Improving Kernel Locality-Sensitive Hashing Using Pre-Images and Bounds

Zalán Bodó
Department of Computer Science
Babeş–Bolyai University
RO-400084 Cluj-Napoca, Romania
Email: zbodo@cs.ubbcluj.ro

Lehel Csató
Department of Computer Science
Babeş–Bolyai University
RO-400084 Cluj-Napoca, Romania
Email: lehel.csato@cs.ubbcluj.ro

***Abstract*—Large databases become more and more common in (supervised) learning scenarios, containing hundred thousands or even millions of training examples. Finding the k-nearest neighbors (k-NN) of a point from a dataset, however, requires to compare the point to every training example. Locality-sensitive hashing (LSH) [11], [7], [3] hashes the dataset into buckets such that, with high probability, similar examples are grouped together, thus providing a sub-linear search time for neighbors. However, linear k-NN is sometimes not enough; the Euclidean distance does not always capture important data properties, therefore kernels are used to map data into a – possibly higher dimensional – feature space and perform the k-NN search there. To kernelize the LSH from [3], the most important question to be answered is how to generate random normally distributed vectors in the feature space. In this paper we present an improved kernel LSH technique, a modified version of the kLSH algorithm proposed in [12]. We compute the pre-images of the random feature space vectors to save important computational resources. Our proposal of pre-image calculation is interesting, because no *additional* intrinsic computations are required. Furthermore, for positive definite kernel functions we propose two inequalities to speed up searching.**

I. Introduction

The k-nearest neighbor classification was introduced by Cover and Hart [4], and since then it is one of the most successful classification methods in data mining.[1] By its simplicity and ease of implementation it is used in a wide variety of applications from text categorization to image retrieval. But its simplicity and lack of explicit training phase comes with a drawback: for predicting the label of one unseen example the entire training set has to be parsed, which – for large datasets – implies a huge number of distance calculations, which is extremely time consuming.

In the last years several algorithms were proposed for fast approximate nearest neighbor search. One such technique is based on locality-sensitive hashing (LSH) [11]. LSH partitions the training dataset into clusters such that similar points are grouped together. Although this seems an ordinary clustering method, it is not: in this case the *clustering* is realized without comparisons between data points. The points are hashed such that similar points are put in *nearby* buckets. For example in [3] hashing is based on computing the dot products of the data points with randomly generated normally distributed vectors.

Other LSH methods are based on p-stable distributions [5], spectral hashing [20], semantic hashing [17] etc., but unfortunately, most of them operate in the L^2 or L^1 or in a fixed feature space. One interesting kernelized variant using random Fourier features was proposed in [15], but the method supports shift invariant kernels only (e.g. Gaussian kernel). In a recent article [12] the method of [3] was extended to support arbitrary kernels to perform k-NN search in high dimensional feature spaces. However, to calculate a dot product in the feature space p kernel computations are to be performed, where p is the size of a smaller dataset sampled without replacement from the large initial dataset. In this paper we present an improved kernel LSH technique, a modified version of the kernelized LSH (kLSH) algorithm proposed in [12]. We compute the pre-images of the random feature space vectors to save important computational resources. Furthermore, for positive definite kernel functions we propose two inequalities to speed up the searching procedure.

The paper is structured as follows: Section II introduces notational conventions used in the paper. Section III presents the k-nearest neighbor search and its kernelization, after which in Section IV a slightly modified version of k-NN is presented with two inequalities that can reduce the number of distance/kernel computations. Section V introduces locality-sensitive hashing and its kernelized variant. Our method is described in Section VI with experimental results presented in Section VII. Section VIII concludes the paper.

II. Notation

We denote with $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \ldots, \ell\}$ the training data, $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$, $y_i \in \mathcal{Y}$. We use the scalar k to denote the number of nearest neighbors, but also to denote the kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$; from the context it will be clear to which it refers. The scalar L denotes an important parameter of the algorithm: the number of random vectors generated in the feature space.

K denotes the number of classes in the supervised learning problem and $\mathbf{K}$ is the Gram matrix of data points. By $\mathbf{k}_{\mathbf{x}_i}$ we refer to the column of $\mathbf{K}$ corresponding to $\mathbf{x}_i$.

[1] In 2006 k-NN was voted as the 7th most influential algorithm in data mining [22] (http://www.cs.uvm.edu/~icdm/algorithms/index.shtml).

In general, boldface lowercase letters denote vectors, boldface capitals are matrices, calligraphic capitals are sets, while all other variables denote scalars. The matrix transpose of $\mathbf{A}$ is denoted by $\mathbf{A}'$. For a matrix $\mathbf{A}$, $\mathbf{A}_{i\cdot}$ and $\mathbf{A}_{\cdot j}$ denote the vectors corresponding to the i-th row and j-th column, respectively. The dot (or inner) product of vectors $\mathbf{x}$ and $\mathbf{z}$ is denoted by $\mathbf{x}'\mathbf{z}$.

Throughout the paper $\|\cdot\|$ denotes the Euclidean (L_2) distance.

