Supplementary material for: Optimizing affinity-based binary hashing using auxiliary coordinates

Ramin Raziperchikolaei Miguel Á. Carreira-Perpiñán
Electrical Engineering and Computer Science, University of California, Merced
http://eecs.ucmerced.edu

May 20, 2016

Abstract

We provide the following. 1) Pseudocode for our MAC algorithm. 2) Proofs for the statements in the main paper [12] (paragraph "theoretical results") regarding the path of optima of the $\mathcal{L}_P$ objective function, and the stopping criterion. 3) A detailed description of the stopping criterion and the practical construction of a good μ schedule. 4) A detailed description of the $\mathbf{Z}$ step of our MAC algorithm to learn the binary codes given the hash functions. 5) Extended experiments with additional datasets, different numbers of bits b, precision/recall curves, and training on centered and normalized vectors.

1 Theoretical results

We give theoretical results regarding the main paper [12]. We quote the following equations from that paper, to which we refer frequently:

- The affinity-based objective function we want to optimize:

$$\min \mathcal{L}(\mathbf{h}) = \sum_{n,m=1}^{N} L(\mathbf{h}(\mathbf{x}_n), \mathbf{h}(\mathbf{x}_m);\ y_{nm}). \tag{1}$$

- The MAC-constrained problem:

$$\min_{\mathbf{h},\mathbf{Z}} \sum_{n=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm}) \quad \text{s.t.} \quad \mathbf{z}_1 = \mathbf{h}(\mathbf{x}_1), \cdots, \mathbf{z}_N = \mathbf{h}(\mathbf{x}_N). \tag{2}$$

- The quadratic-penalty objective function:

$$\min \mathcal{L}_P(\mathbf{h}, \mathbf{Z}; \mu) = \sum_{n,m=1}^{N} L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm}) \ + \mu \sum_{n=1}^{N} \|\mathbf{z}_n - \mathbf{h}(\mathbf{x}_n)\|^2 \quad \text{s.t.} \quad \mathbf{z}_1, \ldots, \mathbf{z}_N \in \{-1, 1\}^b. \tag{3}$$

Fig. 1 gives the overall MAC algorithm to learn a hash function by optimizing an affinity-based loss function.

input $\mathbf{X}_{D\times N} = (\mathbf{x}_1, \ldots, \mathbf{x}_N)$, $\mathbf{Y}_{N\times N} = (y_{nm})$, $b \in \mathbb{N}$
Initialize $\mathbf{Z}_{b\times N} = (\mathbf{z}_1, \ldots, \mathbf{z}_N) \in \{0,1\}^{bN}$
for $\mu = 0 < \mu_1 < \cdots < \mu_\infty$
  for $i = 1, \ldots, b$ — **h step**
    $h_i \leftarrow$ fit hash function to $(\mathbf{X}, \mathbf{Z}_{\cdot i})$
  repeat — **Z step**
    for $i = 1, \ldots, b$
      $\mathbf{Z}_{\cdot i} \leftarrow$ approximate minimizer of $\mathcal{L}_P(\mathbf{h}, \mathbf{Z}; \mu)$ over $\mathbf{Z}_{\cdot i}$
  until no change in $\mathbf{Z}$ or `maxit` cycles ran
  if $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ **then** stop
return $\mathbf{h}$, $\mathbf{Z} = \mathbf{h}(\mathbf{X})$

Figure 1: MAC algorithm to optimize an affinity-based loss function for binary hashing.

1.1 Summary of results

We characterize the relation between $\mathcal{L}(\mathbf{h})$, the objective function of eq. (1) we really want to minimize, and $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$, the proxy function of eq. (3) we actually minimize as $\mu \to \infty$. We establish the following:

- A global minimizer of $\mathcal{L}(\mathbf{h})$ can actually be found by globally minimizing $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ for $\mu \geq \mu_\infty$. This means we need not drive $\mu \to \infty$. Also, the graph of the minima of $\mathcal{L}_P$ is always below the graph of the minima of $\mathcal{L}$ but they coincide for $\mu \geq \mu_\infty$.
- The global minimum of $\mathcal{L}_P$ as a function of $\mu \geq 0$ is a nonnegative, nondecreasing, continuous and piecewise linear function and it has a finite number of pieces $0 < \mu_1 < \cdots < \mu_\infty < \infty$. For an algorithm, this implies that there is no need to start with $\mu < \mu_1$, and that no further changes occur for $\mu > \mu_\infty$.
- If instead of considering global optimization over $(\mathbf{Z}, \mathbf{h})$ for each μ one considers an alternating optimization as in the MAC algorithm, then the approximate minima of $\mathcal{L}_P$ is also a piecewise linear function. So one can start with $\mu \geq \mu_1$ and the algorithm stops at a finite μ value, which occurs whenever $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ after a $\mathbf{Z}$ step. However, the resulting $(\mathbf{Z}, \mathbf{h})$ need not be a global minimizer of $\mathcal{L}_P$ or $\mathcal{L}$. The result depends on the initial $\mathbf{Z}$ value (for $\mu = 0$) and on the schedule of μ values used. Following the path over μ slowly leads to larger improvements over the initial $\mathbf{Z}$.

These results are in stark contrast with the behavior of quadratic-penalty methods in continuous optimization [11]. In the latter, the path of minimizers is a continuous function of μ, and one must drive $\mu \to \infty$ in order for the path of global minima of the quadratic-penalty objective function to reach a minimum of the original objective function. Also, with nonconvex problems there are different such paths, each leading to a different local minimum or stationary point in general, and each path may only be defined for sufficiently large values of μ. Finally, in continuous optimization ill-conditioning typically appears as $\mu \to \infty$, because the Hessian of the penalty term is low-rank and dominates the Hessian of the quadratic-penalty function for large μ. Hence, numerically optimizing at large μ values has very slow convergence. This is the reason why practical optimization with quadratic-penalty methods requires following the path from small values of μ.

In our case, the geometry of this path is simpler. The path is piecewise linear, with $(\mathbf{Z}, \mathbf{h})$ being constant within each piece, so the path is really a finite sequence of $(\mathbf{Z}_i, \mathbf{h}_i)$ pairs. Ill-conditioning does not arise because we stop with a finite μ (and the optimization over $\mathbf{Z}$ works on a discrete space anyway). It is tempting to solve the problem by doing a minimization of $\mathcal{L}_P$ for a single, large enough value of μ directly, since this will solve the problem over $\mathcal{L}$ (from theorems 1.2–1.3). However, this requires finding the global minimum of $\mathcal{L}_P$ over $(\mathbf{Z}, \mathbf{h})$ jointly, which is impractical, and alternating optimization will generally not find it. The reason is that, if we set μ to a large enough value, alternating optimization over $(\mathbf{Z}, \mathbf{h})$ of $\mathcal{L}_P$ will stop in one step (theorem 1.12) with a suboptimal result corresponding to fitting $\mathbf{h}$ to the current $\mathbf{Z}$ (thus behaving like a "filter", or two-step, approach), where the quality of the resulting $\mathbf{h}$ is entirely dependent on the $\mathbf{Z}$ it is fitted to. Hence, in practice we still have to follow the path starting from small μ. By starting the alternating optimization from $\mu = 0$ and following the path slowly, we let the algorithm adjust the initial codes (which are optimal by themselves, because they were obtained by minimizing $E(\mathbf{Z})$ regardless of $\mathbf{h}$) so that they are eventually realizable by a hash function $\mathbf{h} \in \mathcal{H}$. From this point of view, following the path leads us to a better $\mathbf{Z}$ value, setting the stage for this last step (for sufficiently large μ), where the algorithm fits $\mathbf{h}$ with no error and stops.

1.2 Notation and preliminaries

We consider a more general type of objective function than the one in eqs. (1), (2) and (3), because the theoretical development simplifies a bit and is more generally applicable. Consider the optimization problem

$$\min_{\mathbf{h}\in\mathcal{H}} \mathcal{L}(\mathbf{h}) \tag{4}$$

where $\mathbf{h}\colon \mathbb{R}^D \to \mathcal{Z}$ is a function belonging to a set of functions $\mathcal{H}$ (possibly uncountably infinite) and $\mathcal{Z}$ is a finite set. (In binary hashing, $\mathcal{Z} = \{0,1\}^b$, $\mathbf{h}$ is a b-bit hash function and $\mathcal{H}$ could be the set of thresholded linear functions, for example.) We define $\mathcal{L}(\mathbf{h}) = E(\mathbf{h}(\mathbf{X}))$, where $\mathbf{h}(\mathbf{X}) = (\mathbf{h}(\mathbf{x}_1), \dots, \mathbf{h}(\mathbf{x}_N)) \in \mathcal{Z}^N$ and $E(\mathbf{Z})$ is w.l.o.g. a nonnegative function of $\mathbf{Z} = (\mathbf{z}_1, \dots, \mathbf{z}_N) \in \mathcal{Z}^N$. (In binary hashing, $E(\mathbf{Z})$ is an affinity-based loss function as in equation (1), $\mathbf{Z}$ are the $b \times N$ binary codes, or binary embedding, and $\mathbf{h}(\mathbf{X})$ are the $b \times N$ binary codes produced by a hash function $\mathbf{h}$ when applied to a dataset $\mathbf{X}$.)

