{W^*,B^*}(\boldsymbol{x}) < 0]]| \le \delta \tag{12}$$

**Lemma 2** (Bound for Eq. (11d))**.** *For any $\epsilon$ satisfying*

$$\frac{2^{1+1/(4d+4)}(32r\gamma)^{d/(2d+2)}}{\sqrt{\alpha\xi}N^{1/(4d+4)}} < \epsilon < 1, \tag{13}$$

*we have w.p. $1 - e^{-N\epsilon^2/C_3}$*

$$|\mathbb{E}[\Phi(y \tilde{f}_{W^*,B^*}(\boldsymbol{x}))] - \hat{\mathbb{E}}[\Phi(y \tilde{f}_{W^*,B^*}(\boldsymbol{x}))]| \le 2\epsilon, \tag{14}$$

*where $C_3$ is a constant.*

**Lemma 3** (Bound for Eq. (11e)). *Let the data points be sampled from the unit sphere, i.e., $\|\boldsymbol{x}\| = 1$, $\forall \boldsymbol{x} \in \mathcal{X}$. If $r \geq C_2 d^{3/2} \log(d)/(\frac{\alpha(1-\xi)}{8})^{d-1}$, $\gamma \geq \frac{4(\nu+\alpha)}{\alpha(1-\xi)}$ and $m = \beta N$, then for some $\epsilon$ satisfying*

$$\frac{2^{1+1/(4d+4)}(32r\gamma)^{d/(2d+2)}}{\sqrt{\alpha\xi}(\beta N)^{1/(4d+4)}} < \epsilon < 1, \tag{15}$$

*w.p. $1 - 2e^{-\beta N\epsilon^2/C_3}$, we have*

$$\hat{\mathbb{E}}[\Phi(y\tilde{f}_{W^*,B^*}(\boldsymbol{x}))] - \hat{\mathbb{E}}_\beta[\Phi(yf^*_{2\alpha}(\boldsymbol{x}))] \leq 4\epsilon \tag{16}$$

*where $C_2$ is a constant.*

**Lemma 4** (Bound for Eq. (11f)). *W.p. $1 - 2e^{-2\beta N t^2}$*

$$|\hat{\mathbb{E}}_\beta[\Phi(yf^*_{2\alpha}(\boldsymbol{x}))] - \mathbb{E}[\Phi(yf^*_{2\alpha}(\boldsymbol{x}))]| \leq t \tag{17}$$

*for any $t > 0$*

**Lemma 5** (Bound for Eq. (11h), Lemma 5 in [14]).

$$R^* = \lim_{\alpha\to 0} \inf_{f\in\mathcal{F}_2} \mathbb{E}[\mathbb{1}(yf(\boldsymbol{x}) < \alpha)].$$

## 5 Experimental Results

In this section, we compare our Binary Neighbor Compression (BNC) method on 1-NN classification task with state-of-the-art techniques. [16] has shown that SNC (Stochastic Neighbor Compression) outperforms existing techniques, kNN, CNN [10] and FCNN [2]. One obvious question is what happens if, instead of jointly learning binarization and neighbor compression as proposed in our work, we binarize the prototypes learned by SNC using a standard hashing technique. To this end, we construct a simple baseline called SNC+LSH for comparison, where we use signed-random LSH for binarization. In particular, we take $B_s = \text{sign}\,(WP + \boldsymbol{b})$ as the binary prototypes, where $W$ is a random Gaussian matrix, $\boldsymbol{b}$ is a random Gaussian vector, and $P$ is the real-valued prototypes generated by SNC. Note that there exist numerous approximate nearest neighbor (ANN) methods in the literature. However, most of them are unsupervised methods, such as Product Quantization [13], tree-based ANN methods, while our BNC method is a supervised method. With the same budget of memory and prediction time, supervised methods typically have better performance on classification task than unsupervised methods. So it is unfair to compare BNC with unsupervised methods. Nevertheless, we still include one of recent ANN packages, ANNOY [1], to make a comparison. In summary, we compare our method with kNN, CNN, FCNN, SNC, SNC+LSH and ANNOY on the memory usage, prediction time and testing accuracy.