III. k-NEAREST NEIGHBORS

The k-NN classifier determines the label of an unseen point $\mathbf{x}$ by simple majority voting: finds the k-nearest neighbors of $\mathbf{x}$ and assigns to it the winning label among these,

$$\arg \max_{c=1,2,\ldots,K} \sum_{\mathbf{z}\in N_k(\mathbf{x}),(\mathbf{z},f(\mathbf{z}))\in\mathcal{D}} sim(\mathbf{z},\mathbf{x}) \cdot \delta(c, f(\mathbf{z})) \ ,$$

where the function f returns the label of a point, $N_k(\mathbf{x})$ denotes the set of k-nearest neighbors of $\mathbf{x}$ from the training data, K is the number of classes. The function $sim(\cdot,\cdot)$ returns the similarity of two examples, and δ is the Kronecker delta function, $\delta(a,b) = 1$ if $a = b$, 0 otherwise. The function $sim(\cdot,\cdot)$ is used to give different weights for different points. One choice could be to use some distance metric $d(\cdot,\cdot)$ with the property of assigning a lower value to nearby points and a higher value to farther points to $\mathbf{x}$. Then one can choose for example

$$sim(\mathbf{x},\mathbf{z}) = \frac{1}{d(\mathbf{x},\mathbf{z})+1} \ .$$

If the constant function $sim(\mathbf{x},\mathbf{z}) = 1$ is chosen, we arrive to simple k-NN, where all the neighbors have the same influence on the predicted label. In the following we consider this variant.

In order to efficiently implement the k-NN method, no explicit form of the inductive classifier is built, since representing and storing the decision boundaries can become very complex.

The k-nearest neighbor algorithm determines labels based on the labels of the nearest points. *Nearest* is defined using some metric – most of the time using the Euclidean distance. It is known that the squared Euclidean distance can be rewritten with dot products as

$$\|\mathbf{x}-\mathbf{z}\|^2 = \mathbf{x}'\mathbf{x} + \mathbf{z}'\mathbf{z} - 2\mathbf{x}'\mathbf{z} \ .$$

Applying the *kernel trick* [18], the dot products can be replaced by an arbitrary positive definite kernel. Hence, points are implicitly mapped to a – possibly higher dimensional – space, where their dot product is given by the kernel function $k(\cdot,\cdot)$.

IV. DISTANCE INEQUALITIES

Since k-NN operates using distances or equivalently dot products, the properties of distance metrics and positive definite kernels can be exploited. Namely, we will use the definition of positive definite kernels and the triangle inequality to eliminate a considerable number of comparisons in the testing phase.

Algorithm 1 Modified k-nearest neighbor search

```
1:  i ← 0
2:  dmax ← −1
3:  imax ← −1
4:  for every training point z_i do
5:      if i < k then
6:          Compute/store d(x, z_i) (and index) in a list.
7:          if distance > dmax then
8:              dmax ← d(x, z_i)
9:              imax ← index of dmax in the list
10:         end if
11:         i ← i + 1
12:         Go to step 4.
13:     end if
14:     Compute distance d(x, z_i).
15:     if distance < dmax then
16:         Store the distance (and index) in the list at imax.
17:         Find new dmax and imax in the list.
18:     end if
19:     i ← i + 1
20: end for
```

The following two inequalities are to be used:

$$d(\mathbf{x},\mathbf{z}) \geq k(\mathbf{x},\mathbf{x}) + k(\mathbf{z},\mathbf{z}) - 2\sqrt{k(\mathbf{x},\mathbf{x})k(\mathbf{z},\mathbf{z})} \ , \quad (1)$$

$$d(\mathbf{x},\mathbf{z}) \geq \frac{1}{2}(k(\mathbf{x},\mathbf{x}) - k(\mathbf{z},\mathbf{z})) \ . \quad (2)$$

Both formulae can be used, however, we will show that (2) offers a *better solution*, since most of the time it gives a lower upper bound on the distance $d(\mathbf{x},\mathbf{z})$. It is clear, that for translation invariant and normalized kernels the formulae are ineffectual.

In the training phase we store the data along with their squared norm in the feature space, that is $k(\mathbf{z},\mathbf{z})$. Similarly, at testing, $k(\mathbf{x},\mathbf{x})$ is computed for every test point $\mathbf{x}$ when it is encountered. Most k-NN implementations store all the distances, and when computations are over, this list of (distance, index) pairs is sorted in increasing order by distance, from which the first k neighbors are obtained. Here a slightly modified k-NN search is proposed, which maintains a list of k (distance, index) pairs and stores the smallest distances in it. Since k is usually small – e.g. between 1 and 11 – this method is inherently faster then the one mentioned previously. The algorithm proceeds as follows: if the list is not full, the (distance, index) pair is stored on the next available position and $dmax$ is updated to contain the maximal distance value encountered so far. If it is full, we perform the following steps: check for whether $\frac{1}{2}(k(\mathbf{x},\mathbf{x}) - k(\mathbf{z},\mathbf{z})) \geq dmax$. If it evaluates to true, then skip and move on to the next training example. If $dmax$ is less, then store the distance in the list on the position of $dmax$; go through the list and find the new maximum $dmax$ and its index.

Algorithm 1 shows the pseudo-code of this modified k-NN search procedure for a point $\mathbf{x}$. However, we left out the computation of $k(\mathbf{x},\mathbf{x})$ for the training data, and similarly

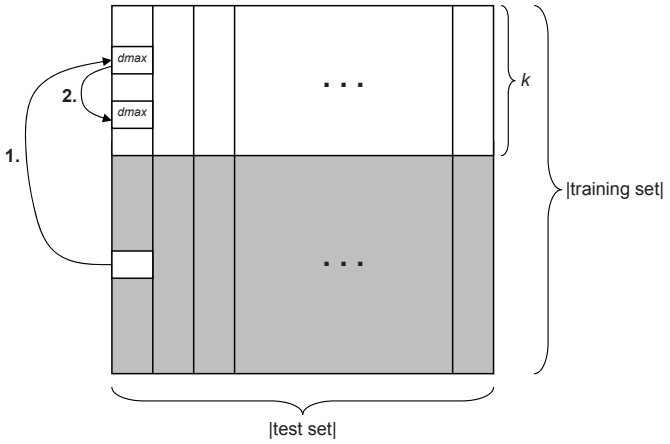

Fig. 1. Schematic description of the modified k-NN search. For each training example two steps are performed: 1. distance is computed; if distance $<$ $dmax$, then the distance is stored at the position of $dmax$; 2. the new $dmax$ is determined.

the application of inequality (2). If one decides to use the inequality, the examination $\frac{1}{2}(k(\mathbf{x},\mathbf{x}) - k(\mathbf{z},\mathbf{z})) \geq dmax$ has to be included before step 14: if that evaluates to true, the algorithm moves on to the next training example. Figure 1 shows the schematic description of this algorithm. We note that the number of neighbors (the length k of the list) and the size of the training set are not scaled proportionally in the diagram.