For each $\mu \geq 0$, we define the penalized objective function

$$\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) = E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h}) \tag{5}$$

where $P(\mathbf{Z}, \mathbf{h})$ is a "penalty" function satisfying $P(\mathbf{Z}, \mathbf{h}) \geq 0 \; \forall \mathbf{Z} \in \mathcal{Z}^N,\ \mathbf{h} \in \mathcal{H}$ and $P(\mathbf{Z}, \mathbf{h}) = 0 \Leftrightarrow \mathbf{Z} = \mathbf{h}(\mathbf{X})$. That is, P penalizes violations of the constraint $\mathbf{Z} = \mathbf{h}(\mathbf{X})$. (In eq. (3) we use a quadratic penalty $P(\mathbf{Z}, \mathbf{h}) = \|\mathbf{Z} - \mathbf{h}(\mathbf{X})\|^2$, equivalent to the Hamming distance between $\mathbf{Z}$ and $\mathbf{h}(\mathbf{X})$. Using a more general penalty P allows us to consider, for example, weighted penalties for each 1-bit hash function h_i.) We will also require the following assumption:

Assumption 1.1. *For each* $\mathbf{Z} \in \mathcal{Z}^N,\ \exists! \mathbf{h}^* \in \mathcal{H}\colon\ P(\mathbf{Z}, \mathbf{h}^*) < P(\mathbf{Z}, \mathbf{h})\ \forall \mathbf{h} \in \mathcal{H},\ \mathbf{h} \neq \mathbf{h}^*$.

This assumption captures the notion that there is a unique best-fit $\mathbf{h}$ of the inputs $\mathbf{X}$ to the binary codes $\mathbf{Z}$. This assumption is not essential but simplifies the development. If ties exist, they could be broken in some arbitrary way.

We will use the following shorthand notation. Let $\mathbf{Z}_1, \dots, \mathbf{Z}_{|\mathcal{Z}|}$ be the $|\mathcal{Z}|$ elements of $\mathcal{Z}$, labeled w.l.o.g. so that $E(\mathbf{Z}_1) \leq \cdots \leq E(\mathbf{Z}_{|\mathcal{Z}|})$, and define the E-error $e_i = E(\mathbf{Z}_i) \geq 0$, the optimal hash function $\mathbf{h}_j = \arg\min_{\mathbf{h}\in\mathcal{H}} P(\mathbf{Z}_j, \mathbf{h})$ (which is well defined by assumption 1.1), and the P-error $p_{ij} = P(\mathbf{Z}_i, \mathbf{h}_j)$, for $i, j = 1, \dots, |\mathcal{Z}|$. Then $0 \leq p_{ii} < p_{ij}$ if $i \neq j$ and $0 \leq e_1 \leq \cdots \leq e_{|\mathcal{Z}|}$. (In binary hashing, e_i is the ith smallest error over all 2^{Nb} binary codes and p_{ij} is the Hamming distance between $\mathbf{Z}_i$ and $\mathbf{h}_j(\mathbf{X})$. Note that we do not require $e_1 < \cdots < e_{|\mathcal{Z}|}$ (i.e., strict inequality), because practical objective functions usually have codes $\mathbf{Z}_i \neq \mathbf{Z}_j$ with $e_i = e_j$ caused by symmetries arising from the use of the Hamming distance in the loss function. For example, flipping all the bits in $\mathbf{Z}$, which corresponds to relabeling the 0s as 1s and vice versa, will not change the loss.)

We begin with two theorems that connect the penalized objective $\mathcal{L}_P$ with the objective $\mathcal{L}$. The first one tells us that we can solve the original problem on $\mathcal{L}$ by solving the problem on $\mathcal{L}_P$ for a finite value of μ ($\mu \geq \mu_\infty$), so $\mathcal{L}_P$ is an *exact penalty function* [11]. The second one tells us that the graph of the minima of $\mathcal{L}_P$ is always below the graph of the minima of $\mathcal{L}$ but they coincide for $\mu \geq \mu_\infty$.

Theorem 1.2. $\mathbf{h} \in \mathcal{H}$ *is a global minimizer of* $\mathcal{L}(\mathbf{h}) \Leftrightarrow (\mathbf{Z}, \mathbf{h})$ *with* $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ *is a global minimizer of* $\mathcal{L}_P$ *as* $\mu \to \infty$.

Proof. This follows from the fact that as $\mu \to \infty$ then the problem "$\min_{\mathbf{Z},\mathbf{h}} \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$" becomes equivalent to "$\min_{\mathbf{Z},\mathbf{h}} E(\mathbf{Z})$ s.t. $\mathbf{Z} = \mathbf{h}(\mathbf{X})$" and to "$\min_{\mathbf{h}} \mathcal{L}(\mathbf{h})$", since $\mathcal{L}(\mathbf{h}) = E(\mathbf{h}(\mathbf{X}))$. □

Theorem 1.3. *For any* $\mathbf{h} \in \mathcal{H}$*:*

$$\min_{\mathbf{Z}\in\mathcal{Z}} \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) \leq \mathcal{L}(\mathbf{h})\ \forall \mu \geq 0, \qquad \min_{\mathbf{Z}\in\mathcal{Z}} \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) = \mathcal{L}(\mathbf{h})\ \forall \mu \geq \mu_\infty.$$

Proof. $\mathcal{L}(\mathbf{h}) = E(\mathbf{h}(\mathbf{X})) = E(\mathbf{h}(\mathbf{X})) + \mu P(\mathbf{h}(\mathbf{X}), \mathbf{h})$ since $P(\mathbf{h}(\mathbf{X}), \mathbf{h}) = 0$. Hence

$$\min_{\mathbf{Z}\in\mathcal{Z}} \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) = \min_{\mathbf{Z}\in\mathcal{Z}} E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h}) \leq E(\mathbf{h}(\mathbf{X})) + \mu P(\mathbf{h}(\mathbf{X}), \mathbf{h}) = \mathcal{L}(\mathbf{h}).$$

The second part follows from $P(\mathbf{Z}, \mathbf{h}) = 0$ and $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ for $\mu \geq \mu_\infty$. □

1.3 Algorithm-free characterization of the optima path

We now characterize the optimal value of $\mathcal{L}_P$ as a function of μ. By this we mean the true global minimum, whether or not this can be found efficiently by an algorithm; section 1.4 considers the case where we use an algorithm to find an approximate minimum of $\mathcal{L}_P$. Call $\mathcal{L}_P^*(\mu) = \min_{\mathbf{Z}\in\mathcal{Z},\mathbf{h}\in\mathcal{H}} \mathcal{L}_P(\mathbf{Z},\mathbf{h};\mu)$ for each $\mu \geq 0$.

Theorem 1.4. *The function $\mathcal{L}_P^*(\mu)$ is nonnegative, nondecreasing, continuous and piecewise linear over $\mu \geq 0$ and it has a finite number of pieces. Specifically,*

$$\mathcal{L}_P^*(\mu) = \begin{cases} \alpha_1 + \mu\beta_1, & \mu \in [0, \mu_1] \\ \alpha_2 + \mu\beta_2, & \mu \in [\mu_1, \mu_2] \\ \dots & \\ \alpha_{K+1} + \mu\beta_{K+1}, & \mu \in [\mu_K, \infty) \end{cases}$$

where $\alpha_i + \mu_i\beta_i = \alpha_{i+1} + \mu_i\beta_{i+1}$ and $\alpha_i \geq 0$ for $i = 1, \dots, K$, and $\beta_1 > \beta_2 > \cdots > \beta_{K+1} = 0$.

Proof. Since $\mathcal{Z}$ is finite and the minimum of $P(\mathbf{Z},\mathbf{h})$ given $\mathbf{Z}$ is unique (by assumption 1.1), we have

$$\mathcal{L}_P^*(\mu) = \min_{\mathbf{Z}\in\mathcal{Z},\mathbf{h}\in\mathcal{H}} \mathcal{L}_P(\mathbf{Z},\mathbf{h};\mu) = \min_{1\leq i\leq|\mathcal{Z}|} (e_i + \mu p_{ii})$$

so $\mathcal{L}_P^*(\mu)$ is the minimum of $|\mathcal{Z}|$ straight lines with nonnegative slopes p_{ii} and nonnegative intercepts e_i in the ordinate axis, for each $\mu \geq 0$ (see fig. 2). Consider the following procedure to construct $\mathcal{L}_P^*(\mu)$. Call $\mu_0 = 0$.

1. Remove non-optima (or redundant optima). For $i = 1, \dots, |\mathcal{Z}|$, remove all $j > i$: $e_i \leq e_j$ and $p_{ii} \leq p_{jj}$. This removes all lines that are lower bounded by some other line in $\mu \geq 0$, and picks the one with lower index in case of ties.
   Relabel the remaining k lines as $1, \dots, k$ so that $e_1 < \cdots < e_k$ and $p_{11} > \cdots > p_{kk}$.