We compare all aforementioned methods on these benchmark datasets: isolet, letters, adult, w8a, yalefaces and mnist. Statistics of the datasets are shown in Table 1. These datasets are those used in SNC [16]. We do so to avoid any selection bias on the datasets. We refer interested reader to [16] for a full description of these datasets.

We fix the number of bits to be 128 for BNC. Two compression ratios (ratios between number of prototypes and sample size), 1% and 8%, are used for all methods those are applicable. We set the number of trees as 10 and 1 for ANNOY. Note that a larger number of trees leads to higher accuracy but slower computation. We find that empirical performance is not sensitive to the parameters $\gamma, \lambda$, so we fix them to be $\gamma = 1, \lambda = 1$. The batch size of SGD is set as 128. The learning rate of SGD and the margin $\alpha$ are chosen to achieve the fastest decrease of the training error. For SNC+LSH we simply binarize the output prototypes of SNC. For the calculation of kNN classification for real-valued vectors, we use the *knncl* function provided by LMNN2 [27] and the Hamming distance calculation is implemented using *Yael* library[2]. For ANNOY, except for the number of trees, all the other parameters are set as default values. The results of SNC, CNN and FCNN come from the numbers reported in [16].

**Prediction Time.** When doing prediction, BNC needs a linear projection onto a $r$-dim space to get the binary code, followed by computing the Hamming distance between the hashed code and all binary prototypes. The first step takes $O(rd)$ time, and the second step takes $O(m)$ Hamming distance calculations for $r$-bit codes, which is very fast. For SNC, the prediction time is $O(md)$. When $r \ll m$, which is usually the case, it is much slower than BNC.

**Memory Usage.** BNC only needs to store $W$, $B$ and the labels of the prototypes, which are $O(rd)$ real numbers and $m$ $r$-bit binary codes. SNC need store $m$ $d$-dimensional non-binary prototypes, which counts for $O(md)$ real numbers. Since ANNOY needs to save all the original data points, there is almost no memory reduction by ANNOY.

Both CNN and FCNN select a subset of the training set, so they will have similar memory reduction and speedup to SNC, while SNC+LSH has the same memory complexity and time complexity as BNC. Therefore we only list the memory usage and prediction time for kNN, BNC, SNC and ANNOY in Table 1. To highlight the changes, we list the difference of each method with

[2]http://yael.gforge.inria.fr/index.html

Table 1: Dataset descriptions and performance comparison of different neighbor compression methods against brute force kNN on the entire dataset. The speedup and memory consumption are shown relative to kNN. 'x%' implies the fraction of database points kept as prototypes by neighbor compression techniques. "10 ANN" and "1 ANN" imply ANNOY with 10 trees and 1 tree repsectively. BNC tends to have much higher speed and lower memory usage than SNC and ANNOY.

| | $\|\mathcal{Y}\|$ | dim | #Train | #Test | kNN | | 1% BNC | 8% BNC | 1% SNC | 8% SNC | 10 ANN | 1 ANN |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| adult | 2 | 50 | 32561 | 16281 | 19.32s | speedup | **479.51** | 171.77 | 115.92 | 31.67 | 66.12 | 106.68 |
| | | | | | 12M | mem. red. | 1500.00 | 375.00 | 125.00 | 15.96 | 1.09 | 1.71 |
| w8a | 2 | 100 | 49749 | 14951 | 52.47s | speedup | **1028.82** | 295.60 | 248.91 | 41.09 | 203.58 | 257.47 |
| | | | | | 31M | mem. red. | 3875.00 | 704.55 | 117.42 | 15.50 | 1.24 | 1.55 |
| isolet | 26 | 172 | 3898 | 3899 | 0.77s | speedup | **33.15** | 30.76 | 16.83 | 8.48 | 4.38 | 5.49 |
| | | | | | 4.9M | mem. red. | 1225.00 | 612.50 | 94.23 | 12.13 | 1.58 | 1.81 |
| letters | 26 | 16 | 16K | 4K | 2.36s | speedup | **274.71** | 123.69 | 101.39 | 25.08 | 46.50 | 86.65 |
| | | | | | 1.9M | mem. red. | 475.00 | 118.75 | 95.00 | 12.84 | 0.59 | 1.36 |
| yalefaces | 38 | 100 | 1961 | 491 | 0.10s | speedup | **60.24** | 56.89 | 35.31 | 14.22 | 9.29 | 10.92 |
| | | | | | 1.4M | mem. red. | 350.00 | 350.00 | 87.50 | 11.67 | 1.40 | 1.72 |
| mnist | 10 | 164 | 60K | 10K | 93.46s | speedup | **1865.53** | 659.12 | 419.87 | 56.29 | 155.97 | 187.64 |
| | | | | | 72M | mem. red. | 6000.00 | 947.37 | 97.30 | 12.41 | 1.57 | 1.85 |