In our experiments this simple modification of the algorithm along with the application of the second formula could save up to 50% of comparisons. Nevertheless, the savings are greatly influenced by the dataset, the kernel and the number of neighbors. The derivation of the proposed bounds and argumentation for (2) can be found in the appendix.

V. Locality-sensitive hashing

Locality-sensitive hashing aims to hash nearby points to the same value, thus partitioning the dataset into buckets holding similar points. At testing only the bucket of the respective point (found by hashing) has to be searched, that is only a small fraction of the entire dataset. LSH is widely used to facilitate sub-linear search in very large databases in content-based retrieval.

Hash tables are generally used to provide fast access, retrieving data in $O(1)$ without parsing the entire dataset. From this point of view, a hash function inducing less collisions is better than another function with a larger probability of collision. LSH, however, exploits collision to map *similar* values to the same bucket.

In order $h : \mathcal{X} \to \mathcal{V}$ to be called a locality-sensitive hash function, it must fulfill the following conditions:

1) The probability of collision for nearby points is $P(h(\mathbf{x}) = h(\mathbf{z})) \geq p_1$, for $\|\mathbf{x} - \mathbf{z}\| \leq R_1$;
2) The probability of collision for distant points is $P(h(\mathbf{x}) = h(\mathbf{z})) \leq p_2$, for $\|\mathbf{x} - \mathbf{z}\| \geq R_2 = (1+\epsilon)R_1$;
3) $p_1 > p_2$.

One such hash function is the dot-product based function of Charikar [3], which uses a randomly generated hyperplane with normal vector $\mathbf{r}$. The hash keys are binary vectors of length L, where each bit assigned to a random hyperplane is generated by the function

$$h_{\mathbf{r}}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{r}'\mathbf{x} \geq 0 \\ 0, & \text{otherwise} \end{cases} .$$

A hash key for a data point is composed of a sequence of L such random hash functions, $[h_{\mathbf{r}_1}(\mathbf{x}), \ldots, h_{\mathbf{r}_L}(\mathbf{x})]$. It can be proven that if $\mathbf{r}$ is generated from a multivariate Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$, then [8], [16]

$$P(h_{\mathbf{r}}(\mathbf{x}_i) = h_{\mathbf{r}}(\mathbf{x}_j)) = 1 - \frac{1}{\pi}\arccos\left(\frac{\mathbf{x}_i'\mathbf{x}_j}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}\right) .$$

The theorem states that the probability that a random hyperplane separates two points is directly proportional to the angle enclosed by the points, i.e. to $\theta(\mathbf{x}_i, \mathbf{x}_i) = \arccos(\mathbf{x}_i'\mathbf{x}_j/(\|\mathbf{x}_i\|\|\mathbf{x}_j\|))$.

The approximation in k-NN search can be controlled by adjusting the parameter L and defining the search procedure. In the simplest method only one bucket is searched, the one with the obtained hash key. However, if L is large the neighboring hash keys also have to be checked [3], [16], [12], where neighborhood is defined via Hamming distance.

A. Kernelization

The method presented in the previous section is defined on the Euclidean space, however, for some data better performance can be obtained by working in another space by the means of kernels. In this section the kernelization of the method presented previously is described for a single hash function; $\mathbf{r}$ now denotes the random vector in the feature space.

In the LSH literature there were developed only a few kernel-based techniques among which only a small percent can use an arbitrarily chosen kernel. One such method is presented in [12]. It extends on the previous method by choosing a relatively small random sample of the dataset by which it approximates the fractional exponent $-\frac{3}{2}$ of the kernel and again, by randomly sampling from this sample, generates approximately normally distributed vectors.

We define the mean of t randomly sampled feature vectors by $\mathbf{z}_t = \frac{1}{t}\sum_{i\in\mathcal{S}}\phi(\mathbf{x}_i)$, where $\mathcal{S}$ denotes the index set and $\phi : \mathcal{X} \to \mathcal{H}$ is the feature mapping, $k(\mathbf{x},\mathbf{z}) = \phi(\mathbf{x})'\phi(\mathbf{z})$. By the central limit theorem $\frac{1}{\sqrt{t}}\sum_{i\in S}\phi(\mathbf{x}_i)$ is distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{\Sigma})$, provided that the data is centered in the feature space. Here $\mathbf{\Sigma}$ is the unknown covariance matrix of the data. By using the whitening transformation [6], the vector can be distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Thus, we need to compute the vector $\frac{1}{\sqrt{t}}\mathbf{\Sigma}^{-\frac{1}{2}}\sum_{i\in S}\phi(\mathbf{x}_i)$. By the spectral theorem [9], we know that the mapped data, the covariance and the kernel matrix can be written as

$$\begin{aligned} \mathbf{\Phi} &= \mathbf{USV}' , \\ \mathbf{\Sigma} &= \mathbf{\Phi\Phi}' = \mathbf{US}^2\mathbf{U}' , \\ \mathbf{K} &= \mathbf{\Phi}'\mathbf{\Phi} = \mathbf{VS}^2\mathbf{V}' , \end{aligned}$$