2. Line 1 intersects all other lines at points with positive abscissa μ. Let $\mu_1 > 0$ be the smallest such abscissa. Clearly, line 1 is strictly below all other lines in $[0, \mu_1)$.

The procedure now repeats these two steps for $\mu = \mu_1$ using the set of lines $i = 2, \dots, k$, each having a slope p_{ii} and an ordinate value of $e_i + \mu_1 p_{ii}$ at $\mu = \mu_1$, and continues in this way. It stops when there is only one line left. Call the corresponding μ value μ_K. This produces a sequence of μ values $0 = \mu_0 < \mu_1 < \cdots < \mu_K$ with $0 \leq K < |\mathcal{Z}|$, and a nondecreasing continuous piecewise linear function of the form $e_i + \mu p_{ii}$ within each $[\mu_{i-1}, \mu_i]$, where $e_i + \mu_i p_{ii} = e_{i+1} + \mu_i p_{i+1,i+1}$ and $p_{ii} > p_{i+1,i+1} \geq 0$. It is nondecreasing because each line has a nonnegative slope. □

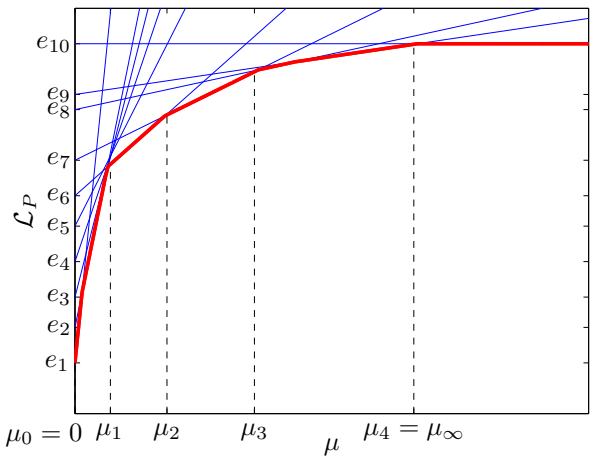

Figure 2: Illustration of the piecewise linear minimum function $\mathcal{L}_P^*(\mu)$ in theorem 1.4.

Corollary 1.5. *Let $(\mathbf{Z}_1, \mathbf{h}_1)$ be a global minimum of $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ for $\mu \to 0^+$ (that is, $\mathbf{Z}_1$ is a global minimum of $E(\mathbf{Z})$ and $\mathbf{h}_1 = \arg\min_{\mathbf{h} \in \mathcal{H}} P(\mathbf{Z}_1, \mathbf{h})$). Then $\exists \mu_1 > 0$: $(\mathbf{Z}_1, \mathbf{h}_1)$ is a global minimizer of $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ for $\mu \leq \mu_1$.*

Corollary 1.6. *$\exists \mu_K > 0$, $\mathbf{Z}_K \in \mathcal{Z}$, $\mathbf{h}_K \in \mathcal{H}$ such that $(\mathbf{Z}_K, \mathbf{h}_K)$ is a global minimizer of $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ $\forall \mu \geq \mu_K$.*

Corollary 1.7. *$\exists \mu \in (0, \infty)$: $\mathbf{h}_\infty \in \mathcal{H}$ is a global minimizer of $\mathcal{L}(\mathbf{h}) \Leftrightarrow (\mathbf{Z}_\infty, \mathbf{h}_\infty)$, where $\mathbf{Z}_\infty = \mathbf{h}_\infty(\mathbf{X})$, is a global minimizer of $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ $\forall \mu \geq \mu_\infty$.*

Proof. It follows from theorems 1.2 and corollary 1.6. □

Theorem 1.8. *Let $\mathbf{Z}_1 \in \mathcal{Z}$ be a global minimizer of $E(\mathbf{Z})$ and $\mathbf{h}_1 = \arg\min_{\mathbf{h} \in \mathcal{H}} P(\mathbf{Z}_1, \mathbf{h})$. If $P(\mathbf{Z}_1, \mathbf{h}) \leq P(\mathbf{Z}, \mathbf{h})$ $\forall \mathbf{Z} \in \mathcal{Z}$, $\forall \mathbf{h} \in \mathcal{H}$ then $\mathbf{h}_1$ is a global minimizer of $\mathcal{L}(\mathbf{h})$ and $(\mathbf{Z}_1, \mathbf{h}_1)$ is a global minimizer of $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ $\forall \mu \geq 0$.*

Proof. $\mathcal{L}_P(\mathbf{Z}_1, \mathbf{h}_1; \mu) = E(\mathbf{Z}_1) + \mu P(\mathbf{Z}_1, \mathbf{h}_1) \leq E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h})$ $\forall \mathbf{Z} \in \mathcal{Z}$, $\forall \mathbf{h} \in \mathcal{H}$, $\forall \mu \geq 0$. The second part follows from theorem 1.2. □

1.4 Algorithm-based characterization of the optima path

We consider approximately minimizing $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ by alternating optimization[1]:

- Over $\mathbf{Z}$ given $\mathbf{h}$: $\min_{\mathbf{Z} \in \mathcal{Z}} \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) = E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h})$. This is a combinatorial optimization problem.
- Over $\mathbf{h}$ given $\mathbf{Z}$: $\min_{\mathbf{h} \in \mathcal{H}} \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) \Leftrightarrow \min_{\mathbf{h} \in \mathcal{H}} P(\mathbf{Z}, \mathbf{h})$. This is a continuous optimization problem.

We make the following assumptions. Assumption 1.9 is practical for several important function classes such as linear SVMs and other classifiers whose training can be done by convex optimization. Assumption 1.10 includes many combinatorial optimization algorithms, in particular alternating optimization over bits (or groups of bits) of $\mathbf{Z}$.

Assumption 1.9. *We have access to an efficient algorithm that can find the global minimizer of $P(\mathbf{Z}, \mathbf{h})$ over $\mathbf{h} \in \mathcal{H}$.*

Assumption 1.10. *We have access to a move-based algorithm for approximately minimizing $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ over $\mathbf{Z}$, initialized from $\mathbf{Z} = \mathbf{Z}_{init}$ (usually the result from the previous MAC iteration, i.e., warm-start). This tries a set of points for $\mathbf{Z}$ that are usually "close" to $\mathbf{Z}_{init}$ and picks the one with lowest $\mathcal{L}_P$ value. The move set always contains $\mathbf{Z}_{init}$ and also $\mathbf{h}(\mathbf{X})$ (since, for large enough μ, this will be the global minimizer). We further assume the algorithm is deterministic given its initialization $\mathbf{Z}_{init}$, i.e., it returns the same result if called with the same μ and $\mathbf{h}$. Hence, this algorithm is guaranteed to decrease or leave unchanged $\mathcal{L}_P$ compared to $\mathbf{Z}_{init}$, although in general it is not guaranteed to find a global minimizer of $\mathcal{L}_P$ over $\mathbf{Z}$. The algorithm can be made exact (i.e., return a global minimizer) by using as move set the entire $\mathcal{Z}$, but this is computationally intractable.*

We use the notation "$\text{MIN}_{\mathbf{Z}}\, \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$" or "$\text{ARG MIN}_{\mathbf{Z}}\, \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$" to indicate an approximate minimization by an algorithm satisfying assumption 1.10, i.e., the output of the algorithm when approximately minimizing $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ over $\mathbf{Z}$ for given $\mathbf{h}$ and μ.

We have the following results.

Theorem 1.11. *For any $\mathbf{h} \in \mathcal{H}$:*

$$\text{MIN}_{\mathbf{Z}}\, \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) \leq \mathcal{L}(\mathbf{h}) \;\forall \mu \geq 0, \qquad \text{MIN}_{\mathbf{Z}}\, \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu) = \mathcal{L}(\mathbf{h}) \;\forall \mu \geq \mu_\infty.$$

Proof. As for theorem 1.3 but taking "$\mathbf{Z} \in$ move set" rather than "$\mathbf{Z} \in \mathcal{Z}$", and noting that $\mathbf{h}(\mathbf{X})$ is in the move set. □

[1] Alternating optimization here does not refer to minimizing $E(\mathbf{Z})$ itself by alternating optimization, but to minimizing $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ alternatingly over $\mathbf{Z}$ and over $\mathbf{h}$, for fixed μ, as in our MAC algorithm.