Table 2: Comparison of testing accuracy of different methods. * These values are missing in the paper [16], so the best performance among kNN, CNN/FCNN using other compression ratios is filled.

| Dataset | adult | w8a | isolet | letters | yale-faces | mnist |
|---|---|---|---|---|---|---|
| KNN | 0.793 | 0.979 | 0.944 | 0.967 | 0.945 | 0.978 |
| 8% BNC | 0.838 | 0.990 | 0.931 | 0.929 | 0.937 | 0.968 |
| 8% SNC | 0.812 | 0.994 | 0.951 | 0.972 | 0.943 | 0.982 |
| 8% SNC+LSH | 0.765 | 0.981 | 0.916 | 0.483 | 0.933 | 0.961 |
| 8% CNN | 0.600 | 0.994* | 0.920 | 0.780 | 0.780 | 0.960 |
| 8% FCNN | 0.570 | 0.994* | 0.944* | 0.720 | 0.730 | 0.960 |
| 10 trees ANNOY | 0.79 | 0.97 | 0.93 | 0.94 | 0.93 | 0.96 |
| 1% BNC | 0.838 | 0.991 | 0.934 | 0.914 | 0.945 | 0.969 |
| 1% SNC | 0.820 | 0.991 | 0.944 | 0.898 | 0.945 | 0.978 |
| 1% SNC+LSH | 0.801 | 0.962 | 0.872 | 0.169 | 0.265 | 0.939 |
| 1% CNN | 0.550 | 0.953 | 0.740 | 0.330 | 0.280 | 0.720 |
| 1% FCNN | 0.530 | 0.961 | <0.700 | <0.300 | <0.200 | <0.700 |
| 1 tree ANNOY | 0.79 | 0.89 | 0.91 | 0.89 | 0.88 | 0.94 |

respect to kNN. We can see from the table that BNC is much more efficient, both in terms of speed and memory usage. Compared to full kNN, it reduces memory consumption by *hundreds or thousands* of times across all datasets while speeding up prediction by hundreds or thousands of times as well.

**Testing Accuracy.** We illustrate the testing accuracy of different methods in Table 2. For most datasets, both BNC and SNC have competitive results compared to full kNN and are better than other compression methods, CNN, FCNN and SNC+LSH, ANNOY. 128-bit binary prototypes of our method achieve almost the same testing accuracy as that of SNC, which uses non-binary prototypes. With the same length binary codes and the same ratio of neighbor compression, BNC outperforms SNC+LSH on all datasets. This implies that the joint optimization of $W$ and $B$ is indeed useful for learning better binary prototypes. Note that the *letters* dataset has very low accuracy after LSH. This is because some features in the *letters* dataset are much more important than others. LSH eliminates the effect of these important features in the randomized projection step.

## 5.1 Performance from Different Phases

Next, we analyze the testing accuracy of our algorithm in different phases under different initializations, as shown in Table 3. K-means initialization usually has performance superior to random initialization before optimization. First phase (jointly optimizing $W$ and $B$) significantly improves accuracy over K-means initialization. In the second phase (fixing $B$), the performance further improves compared to the first phase.