where $\mathbf{\Phi}$ denotes the mapped data matrix. The whitening transform thus can be written as $\mathbf{U}\mathbf{S}^{-1}\mathbf{U}'$. It can be seen that if $\mathbf{V}\mathbf{S}^m\mathbf{V}'$ is multiplied by $\mathbf{\Phi}$ and $\mathbf{\Phi}'$ from the left and right respectively, then we obtain $\mathbf{U}\mathbf{S}^{m+2}\mathbf{U}'$. To obtain $\mathbf{\Sigma}^{-\frac{1}{2}}$ we have that $m = -3$. That is

$$\mathbf{\Sigma}^{-\frac{1}{2}} = \mathbf{K}^{-\frac{3}{2}} .$$

Hence, the random Gaussian vector generated in feature space can be written as

$$\mathbf{r} = \frac{1}{\sqrt{t}}\mathbf{\Phi}\mathbf{K}^{-\frac{3}{2}}\sum_{i\in\mathcal{S}}\mathbf{\Phi}'\phi(\mathbf{x}_i) . \tag{3}$$

where randomness is determined by the index set S.

The matrix $\mathbf{K}$ is centered using the centering formula known from kernel PCA [19]:

$$\mathbf{K}_c = \left(\mathbf{I} - \frac{1}{p}\mathbf{1}\mathbf{1}'\right)\mathbf{K}\left(\mathbf{I} - \frac{1}{p}\mathbf{1}\mathbf{1}'\right) .$$

Centering is essential for two reasons: in order to be allowed to omit the mean of the sample (see [12]) and because for positive kernels, e.g. Gaussian or second order polynomial, $\mathbf{r}'\phi(\mathbf{x})$ would only generate sequences of 1's. When calculating the required dot products – i.e. at training and testing phase – $\phi(\mathbf{x})$ and $\mathbf{\Phi}$ has to be zero-centered. The centered points in the feature space can be written as $\mathbf{\Phi}_c = \mathbf{\Phi}(\mathbf{I} - \frac{1}{p}\mathbf{1}\mathbf{1}')$ and the centered $\phi(\mathbf{x})$ is by definition $\phi_c(\mathbf{x}) = \phi(\mathbf{x}) - \frac{1}{p}\sum_{j=1}^{p}\phi(\mathbf{x}_j)$. Thus,

$$\begin{aligned}\phi_c(\mathbf{x})'\mathbf{r} &= \left(\mathbf{k}'_{\mathbf{x}} - \frac{1}{p}\sum_j \mathbf{k}'_{\mathbf{x}_j}\right) \\ &\quad \cdot \underbrace{\left(\mathbf{I} - \frac{1}{p}\mathbf{1}\mathbf{1}'\right)\frac{1}{\sqrt{t}}\mathbf{K}^{-\frac{3}{2}}\sum_{i\in\mathcal{S}}\mathbf{k}_{\mathbf{x}_i}}_{} .\end{aligned}$$

The underbraced expression resulting in a vector can be precomputed for each $\mathbf{r}$ random vector, similarly to the offsets $\frac{1}{p}\sum_j \mathbf{k}_{\mathbf{x}_j}$.

The problem with the above formulation is that for testing a point p kernel calculations are needed. In the following we reduce this number to one by computing the pre-images of the random feature space Gaussian vectors, using a method that does not require *additional* computations.

VI. Kernel LSH with pre-images

The problem of finding the pre-image of a point can be formulated as the following optimization problem: given the image $\boldsymbol{\psi}$ find its (approximate) pre-image $\phi(\mathbf{g}^*)$ by solving

$$\mathbf{g}^* = \arg\min_{\mathbf{g}} \|\boldsymbol{\psi} - \phi(\mathbf{g})\| .$$

where $\mathbf{g} = \phi^{-1}(\boldsymbol{\psi})$, $\mathbf{g} \in \mathcal{X}$ is the point mapped to the feature space. Since we do not know the feature map ϕ, only implicitly through the kernel $k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})'\phi(\mathbf{z})$, the *restoration* of the input space point can pose a difficult problem. A simple method to compute the exact pre-image if that exists, or similarly, to obtain an approximate pre-image – e.g. for the RBF kernel – is to express the kernel by an invertible function of the dot products. That is, we consider the feature expansion $\boldsymbol{\psi} = \sum_{i=1}^{\ell}\alpha_i\phi(\mathbf{x}_i)$. Then, if there exists $\mathbf{g} \in \mathcal{X}$ such that $\boldsymbol{\psi} = \phi(\mathbf{g})$ and there exists an invertible function f_k such that $k(\mathbf{x}, \mathbf{z}) = f_k(\mathbf{x}'\mathbf{z})$, then $\mathbf{g}$ can be computed as

$$\mathbf{g} = \sum_{i=1}^{d} f_k^{-1}\left(\sum_{j=1}^{\ell}\alpha_j k(\mathbf{x}_j, \mathbf{e}_i)\right)\mathbf{e}_i ,$$

where $\{\mathbf{e}_1, \ldots, \mathbf{e}_d\}$ is an orthonormal basis of the input space [18]. Thus, for instance for the polynomial kernel $k_{poly}(\mathbf{x}, \mathbf{z}) = (a\mathbf{x}'\mathbf{z} + b)^c$, we can write

$$f_{poly}^{-1}(k_{poly}(\mathbf{x}, \mathbf{z})) = \frac{1}{a}(k_{poly}(\mathbf{x}, \mathbf{z})^{1/c} - b) .$$

Other approaches include fixpoint methods [14], learning the mapping using ridge regression [1], distance constraint-based methods [13], methods using weakly supervised learning [23] or local isomorphisms [10]. Finding pre-images has important applications in denoising by kernel PCA [14], reduced set expansions [2] and kernel dependency estimation [21].