Theorem 1.12. *There exists a $\mu^* > 0$ such that, for any $\mu \geq \mu^*$, running alternating optimization (first over $\mathbf{h}$, then over $\mathbf{Z}$) of $\mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ from an initial point $\mathbf{Z}_{init}$ stops after one step with $\mathbf{h} = \arg\min_{\mathbf{h}\in\mathcal{H}} P(\mathbf{Z}_{init}, \mathbf{h})$ and $\mathbf{Z} = \mathbf{h}(\mathbf{X})$.*

Proof. The step over $\mathbf{h}$ gives $\mathbf{h} = \arg\min_{\mathbf{h}\in\mathcal{H}} P(\mathbf{Z}_{\text{init}}, \mathbf{h})$. Then, the step over $\mathbf{Z}$ is $\text{MIN}_{\mathbf{Z}}\, E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h})$, which for $\mu \geq \mu^*$ gives as optimal $\mathbf{Z}$ that which minimizes $P(\mathbf{Z}, \mathbf{h})$, i.e., $\mathbf{Z} = \mathbf{h}(\mathbf{X})$, and the algorithm stops. □

Theorem 1.13. *The MAC algorithm (fig. 1) stops when $\mu \geq \mu_\infty$, and this can be achieved in a finite number of iterations.*

Proof. From corollary 1.7, for $\mu \geq \mu_\infty$ a global minimizer of $\mathcal{L}_P$ occurs at $\mathbf{Z} = \mathbf{h}(\mathbf{X})$, and the algorithm always tries $\mathbf{h}(\mathbf{X})$ when optimizing $\mathcal{L}_P$ over $\mathbf{Z}$. Hence, if the MAC algorithm uses a finite sequence of μ values $\mu_0 < \mu_1 < \cdots < \mu_K$ such that $\mu_K \geq \mu_\infty$, it will stop in a finite number of iterations. Such sequence can be generated using an initial value $\mu_0 > 0$ and repeatedly multiplying it times a constant $\alpha > 1$, i.e., $\mu_i = \mu_0 \alpha^i$. □

Theorem 1.14. *For any $\mathbf{h} \in \mathcal{H}$ and $\mu \geq 0$, if $\text{MIN}_{\mathbf{Z}}\, E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h})$ occurs at $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ then the alternating optimization will make no further changes to $\mathbf{Z}$ or $\mathbf{h}$, for any $\mu' \geq \mu$.*

Proof. Assume the $\mathbf{Z}$ step produces $\mathbf{Z} = \mathbf{h}(\mathbf{X})$. We first run an $\mathbf{h}$ step. This does not change $\mathbf{h}$, because $P(\mathbf{h}(\mathbf{X}), \mathbf{h}) = 0$ is a global minimum (regardless of the value of μ). We then run a $\mathbf{Z}$ step. This again returns $\mathbf{Z} = \mathbf{h}(\mathbf{X})$, since the algorithm is deterministic. This also holds for any larger value $\mu' > \mu$, because $E(\mathbf{Z}) + \mu' P(\mathbf{Z}, \mathbf{h}) \geq E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h})\ \forall \mathbf{Z} \in \mathcal{Z}$ and $E(\mathbf{Z}) + \mu' P(\mathbf{Z}, \mathbf{h}) = E(\mathbf{Z}) + \mu P(\mathbf{Z}, \mathbf{h})$ if $\mathbf{Z} = \mathbf{h}(\mathbf{X})$. □

Theorem 1.14 means that whenever $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ after a $\mathbf{Z}$ step (regardless of the value of μ) we can stop because the algorithm will not change $\mathbf{Z}$ or $\mathbf{h}$ anymore, so we can use this as a reliable stopping criterion that is easy to check. However, if $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ occurs after an $\mathbf{h}$ step, the algorithm can still change $\mathbf{Z}$ in the next $\mathbf{Z}$ step. This can happen, for example, if we initialize the MAC algorithm (for small μ) with a $\mathbf{Z}$ for which $\min_{\mathbf{h}\in\mathcal{H}} P(\mathbf{Z}, \mathbf{h}) = 0$ (e.g. if the initial binary codes are linearly separable and $\mathbf{h}$ is a thresholded linear function), but for which $E(\mathbf{Z})$ is large. The following $\mathbf{Z}$ step will pick a different $\mathbf{Z}$ that lowers $E(\mathbf{Z})$ sufficiently to compensate for an increase in $\mu P(\mathbf{Z}, \mathbf{h})$.

Theorem 1.15. *The function $\text{MIN}_{\mathbf{Z},\mathbf{h}}\, \mathcal{L}_P(\mathbf{Z}, \mathbf{h}; \mu)$ (i.e., the approximate minimum value of $\mathcal{L}_P$ obtained by an algorithm satisfying assumptions 1.9 and 1.10) is nonnegative, nondecreasing, continuous and piecewise linear over $\mu \geq 0$ and it has a finite number of pieces, on intervals $0 < \mu'_1 < \cdots < \mu'_\infty < \infty$.*

Proof. The proof is similar to that of theorem 1.4, except that now the total possible number of lines is $|\mathcal{Z}|^2$ rather than $|\mathcal{Z}|$. □

2 Stopping criterion, schedule over μ and path of optimal values

Once $\mathbf{Z} = \mathbf{h}(\mathbf{X})$ after a $\mathbf{Z}$ step (regardless of the value of μ), the MAC algorithm will make no further changes to $\mathbf{Z}$ or $\mathbf{h}$, since then the constraints are satisfied (theorem 1.14). This gives us a reliable stopping criterion that is easy to check, and the MAC algorithm will stop after a finite number of iterations (see below).

Among the theoretical results of section 1, we know that the path of minimizers of $\mathcal{L}_P$ over the continuous penalty parameter $\mu \in [0, \infty)$ is in fact discrete, with changes to $(\mathbf{Z}, \mathbf{h})$ happening only at a finite number of values $0 < \mu_1 < \cdots < \mu_\infty < \infty$. Based on this and on our practical experience, we have found that the following approach leads to good schedules for μ with little effort. We use exponential schedules, of the form $\mu_i = \mu_1 \alpha^{i-1}$ for $i = 1, 2, \ldots$, so the user has to set only two parameters: the initial μ_1 and the multiplier $\alpha > 1$. We choose exponential schedules because typically the algorithm makes most progress at the beginning, and it is important to track a good minimum there. The upper value μ_∞ past which no changes occur will be reached by our exponential schedule in a finite number of iterations, and our stopping criterion will detect that. We set the multiplier to a value $1 < \alpha < 2$ that is as small as computationally convenient. If α is too small, the algorithm will take many iterations, some of which may not even change $\mathbf{Z}$ or $\mathbf{h}$ (because the path of minima is discrete). If α is too big, the algorithm will reach too quickly a stopping point, without having had time to find a better minimum. As for the initial μ_1, we estimate it by trying values (exponentially spaced) until we find a μ for which changes to $\mathbf{Z}$ from its initial value (for $\mu = 0$) start to occur. (It is also possible to find lower and upper bounds for μ_1 and μ_∞, respectively, for a particular loss function, such as KSH, eSPH or EE.) Overall, the computational time required to estimate μ_1 and α is comparable to running a few extra iterations of the MAC algorithm.

Finally, in practice we use a form of early stopping in order to improve generalization. We use a small validation set to evaluate the precision achieved by the hash function $\mathbf{h}$ along the MAC optimization. If the precision decreases over that of the previous step, we ignore the step and skip to the next value of μ. Besides helping to avoid overfitting, this saves computation, by avoid such extra optimization steps. Since the validation set is small, it provides a noisy estimate of the generalization ability at the current iterate, and this occasionally leads to skipping a valid μ value. This is not a problem because the next μ value, which is close to the one we skipped, will likely work. At some point during the MAC optimization, we do reach an overfitting region and the precision stops increasing, so the algorithm will skip all remaining μ values until it stops. In summary, using this validation procedure guarantees that the precision (in the validation set) is greater or equal than that of the initial $\mathbf{Z}$, thus resulting in a better hash function.

3 Z step of the MAC algorithm

In the **Z** step of our algorithm, the goal is to minimize the objective function in eq. (3) over the binary codes $\mathbf{z}_n \in \{0,1\}^b$ given the hash function $\mathbf{h}$. It is an NP-complete problem in Nb binary variables. As mentioned in the paper, two recent works have proposed practical approaches for this problem based on alternating optimization: a quadratic surrogate method [8], and a GraphCut method [9]. In both methods, the starting point is to apply alternating optimization over the ith bit of all points given the remaining bits are fixed for all points (for $i = 1, \dots, b$), and to solve the optimization over the ith bit approximately.

In the main paper [12], we explained briefly our modification to these methods. Here, we give a detailed explanation. We start by describing each method in their original form (which applies to the loss function over binary codes, i.e., the first term in $\mathcal{L}_P$), and then we give our modification to make it work with our **Z** step objective (the regularized loss function over binary codes, i.e., the complete $\mathcal{L}_P$).