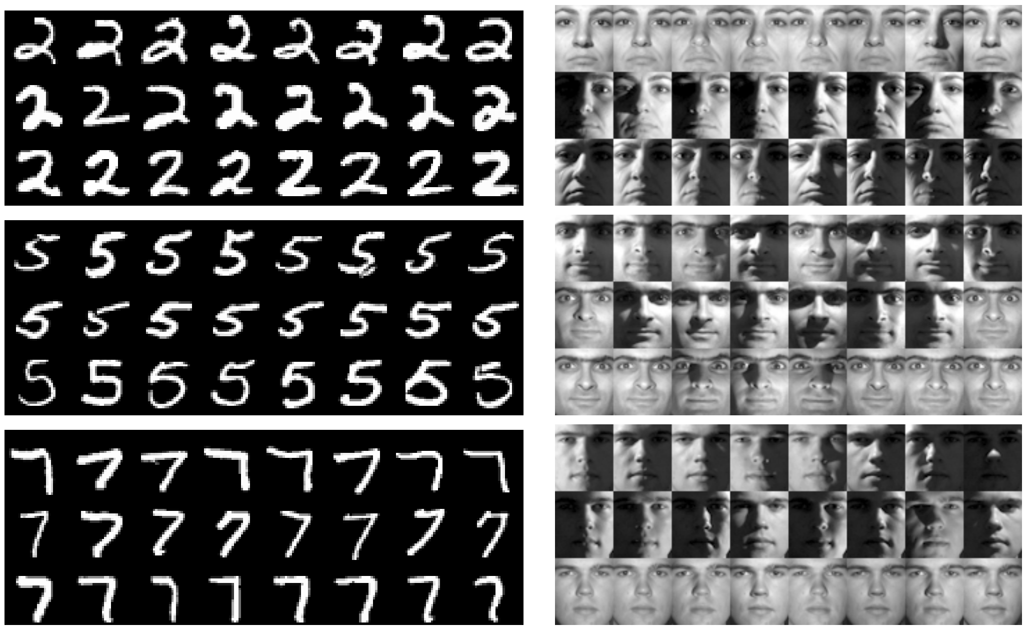
## 5.2 Different Number of Prototypes

The number of prototypes chosen is closely related to the accuracy of classification. As seen in Table 2, the accuracy increases as we use more prototypes per category for both BNC and SNC, at the price of increased memory consumption and prediction time. However, a very small number of prototypes often achieves reasonable performance. Clearly, the number of prototypes

Table 3: Testing accuracy from different phases of the proposed BNC algorithm for 1% compression ratio.

| Dataset | adult | w8a | isolet | letters | yale-faces | mnist |
|---|---|---|---|---|---|---|
| Random init. | 0.758 | 0.971 | 0.663 | 0.405 | 0.550 | 0.950 |
| K-means init. | 0.794 | 0.979 | 0.846 | 0.464 | 0.780 | 0.959 |
| After Phase 1 | 0.840 | 0.974 | 0.900 | 0.880 | 0.945 | 0.967 |
| After Phase 2 | 0.838 | 0.990 | 0.934 | 0.914 | 0.945 | 0.969 |

Figure 1: Prototype visualization on mnist and yalefaces. Each block of three rows visualizes a category, and each row of a block visualizes a distinct prototype of that category with the eight instances having the smallest Hamming distance to the binary prototype.



required is closely related to the separability of data from different classes. In this work, we have treated this as a model selection problem.

### 5.3 Illustration of Prototypes

We use multiple binary prototypes to represent a category. This model is similar to multiple-prototype theory in categorization, which suggests that each prototype should be able to capture a locally similar space within a category. To verify this, we visualize the examples closest to each binary prototype. We perform BNC on the MNIST and yalefaces dataset where we fix each category to have 3 prototypes. Different prototypes tend to capture semantically similar neighborhoods in the input space.In Figure 1, we visualize each prototype by plotting the 8 instances whose hamming distance is closest to the binary prototype. As we can see from the MNIST example, different prototypes capture different writing styles of the same character. For example, number “2" can be written with a straight or a round bottom, as captured by the first and the third prototypes in Figure 1.