Our method is a special case of that proposed in [1]. The authors propose to find the pre-image map $\Gamma_j : \mathcal{H} \to \mathcal{X}$ by solving the following optimization problem,

$$\Gamma_j = \arg\min_{\Gamma_j}\sum_{i=1}^{\ell} E(\mathbf{x}_i, \Gamma_j(\phi(\mathbf{x}_i))) + \lambda\Omega(\Gamma) ,$$

where $E(\cdot, \cdot)$ is a loss function, Ω is a regularizer and $\lambda \geq 0$. In our method we use the linear mapping $\Gamma_j(\boldsymbol{\psi}) = \mathbf{w}'_j\boldsymbol{\psi}$, the squared error and we apply no regularization. Using the training dataset we search for a linear mapping $\Gamma_j(\boldsymbol{\psi}) = \mathbf{w}'_j\boldsymbol{\psi}$ in the feature space such that the projection be close to the input space dimension. Thus, we seek for Γ_j, $j = 1, \ldots, d$, where d is the input space dimension, to minimize

$$\sum_{i=1}^{\ell}(x_{ij} - \Gamma_j(\phi(\mathbf{x}_i)))^2 = \sum_{i=1}^{\ell}(x_{ij} - \mathbf{w}'_j\phi(\mathbf{x}_i))^2 , \tag{4}$$

with x_{ij} denoting the j-th dimension of vector $\mathbf{x}_i$. Differentiating with respect to $\mathbf{w}_j$ we obtain

$$\begin{aligned}\mathbf{w}_j &= \left(\sum_{i=1}^{\ell}\phi(\mathbf{x}_i)\phi(\mathbf{x}_i)'\right)^{-1}\sum_{i=1}^{\ell}x_{ij}\phi(\mathbf{x}_i) \\ &= \mathbf{\Sigma}^{-1}\sum_{i=1}^{\ell}x_{ij}\phi(\mathbf{x}_i) \\ &= \mathbf{\Phi}\mathbf{K}^{-2}\mathbf{\Phi}'\sum_{i=1}^{\ell}x_{ij}\phi(\mathbf{x}_i) .\end{aligned}$$

Since we cannot calculate neither $\mathbf{\Sigma}^{-1}$ nor $\mathbf{K}^{-2}$, we use the same p-point sample used in the computation of the random feature space vectors.

Our goal is to obtain the pre-image $\mathbf{g}^{\mathbf{r}}$ of a random Gaussian feature space vector $\mathbf{r}$,

$$\mathbf{g}^{\mathbf{r}} = \begin{pmatrix}\mathbf{w}'_1\mathbf{r} & \mathbf{w}'_2\mathbf{r} & \ldots & \mathbf{w}'_d\mathbf{r}\end{pmatrix}' .$$

Algorithm 2 kLSHp training

Set p, t and L.
Generate p random indices, and also $L \cdot t$ random indices from these ($\mathcal{S}_i$).
for $i \leftarrow 1, L$ **do**
  for $j \leftarrow 1, d$ **do**
    $\mathbf{g}_j^i = \frac{1}{\sqrt{t}} \mathbf{X}_{j\cdot} \mathbf{K}^{-\frac{3}{2}} \sum_{k \in \mathcal{S}_i} \mathbf{k}_{\mathbf{x}_k}$
  end for
  $o_i = \frac{1}{p} \sum_{j=1}^{p} k(\mathbf{x}_j, \mathbf{g}^i)$
end for
for all training data $\mathbf{x}$ **do**
  for $i \leftarrow 1, L$ **do**
    $h_i(\mathbf{x}) = \begin{cases} 1, & \text{if } k(\mathbf{x}, \mathbf{g}^i) - o_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$
  end for
end for

Thus, when building the binary hash sequences, only one kernel computation needs to be performed, $k(\mathbf{x}, \mathbf{g}^{\mathbf{r}})$.

Substituting the respective formulae for the weight and the random vector, the dot product $\mathbf{w}_j' \mathbf{r}$ can be calculated as

$$\begin{aligned} \mathbf{w}_j' \mathbf{r} &= \frac{1}{\sqrt{t}} \mathbf{X}_{j\cdot} \mathbf{K}^{-\frac{3}{2}} \sum_{k \in \mathcal{S}} \mathbf{\Phi}' \phi(\mathbf{x}_k) \\ &= \frac{1}{\sqrt{t}} \mathbf{X}_{j\cdot} \mathbf{K}^{-\frac{3}{2}} \sum_{k \in \mathcal{S}} \mathbf{k}_{\mathbf{x}_k} \ . \end{aligned} \tag{5}$$

This is very similar to $\mathbf{r}$ in (3), with the exception that $\mathbf{\Phi}$ from the left has changed to $\mathbf{X}_{j\cdot}$, to the j-th row of the (sampled) data matrix $\mathbf{X}$. Thus, the same exponent of the kernel matrix has to be computed, no additional intrinsic computations are needed to transform the algorithm of [12] to this variant.