3.1 Solution using a quadratic surrogate method [8]

This is based on the fact that any loss function that depends on the Hamming distance of two binary variables can be equivalently written as a quadratic function of those two binary variables [8]. Since this is the case for every term $L(\mathbf{z}_n, \mathbf{z}_m;\ y_{nm})$ (because only the ith bit in each of $\mathbf{z}_n$ and $\mathbf{z}_m$ is free), we can write the first term in $\mathcal{L}_P$ as a binary quadratic problem. We now consider the second term (on μ) as well. (We use a similar notation as that of [8].) The optimization for the ith bit can be written as:

$$\min_{\mathbf{z}_{(i)}} \sum_{n,m=1}^{N} l_i(z_{ni}, z_{mi}) + \mu \sum_{n=1}^{N} (z_{ni} - h_i(\mathbf{x}_n))^2 \tag{6}$$

where $l_i = L(z_{ni}, z_{mi}, \bar{\mathbf{z}}_n, \bar{\mathbf{z}}_m; y_{nm})$ is the loss function defined on the ith bit, z_{ni} is the ith bit of the nth point, $\bar{\mathbf{z}}_n$ is a vector containing the binary codes of the nth point except the ith bit, and $h_i(\mathbf{x}_n)$ is the ith bit of the binary code of the nth point generated by the hash function $\mathbf{h}$. Lin et al. [8] show that $l(z_1, z_2)$ can be replaced by a binary quadratic function

$$l(z_1, z_2) = \frac{1}{2} z_1 z_2 \big(l^{(11)} - l^{(-11)}\big) + \text{constant} \tag{7}$$

as long as $l(1,1) = l(-1,-1) = l^{(11)}$ and $l(1,-1) = l(-1,1) = l^{(-11)}$, where $z_1, z_2 \in \{-1, 1\}$. Equation (7) helps us to rewrite the optimization (6) as the following:

$$\min_{\mathbf{z}_{(i)}} \sum_{n,m=1}^{N} \frac{1}{2} z_{ni} z_{mi} \big(l^{(11)} - l^{(-11)}\big) + \mu \sum_{n=1}^{N} (z_{ni} - h_i(\mathbf{x}_n))^2.$$

By defining $a_{nm} = \big(l^{(11)}_{(inm)} - l^{(-11)}_{(inm)}\big)$ as the (n,m) element of a matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$ and ignoring the coefficients, we have the following optimization problem:

$$\min_{\mathbf{z}_{(i)}} \mathbf{z}_{(i)}^T \mathbf{A} \mathbf{z}_{(i)} + \mu \left\| \mathbf{z}_{(i)} - \mathbf{h}_i(\mathbf{X}) \right\|^2 \quad \text{s.t.} \quad \mathbf{z}_{(i)} \in \{-1, +1\}^N$$

where $\mathbf{h}_i(\mathbf{X}) = (h_i(\mathbf{x}_1), \dots, h_i(\mathbf{x}_N))^T$ is a vector of length N (one bit per data point). Both terms in the above minimization are quadratic on binary variables. This is still an NP-complete problem (except in special cases), and we approximate it by relaxing it to a continuous quadratic program (QP) over $\mathbf{z}_{(i)} \in [-1, 1]^N$ and binarizing its solution. In general, the matrix $\mathbf{A}$ is not positive definite and the relaxed QP is not convex, so we need an initialization. (However, the term on μ adds $\mu\mathbf{I}$ to $\mathbf{A}$, so even if $\mathbf{A}$ is not positive definite, $\mathbf{A} + \mu\mathbf{I}$ will be positive definite for large enough μ, and the QP will be convex.) We construct an initialization by converting the binary QP into a binary eigenproblem:

$$\min_{\boldsymbol{\alpha}} \boldsymbol{\alpha}^T \mathbf{B} \boldsymbol{\alpha} \quad \text{s.t.} \quad \alpha_0 = 1, \quad \mathbf{z}_{(i)} \in \{-1, 1\}^N, \boldsymbol{\alpha} = \begin{pmatrix} \mathbf{z}_{(i)} \\ \alpha_0 \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} \mathbf{A} & -\frac{\mu}{2}\mathbf{h}_i(\mathbf{X}) \\ -\frac{\mu}{2}\mathbf{h}_i(\mathbf{X})^T & 0 \end{pmatrix}.$$

To solve this problem we use spectral relaxation, where the constraints $\mathbf{z}_{(i)} \in \{-1, +1\}^N$ and $z_{i+1} = 1$ are relaxed to $\|\boldsymbol{\alpha}\| = N + 1$. The solution to this problem is the eigenvector corresponding to the smallest

eigenvalue of $\mathbf{B}$. We use the truncated eigenvector as the initialization for minimizing the relaxed, bound-constrained QP:

$$\min_{\mathbf{z}_{(i)}} \mathbf{z}_{(i)}^T \mathbf{A} \mathbf{z}_{(i)} + \mu \left\| \mathbf{z}_{(i)} - \mathbf{h}_i(\mathbf{X}) \right\|^2 \text{ s.t. } \mathbf{z}_{(i)} \in [-1, 1]^N$$

which we solve using L-BFGS-B [14].

As noted above, the $\mathbf{Z}$ step is an NP-complete problem in general, so we cannot expect to find the global optimum. It is even possible that the approximate solution could increase the objective over the previous iteration's $\mathbf{Z}$ (this is likely to happen as the overall MAC algorithm converges). If that occurs, we simply skip the update, in order to guarantee that we decrease monotonically on $\mathcal{L}_P$, and avoid oscillating around a minimum.

3.2 Solution using a GraphCut algorithm [9]

To optimize over the ith bit (given all the other bits are fixed), we have to minimize eq. (6). In general, this is an NP-complete problem over N bits (the ith bit for each image), with the form of a quadratic function on binary variables. We can apply the GraphCut algorithm [1, 2, 7], as proposed by the FastHash algorithm of Lin et al. [9]. This proceeds as follows. First, we assign all the data points to different, possibly overlapping groups (blocks). Then, we minimize the objective function over the binary codes of the same block, while all the other binary codes are fixed, then proceed with the next block, etc. (that is, we do alternating optimization of the bits over the blocks). Specifically, to optimize over the bits in block $\mathcal{B}$, we define $a_{nm} = \left(l_{(inm)}^{(11)} - l_{(inm)}^{(-11)}\right)$ and, ignoring the constants, we can rewrite equation (6) as:

$$\min_{\mathbf{z}_{(i,\mathcal{B})}} \sum_{n,m \in \mathcal{B}} a_{nm} z_{ni} z_{mi} + 2 \sum_{n \in \mathcal{B}, m \notin \mathcal{B}} a_{nm} z_{ni} z_{mi} - \mu \sum_{n \in \mathcal{B}} z_{ni} h_i(\mathbf{x}_n).$$

We then rewrite this equation in the standard form for the GraphCut algorithm:

$$\min_{\mathbf{z}_{(i,\mathcal{B})}} \sum_{n \in \mathcal{B}} \sum_{m \in \mathcal{B}} v_{nm} z_{ni} z_{mi} + \sum_{n \in \mathcal{B}} u_{nm} z_{ni}$$

where $v_{nm} = a_{nm}$, $u_{nm} = 2 \sum_{m \notin \mathcal{B}} a_{nm} z_{mi} - \mu h_i(\mathbf{x}_n)$. To minimize the objective function using the GraphCut algorithm, the blocks have to define a submodular function. For the objective functions that we explained in the paper, this can be easily achieved by putting points with the same label in one block (Lin et al. [9] give a simple proof of this).

Unlike in the quadratic surrogate method, using the GraphCut algorithm with alternating optimization on blocks defining submodular functions is guaranteed to find a $\mathbf{Z}$ that has a lower or equal objective value that the initial one, and therefore to decrease monotonically $\mathcal{L}_P$.

4 Experiments

4.1 Supervised datasets

Figures 3– 5 amplify the figures in the main paper [12] with results with different numbers of bits b, precision/recall curves, and training on centered and normalized vectors.

4.2 Unsupervised dataset

Although affinity-based hashing is intended to work with supervised datasets, it can also be used with unsupervised ones, and our MAC approach applies just as well. We use the SIFT1M dataset [6], which contains $N = 1\,000\,000$ training high-resolution color images and $10\,000$ test images, each represented by $D = 128$ SIFT features. The experiments and conclusions are generally the same as with supervised datasets, with small differences in the settings of the experiments. In order to construct an affinity-based objective function, we define neighbors as follows. For each point in the training set we use the $\kappa_+ = 100$ nearest neighbors as positive (similar) neighbors, and $\kappa_- = 500$ points chosen randomly among the remaining points as negative (dissimilar) neighbors. We report precision and precision/recall for the test set queries using as ground truth (set of true neighbors in original space) the K nearest neighbors in unsupervised datasets, and all the training points with the same label in supervised datasets.

Fig. 6 shows results using KSH and eSLPH loss functions, respectively, with different sizes of retrieved neighbor sets and using 8 to 32 bits. As with the supervised datasets, it is clear that the MAC algorithm finds better optima and that *MACcut* is generally better than *MACquad*.

Fig. 8 shows results comparing with binary hashing methods. All methods are trained on a subset of $5\,000$ points. We consider two types of methods. In the first type, we create pseudolabels for each point and then apply supervised methods as in CIFAR (in particular, *cut*/*quad* and *MACcut*/*MACquad*, using the KSH loss function). The pseudolabels y_{nm} for each training point $\mathbf{x}_n$ are obtained by declaring as similar points its $\kappa_+ = 100$ true nearest neighbors and as dissimilar points a random subset of $\kappa_- = 500$ points among the remaining points. In the second type, we use purely unsupervised methods (not based on similar/dissimilar affinities): thresholded PCA (tPCA), Iterative Quantization (ITQ) [4], Binary Autoencoder (BA) [3], Spectral Hashing (SH) [13], AnchorGraph Hashing (AGH) [10], and Spherical Hashing (SPH) [5]. The results are again in general agreement with the conclusions in the main paper.