## 6 Conclusion and Future work

We have proposed a technique where a reduced set of binary prototypes are learned to speed up kNN classification. The theoretical analysis and the empirical performance both demonstrate its efficacy over the existing methods. Note that instead of fixing the number of prototypes for each class according to the fraction of points in that class, classes that are harder to separate from other classes need more prototypes. It will be useful to have the algorithm automatically learn the number of prototypes for each class.

**Acknowledge**

This research was supported by NSF grants CCF-1320746, IIS-1546452 and CCF-1564000.

## References

[1] Approximate nearest neighbors oh yeah. GitHub code, https://github.com/spotify/annoy.

[2] Fabrizio Angiulli. Fast condensed nearest neighbor rule. In *ICML*, pages 25–32. ACM, 2005.

[3] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[4] Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *ICML*, pages 97–104. ACM, 2006.

[5] Károly Böröczky Jr and Gergely Wintsche. Covering the sphere by equal spherical balls. In *Discrete and computational geometry*, pages 235–251. Springer, 2003.

[6] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of a theorem of johnson and lindenstrauss. *Random Structures & Algorithms*, 22(1):60–65, 2003.

[7] Richard O Duda, Peter E Hart, et al. *Pattern classification and scene analysis*. Wiley New York, 1973.

[8] Salvador Garcia, Joaquín Derrac, José Ramón Cano, and Francisco Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study. *PAMI*, 34(3):417–435, 2012.

[9] Lee-Ad Gottlieb, Aryeh Kontorovich, and Robert Krauthgamer. Efficient regression in metric spaces via approximate lipschitz extension. In *Similarity-Based Pattern Recognition*, pages 43–58. Springer, 2013.

[10] Phil Hart. The condensed nearest neighbor rule. *Information Theory, IEEE Transactions on*, 14(3):515–516, 1968.

[11] Piotr Indyk and Rajeev Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*, pages 604–613. ACM, 1998.

[12] Laurent Jacques, Jason N Laska, Petros T Boufounos, and Richard G Baraniuk. Robust 1-bit compressive sensing via binary stable embeddings of sparse vectors. *Information Theory, IEEE Transactions on*, 59(4):2082–2102, 2013.

[13] Herve Jegou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. *PAMI*, 33(1):117–128, 2011.

[14] Aryeh Kontorovich and Roi Weiss. A bayes consistent 1-nn classifier. In *AISTATS*, 2015.

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

[16] Matt Kusner, Stephen Tyree, Kilian Q Weinberger, and Kunal Agrawal. Stochastic neighbor compression. In *ICML*, pages 622–630, 2014.

[17] Michel Ledoux and Michel Talagrand. *Probability in Banach Spaces: isoperimetry and processes*. Springer Science & Business Media, 2013.

[18] Ke Li and Jitendra Malik. Fast k-nearest neighbour search via dynamic continuous indexing. In *ICML*, pages 671–679, 2016.

[19] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081. IEEE, 2012.

[20] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of machine learning*. MIT Press, 2012.

[21] Mohammad Norouzi and David M Blei. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011.

[22] Mohammad Norouzi, David M Blei, and Ruslan R Salakhutdinov. Hamming distance metric learning. In *NIPS*, pages 1061–1069, 2012.

[23] Stephen Malvern Omohundro. *Five balltree construction algorithms*. Berkeley: International Computer Science Institute, 1989.

[24] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *CVPR*, pages 37–45, 2015.

[25] Isaac Triguero, Joaquín Derrac, Salvador Garcia, and Francisco Herrera. A taxonomy and experimental study on prototype generation for nearest neighbor classification. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 42(1):86–100, 2012.

[26] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3424–3431. IEEE, 2010.

[27] Kilian Q Weinberger and Lawrence K Saul. Distance metric learning for large margin nearest neighbor classification. *The Journal of Machine Learning Research*, 10:207–244, 2009.

[28] Yair Weiss, Antonio Torralba, and Rob Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2009.

[29] Peter N Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *SODA*, 1993.

[30] Felix Yu, Sanjiv Kumar, Yunchao Gong, and Shih-fu Chang. Circulant binary embedding. In *ICML*, pages 946–954, 2014.