A. Centering

Centering plays a fairly important role in this algorithm. In (3) and (5) the kernel matrix $\mathbf{K}$ is centered, but when calculating $k(\mathbf{x}, \mathbf{g}^{\mathbf{r}})$ zero-centered points are needed everywhere. Since $\mathbf{z}_{\mathbf{r}}$ is centered, we only need to center $\phi(\mathbf{x})$. Hence, we write

$$\begin{aligned} \phi_c(\mathbf{x})' \phi(\mathbf{g}^{\mathbf{r}}) &= \left(\phi(\mathbf{x}) - \frac{1}{p} \sum_{j=1}^{p} \phi(\mathbf{x}_j) \right)' \phi(\mathbf{g}^{\mathbf{r}}) \\ &= k(\mathbf{x}, \mathbf{g}^{\mathbf{r}}) - \frac{1}{p} \sum_{j} k(\mathbf{x}_j, \mathbf{g}^{\mathbf{r}}) \ . \end{aligned}$$

The offsets $\frac{1}{p} \sum_j k(\mathbf{x}_j, \mathbf{g}^{\mathbf{r}})$ can be calculated for each $\mathbf{g}^{\mathbf{r}}$ in the training phase, thus centering requires only a subtraction at testing.

B. Summary of the algorithm

Although no separate training process was apparent in k-NN search, in kLSH there is a training phase: building the hash table, i.e. defining data clusters. Prediction is the same, however, before going through the training examples the test point is hashed in order to determine its broader neighborhood; the k-nearest neighbors will be selected from this cluster. Thus – assuming a nearly even distribution in the hash buckets – the number of comparisons are reduced by 2^L, where L denotes the length of the binary hash keys.

Algorithm 3 kLSHp prediction

for all test data $\mathbf{x}$ **do**
  for $i \leftarrow 1, L$ **do**
    $h_i(\mathbf{x}) = \begin{cases} 1, & \text{if } k(\mathbf{x}, \mathbf{g}^i) - o_i \geq 0 \\ 0, & \text{otherwise} \end{cases}$
  end for
  Determine k-nearest neighbors from examples hashed to $[h_1(\mathbf{x}), h_2(\mathbf{x}), \ldots, h_L(\mathbf{x})]$
end for

The training and testing processes are summarized in Algorithm 2 and 3. Here the pre-images of the random vectors are denoted by $\mathbf{g}^i$, while the offsets $\frac{1}{p} \sum_{j=1}^{p} k(\mathbf{x}_j, \mathbf{g}^i)$ are denoted by o_i. L is number of hash functions (or equivalently the number of random vectors), p is the size of the sample and t is the smaller sample size, sampled from the p points. There will be L such samples – that is of size t – the indices of which are stored in $\mathcal{S}_i$, $i = 1, \ldots, L$.

VII. Experimental methodology and results

The program was written in C++ and the experiments were run on a desktop PC with Intel Core i7 processor of 3.40GHz and 8GB of RAM. For calculating the powers of the kernel matrix, that is for performing the eigendecomposition of the Gram matrix we used the ALGLIB numerical analysis library.[2]

The experiments were performed on five datasets: Optdigits, DNA, Adult, Forest Covertype and Poker Hand.[3] Optdigits is a handwritten digit recognition database in which the examples are 17-color (greyscale) images of handwritten digits, therefore the attributes are integers between 0 and 16. The training and test examples are distributed approximately equally among the 10 classes.

The DNA Statlog dataset is a database containing DNA sequences (A, C, G, T) for recognizing exon/intron boundaries in order to splice out redundant data. We call exons the segments with *meaning*, while introns are those segments that do not carry any relevant information when "producing" the protein. There are three classes: exon/intron sites, intron/exon sites and sequences with no splicing sites. Each sequence is of length 60, starting from position -30 and ending at 30. The categorical attributes are converted to binary using 3 bits for each, thus transforming an example to a sequence of length 180. The training and test examples are distributed approximately 25%–25%–50% among the three classes.

The Adult dataset contains census data to predict whether a household has larger income than 50 000 dollars per year. Eight of the attributes are categorical and the remaining six are continuous, quantized into quintiles, yielding a total number

[2] http://www.alglib.net

[3] The datasets can be downloaded from the UCI Machine Learning Repository, http://archive.ics.uci.edu/ml/.

Dataset	Training points	Test points	Dimensionality	No. of classes
Optdigits	3823	1797	64	10
DNA	2000	1186	180	3
Adult	1605	30 956	123	2
Forest Covertype	461 012	120 000	54	7
Poker Hand	1 000 000	25 010	10	10

TABLE I
PROPERTIES OF THE DATASETS USED IN THE EXPERIMENTS.

Dataset	k in kNN search	L	p	t	kernel settings
Optdigits	1	5	300	50	poly, $a = 0.01$, $b = 1$, $c = 2$
DNA	3	5	300	50	poly, $a = 0.01$, $b = 1$, $c = 2$
Adult	7	5	300	50	RBF, $\gamma = 0.001$
Forest Covertype	1	80	300	50	poly, $a = 0.01$, $b = 1$, $c = 2$
Poker Hand	3	15	300	50	sigmoid, $a = 0.001$, $b = 1$

TABLE II
PARAMETERS FOR KLSH AND KLSHP EXPERIMENTS. THE INDICATED KERNEL SETTINGS WERE USED IN KERNEL k-NN SEARCH TOO.

	ineq			kLSH			kLSHp		
	Comp.	Time	Acc.	Comp.	Time	Acc.	Comp.	Time	Acc.
Optdigits	81.5 ± 0.17	1.09 ± 0.004	97.87 ± 0.02	4.12 ± 0.23	0.17 ± 0.005	93.18 ± 0.63	5.26 ± 0.88	0.08 ± 0.01	94.64 ± 0.45
DNA	99.75 ± 0.04	1.12 ± 0.004	78.47 ± 0.24	3.24 ± 0.03	0.21 ± 0.004	66.48 ± 1.37	3.33 ± 0.09	0.045 ± 0.005	67.04 ± 1.3
Adult	100	24.96 ± 0.01	82.03 ± 0.17	3.98 ± 0.21	5.89 ± 0.05	79.71 ± 0.41	3.92 ± 0.4	1.23 ± 0.1	79.91 ± 0.42
Forest Covertype	41.64 ± 10^{-6}	4509.95 ± 3.25	94.76 ± 0.01	8.93 ± 10.12	1979.67 ± 1987.6	95.43 ± 0.06	4.67 ± 3.09	1081.7 ± 687.48	95.35 ± 0.05
Poker Hand	52.34 ± 10^{-6}	1323.29 ± 0.65	61.14 ± 0.07	0.89 ± 0.22	94.13 ± 22.52	60.38 ± 0.34	0.97 ± 0.2	101.87 ± 20.57	60.07 ± 0.38