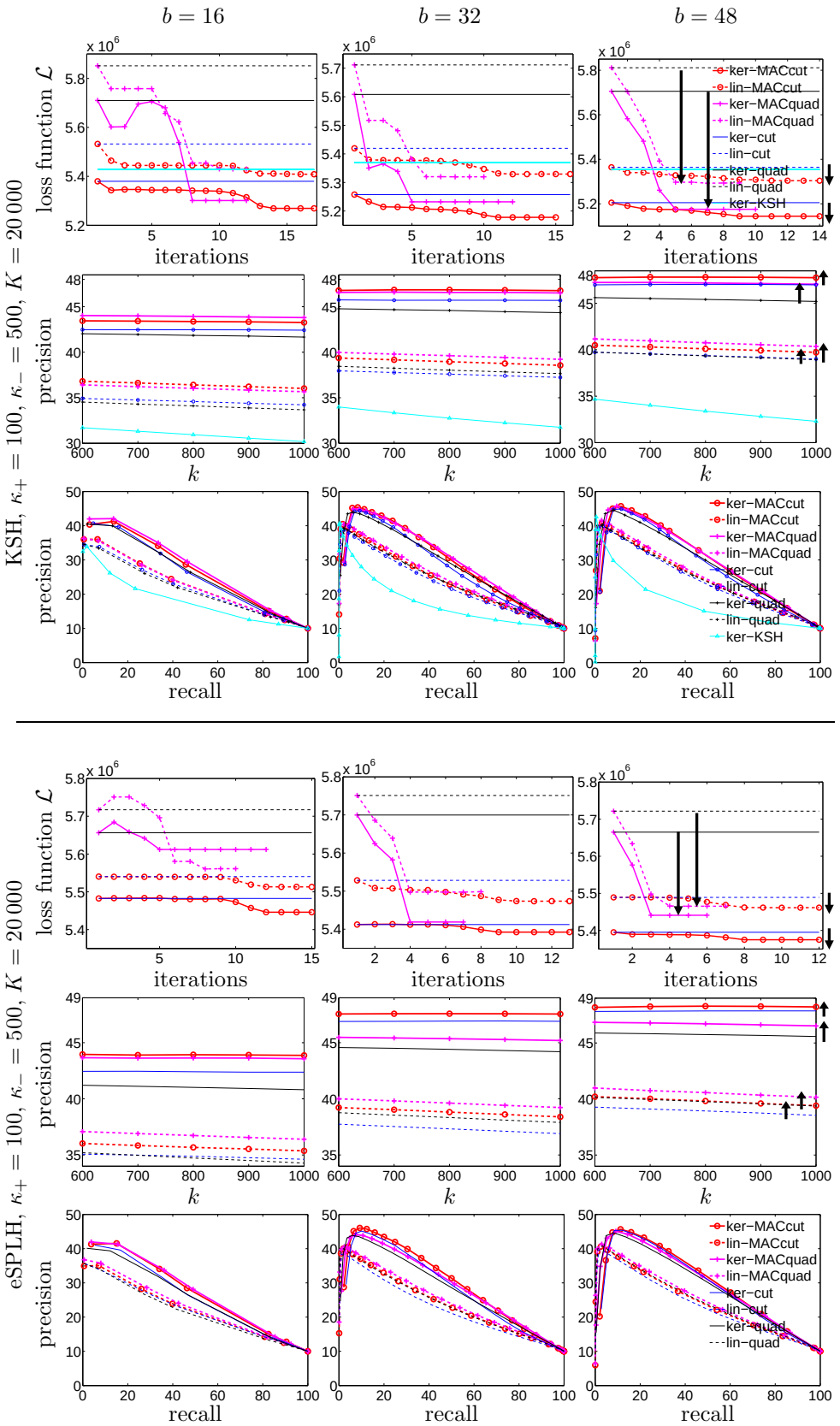

Figure 3: KSH (top panel) and eSPLH (bottom panel) loss functions on CIFAR dataset, using $b = 16$ to 48 bits. The rows in each panel show the value of the loss function $\mathcal{L}$, the precision for k retrieved points and the precision/recall (at different Hamming distances).

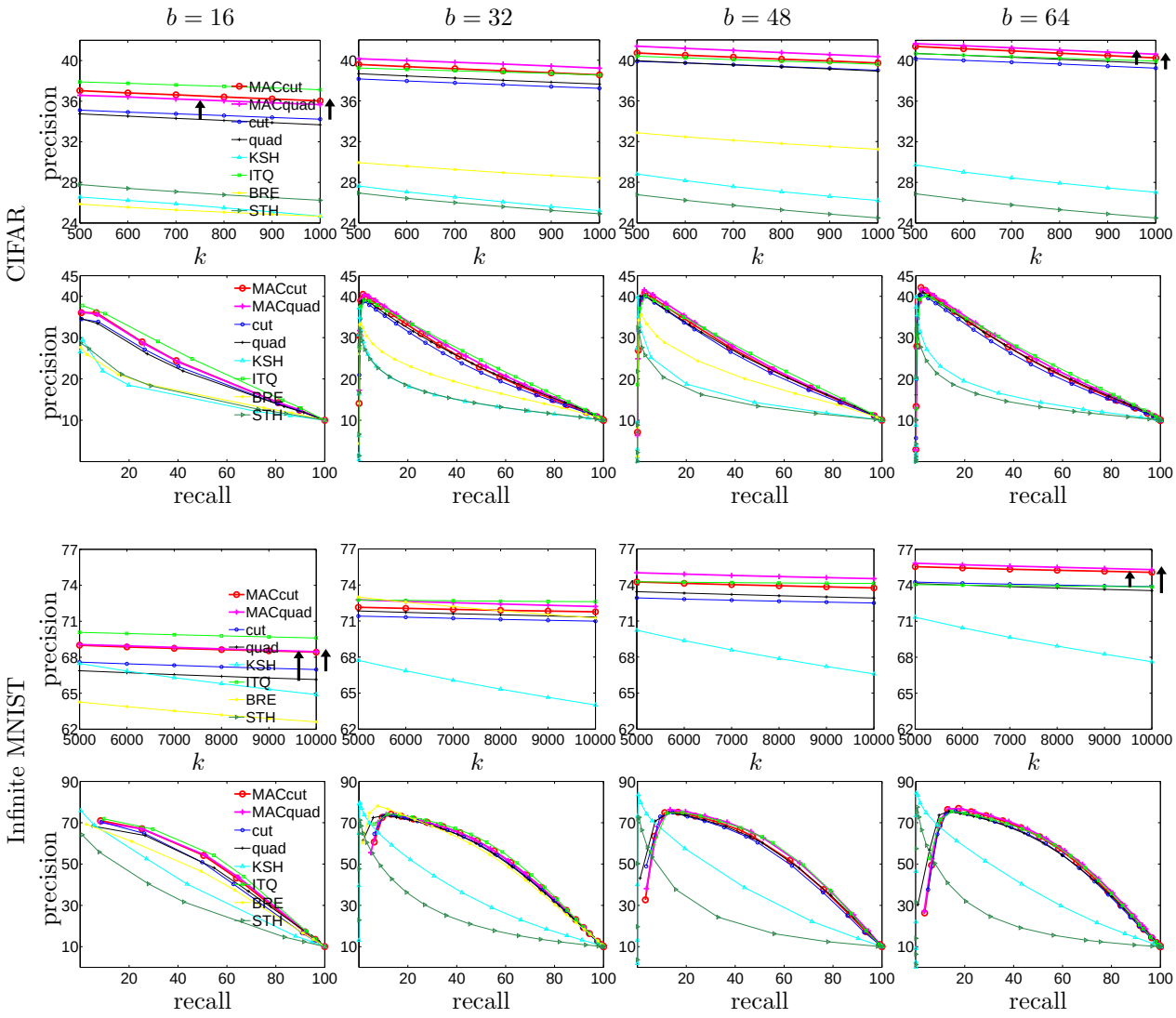

Figure 4: Comparison with binary hashing methods on CIFAR (top panel) and Infinite MNIST (bottom panel), using a linear hash function, using $b = 16$ to 64 bits. The rows in each panel show the precision for k retrieved points, for a range of k, and the precision/recall at different Hamming distances.

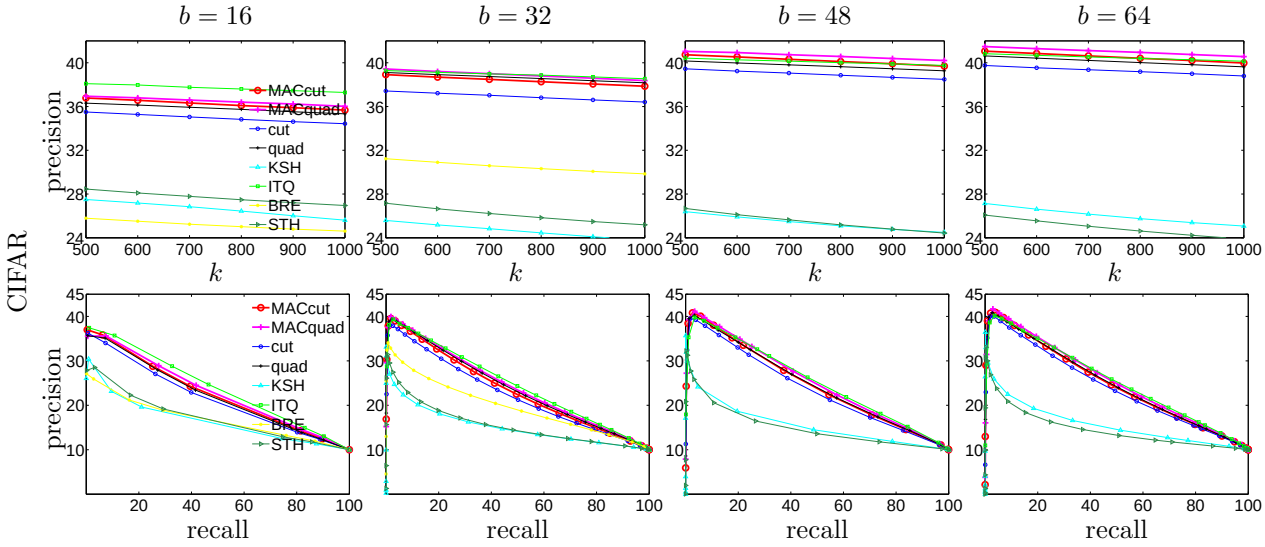

Figure 5: As in fig. 4 but using the cosine similarity instead of the Euclidean distance to find neighbors (i.e., all the points are centered and normalized before training and testing), on CIFAR.

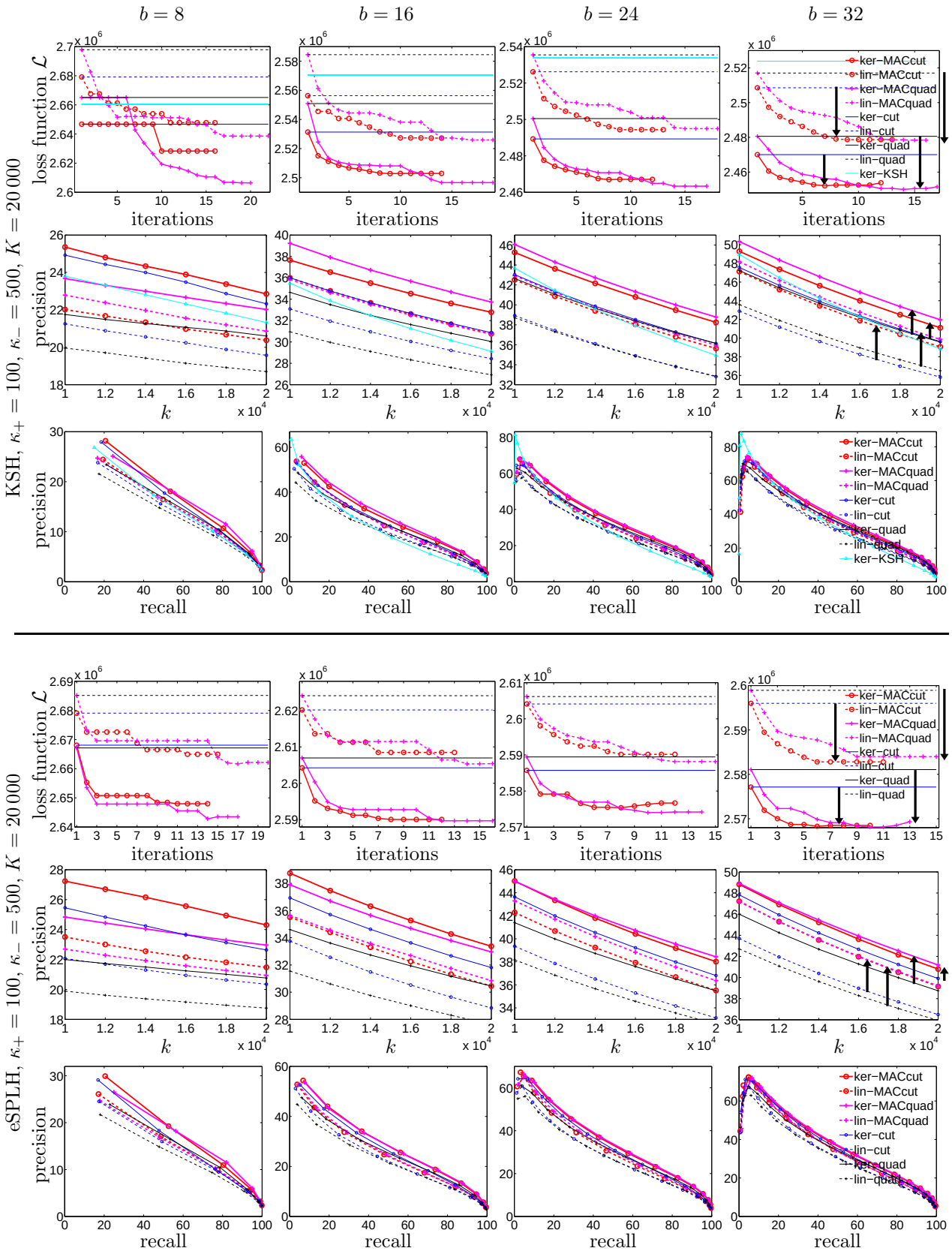

Figure 6: Like fig. 3 but on SIFT1M dataset, for the KSH (top panel) and eSPLH (bottom panel) loss functions. The rows show the value of the loss function $\mathcal{L}$, the precision (for a number of retrieved points k) and the precision/recall (at different Hamming distances), using $b = 8$ to 32 bits.

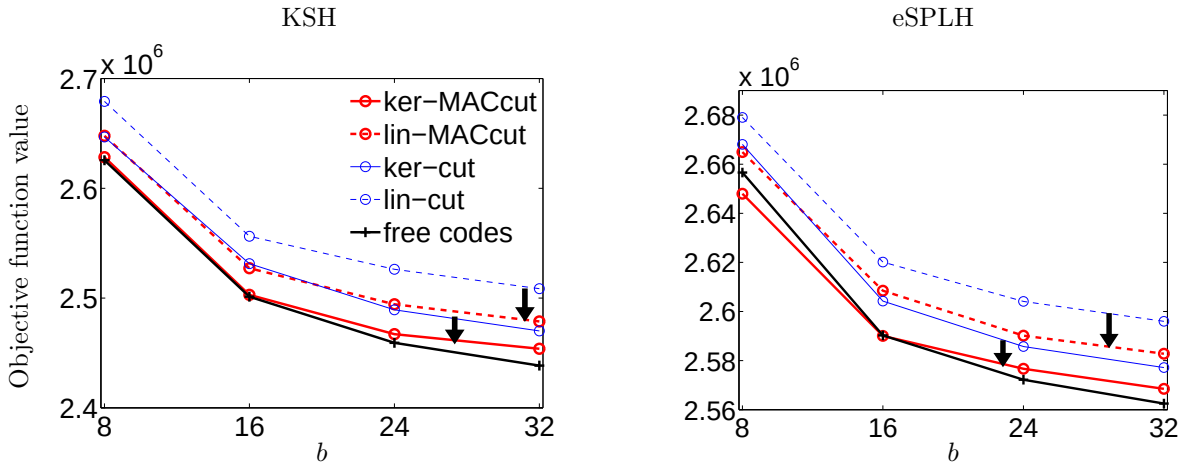

Figure 7: Like fig. 6 for the SIFT1M dataset, but showing the value of the error function $E(\mathbf{Z})$ for the "free" binary codes, and for the codes produced by the hash functions learned by *cut* (the two-step method) and *MACcut*, with linear and kernel hash functions.