TABLE III
EXPERIMENTAL RESULTS OBTAINED FOR THE OPTDIGITS, DNA AND ADULT DATASETS: INEQ – k-NN USING INEQUALITY (2), KLSH – k-NN WITH KERNEL LSH [12], KLSHP – k-NN WITH KERNEL LSH USING PRE-IMAGES (OUR METHOD). THE RESULTS ARE GIVEN IN PERCENTAGE.

of 123 binary features. The examples are distributed as 75%–25% among the negative and positive class.

The Forest Covertype dataset is a database describing forest cover types (spruce-fir, lodgepole pine, etc.) by cartographic variables (elevation, slope, etc.). Among the 54 attributes we find categorical as well as continuous ones. These are used to predict the class variable from the 7 possible forest cover types. In our experiments a random split of the 581 012 examples was used, as shown in Table I.

The Poker Hand dataset contains hands consisting of five cards, and the task is to predict the poker hand (e.g. straight, full house, etc.). Each card is described by 2 variables: one describing the suit of the card, and another one describing its rank, thus resulting in a total of 10 attributes. The class distribution was set to be directly proportional to the actual distribution in the full domain. We swapped the training and test data, because we are interested in learning scenarios where the training data is the larger.

The datasets were chosen such that to gather data from various domains with varying class count and dimensionality.

For each dataset three algorithms were tested: ineq – k-NN using inequality (2), kLSH – k-NN with kernel LSH [12], kLSHp – k-NN with kernel LSH using pre-images (our method). The obtained results are shown in Table III. We measured three quantities: the ratio of the number of comparisons made during the test phase and the total number of comparisons needed for standard k-NN (percentage), timing (seconds) and accuracy (percentage). Since the methods are probabilistic, two values are given: the mean and the standard deviation of the results of 10 independent runs. We did not apply the derived bounds in kLSH and kLSHp algorithms to avoid mixing the results and obfuscating the effects of the different algorithms.

In our experiments we used the polynomial, RBF and sigmoid kernels:

$$k_{poly}(\mathbf{x}, \mathbf{z}) = (a\mathbf{x}'\mathbf{z} + b)^c \ ,$$

$$k_{rbf}(\mathbf{x}, \mathbf{z}) = \exp\left(-\gamma\|\mathbf{x} - \mathbf{z}\|^2\right) \ ,$$

$$k_{sigm}(\mathbf{x}, \mathbf{z}) = \tanh(a\mathbf{x}'\mathbf{z} + b) \ .$$

The kernel and other parameters used in the experiments are shown in Table II. In addition to simple k-NN, for kLSH and kLSHp we needed three other parameters. The length of the binary hash key (the number of random hyperplanes) was set to 5 for Optidigits, DNA and Adult, and to 80 and 15 for the Forest Covertype and Poker Hands datasets. For the first three dataset the value of L was set to a lower value to prevent the application of more sophisticated search mechanisms; the hash key was calculated for each test point and that was compared to all the examples in the respective bucket.

For the last two sets we applied a search method similar to the algorithm described in [3]. For each hash key encountered when building the hash table the Hamming neighborhood was defined; we determined the 10 closest hash keys. Furthermore,

the hash keys were sorted lexicographically. For an unseen point the hash key was computed and the point was compared to the examples found in the buckets of its neighboring hash keys. If the computed hash key was inexistent in the hash table, the closest hash key was determined using a binary search on the lexicographically sorted hash key list.

The size p of the sample was set to 300 and the size t of the second sample was set to 50 in each experiment. These values were chosen according to [12].

In the experiments using inequality (2) we shuffled the data to obtain different learning scenarios, since the order of the training data influences the effect of the inequality. Shuffling was done by selecting each training example with probability 0.5 and swapping it with a randomly selected point.

Our aim was not to improve on the accuracy but the speed of the system, to compare our method using pre-images to the technique proposed in [12] and show its competitiveness. We reported the number of comparisons during testing and also the timing results. The timings show the time spent for label prediction only, excluding the preprocessing steps.

No heuristic parameter selection was performed, since different parameters imply the same results in this case. The actual parameters were selected by observing the best test results for some randomly chosen parameter values.

The effects of using the inequalities are greatly influenced by the datasets, the kernel function applied and the parameter k of the nearest neighbor search – this is shown in Table III in the columns of "ineq". If we find the maximum distance $dmax$ between the test point and training points early in the search process, no further savings in the distance computations can be made. The kernel also determines the number of comparisons: as shown in the table, no distance computations can be skipped for the Adult dataset because the kernel applied (RBF) is translation invariant. Nevertheless, in case of translation invariant or normalized kernels the inequality can also filter out a few computation if k is small, since it can happen then after filling the list, $dmax$ becomes zero. Another influencing factor is the number k of the nearest neighbors: as the list becomes longer, the probability to encounter a point $\mathbf{z}_i$ for which $\frac{1}{2}(k(\mathbf{x},\mathbf{x}) - k(\mathbf{z}_i,\mathbf{z}_i)) \geq dmax$ decreases, which is a natural consequence of Algorithm 1.