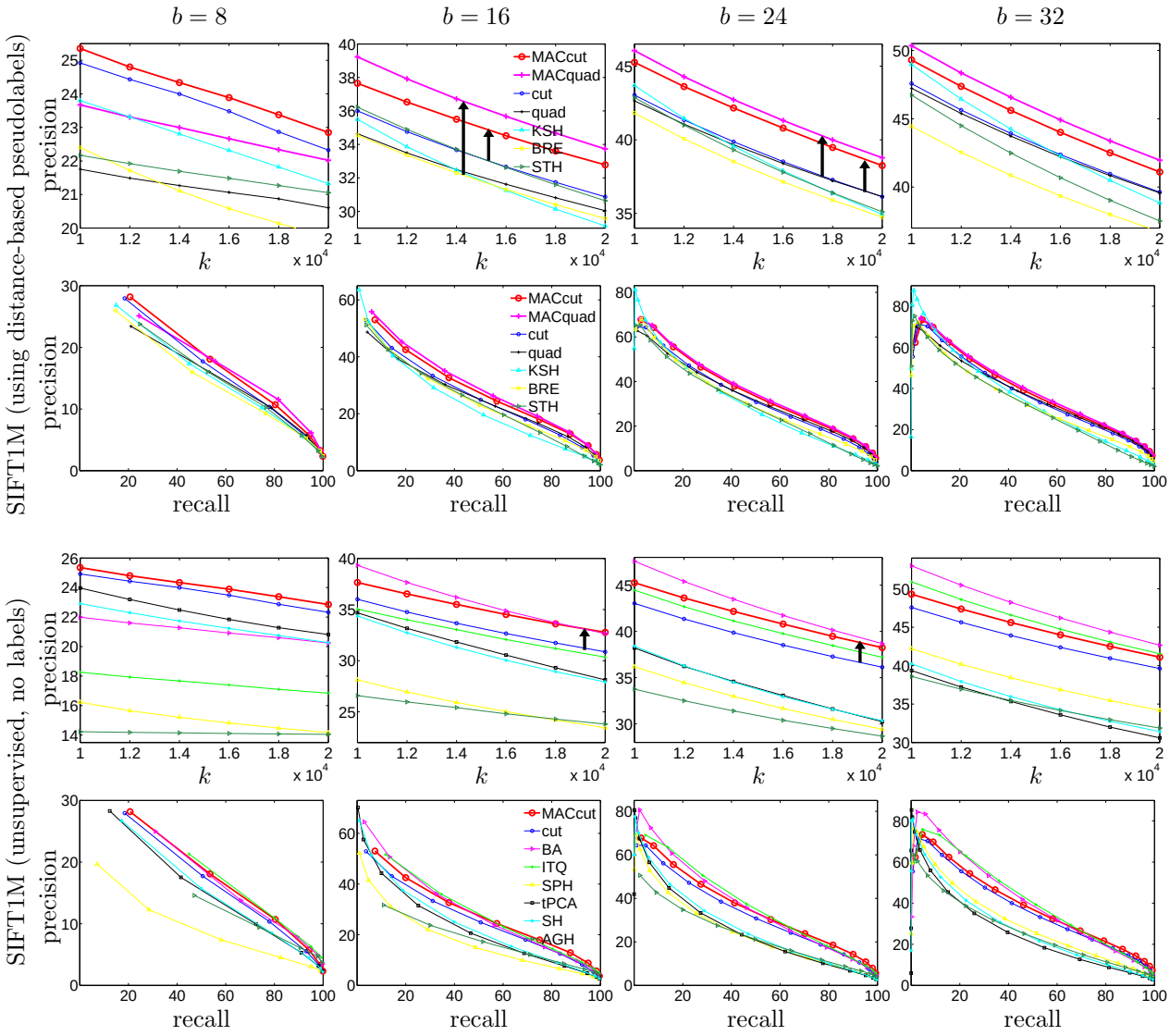

Figure 8: Comparison with binary hashing methods on SIFT1M using pseudolabels (top panel) and without labels (bottom panel). The rows in each panel show the precision (for a range of retrieved points k) and the precision/recall (at different Hamming distances), using $b = 8$ to 32 bits.

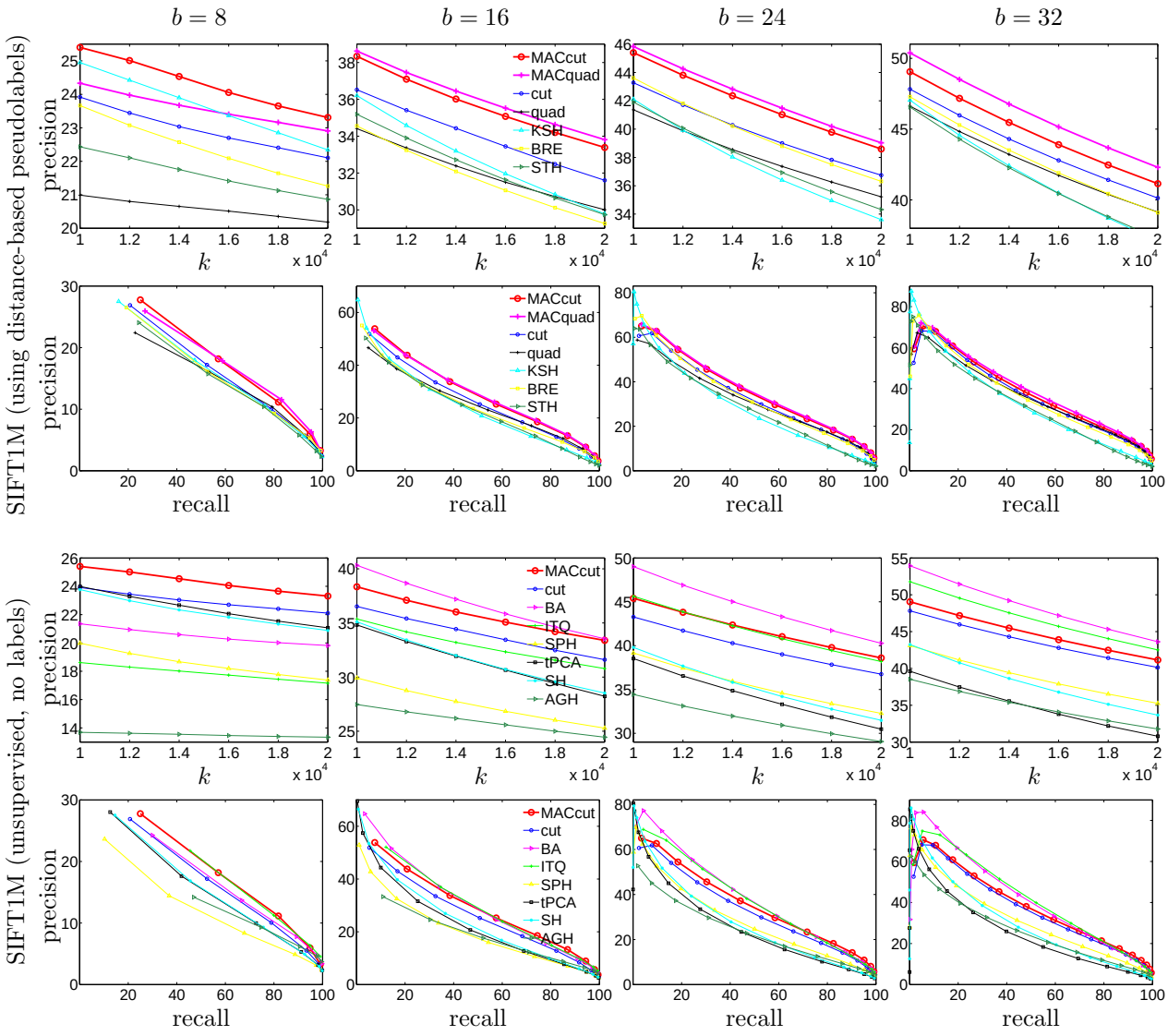

Figure 9: As in fig. 8 but using the cosine similarity instead of the Euclidean distance to find neighbors (i.e., all the points are centered and normalized before training and testing), on SIFT1M.

References

[1] Y. Boykov and V. Kolmogorov. Computing geodesics and minimal surfaces via graph cuts. In *Proc. 9th Int. Conf. Computer Vision (ICCV'03)*, pages 26–33, Nice, France, Oct. 14–17 2003.

[2] Y. Boykov and V. Kolmogorov. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, Sept. 2004.

[3] M. Á. Carreira-Perpiñán and R. Raziperchikolaei. Hashing with binary autoencoders. In *Proc. of the 2015 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'15)*, pages 557–566, Boston, MA, June 7–12 2015.

[4] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A Procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 35(12):2916–2929, Dec. 2013.

[5] J.-P. Heo, Y. Lee, J. He, S.-F. Chang, and S.-E. Yoon. Spherical hashing. In *Proc. of the 2012 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'12)*, pages 2957–2964, Providence, RI, June 16–21 2012.

[6] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 33(1):117–128, Jan. 2011.

[7] V. Kolmogorov and R. Zabih. What energy functions can be minimized via graph cuts? *IEEE Trans. Pattern Analysis and Machine Intelligence*, 26(2):147–159, Feb. 2003.

[8] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *Proc. 14th Int. Conf. Computer Vision (ICCV'13)*, pages 2552–2559, Sydney, Australia, Dec. 1–8 2013.

[9] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proc. of the 2014 IEEE Computer Society Conf. Computer Vision and Pattern Recognition (CVPR'14)*, pages 1971–1978, Columbus, OH, June 23–28 2014.

[10] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In L. Getoor and T. Scheffer, editors, *Proc. of the 28th Int. Conf. Machine Learning (ICML 2011)*, pages 1–8, Bellevue, WA, June 28 – July 2 2011.

[11] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag, New York, second edition, 2006.

[12] R. Raziperchikolaei and M. Á. Carreira-Perpiñán. Optimizing affinity-based binary hashing using auxiliary coordinates. In *Advances in Neural Information Processing Systems (NIPS)*, volume 29. MIT Press, Cambridge, MA, 2016.

[13] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In D. Koller, Y. Bengio, D. Schuurmans, L. Bottou, and A. Culotta, editors, *Advances in Neural Information Processing Systems (NIPS)*, volume 21, pages 1753–1760. MIT Press, Cambridge, MA, 2009.

[14] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: FORTRAN subroutines for large-scale bound-constrained optimization. *ACM Trans. Mathematical Software*, 23(4):550–560, Dec. 1997.