We also made experiments using the regularizer $\lambda\|\mathbf{w}_j\|^2$ in (4) obtaining $\mathbf{w}_j'\mathbf{r} = \frac{1}{\sqrt{t}}\mathbf{X}_{j\cdot}(\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{K}^{-\frac{1}{2}}\sum_{k\in\mathcal{S}}\mathbf{k}_{\mathbf{x}_k}$ and used the approximation $\mathbf{w}_j'\mathbf{r} \approx \frac{1}{\sqrt{t}}\mathbf{X}_{j\cdot}(\mathbf{K}+\lambda\mathbf{I})^{-\frac{3}{2}}\sum_{k\in\mathcal{S}}\mathbf{k}_{\mathbf{x}_k}$. We did not notice significant differences, therefore we decided not to report these results.

The accuracy results show that our method of computing the approximate pre-images implies approximately the same performance as the original method. The timing results, however, indicate that calculating the pre-images can significantly speed up k-NN search.

VIII. Conclusion and future work

In this paper we proposed a method to save significant computational resources in kernel locality-sensitive hashing using arbitrary kernels. Our method extends on the kernelized LSH presented in [12] by computing and using the pre-images of the random Gaussian feature space vectors. Two inequalities are also proposed to speed up more k-NN search.

We applied the method from [1] and obtained pre-images that can be calculated by minimal additional changes compared to the base technique. Thus, an existing application implementing the algorithm from [12] needs only minimal changes to obtain significant speedup. Our method speeds up the kernel computations up to a factor of p (where p is the sample size) while providing similar results to original kLSH. This is also confirmed by the experiments carried out.

Since presently we tested the method only on small and moderately sized datasets we plan to test it on much larger databases. It is also interesting to investigate the benefits of the algorithms using large but sparse data, since then the cost of calculating the dot product or the distance of two vectors decreases significantly.

Acknowledgment

The authors acknowledge the support of the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278.

Appendix
Derivation of the bounds

For deriving the inequalities the positive definiteness property is used. A Gram matrix $\mathbf{K}$, $K_{ij} = k(\mathbf{x}_i,\mathbf{x}_j)$, is called positive definite if

$$\mathbf{c}'\mathbf{K}\mathbf{c} \geq 0$$

for all vectors $\mathbf{c}$. We know that the 2×2 kernel matrix is positive definite, therefore its determinant is non-negative, that is

$$k(\mathbf{x},\mathbf{x})k(\mathbf{z},\mathbf{z}) - k^2(\mathbf{x},\mathbf{z}) \geq 0\ ,$$

or equivalently

$$-2k(\mathbf{x},\mathbf{z}) \geq -2\sqrt{k(\mathbf{x},\mathbf{x})k(\mathbf{z},\mathbf{z})}\ .$$

To obtain the first inequality the left side has to be expanded so that to arrive to the Euclidean distance in the feature space defined by the kernel function. Thus, we obtain

$$d(\mathbf{x},\mathbf{z}) \geq k(\mathbf{x},\mathbf{x}) + k(\mathbf{z},\mathbf{z}) - 2\sqrt{k(\mathbf{x},\mathbf{x})k(\mathbf{z},\mathbf{z})}\ . \quad (6)$$

The second inequality is obtained in a similar fashion. Now we use the 3×3 kernel matrix to obtain the formula. Let $\mathbf{c} = \begin{pmatrix}-1 & 1 & 1\end{pmatrix}$ and we know that $\mathbf{c}'\mathbf{K}\mathbf{c} \geq 0$, leading to

$$\begin{aligned} &k(\mathbf{x},\mathbf{x}) - 2k(\mathbf{x},\mathbf{y}) + k(\mathbf{y},\mathbf{y}) - 2k(\mathbf{x},\mathbf{z}) \\ &+k(\mathbf{z},\mathbf{z}) + 2k(\mathbf{y},\mathbf{z}) \geq 0\ . \end{aligned}$$

Adding $k(\mathbf{x},\mathbf{x}) - k(\mathbf{y},\mathbf{y}) - k(\mathbf{z},\mathbf{z})$ to both sides we obtain

$$d(\mathbf{x},\mathbf{y}) + d(\mathbf{x},\mathbf{z}) - d(\mathbf{y},\mathbf{z}) \geq k(\mathbf{x},\mathbf{x}) - k(\mathbf{y},\mathbf{y}) - k(\mathbf{z},\mathbf{z})\ . \quad (7)$$

The triangle inequality tells us that

$$d(\mathbf{x},\mathbf{y}) \leq d(\mathbf{x},\mathbf{z}) + d(\mathbf{y},\mathbf{z})\ ,$$

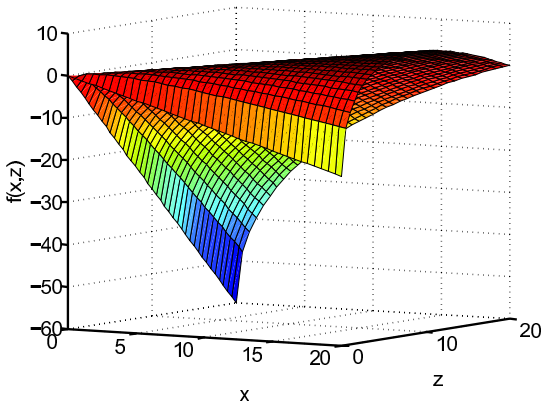

Fig. 2. Plot of function $f(x,z) = -x - 3z + 4\sqrt{xz}$ in $[0,20]$.

from which

$$d(\mathbf{x},\mathbf{y}) + d(\mathbf{x},\mathbf{z}) - d(\mathbf{y},\mathbf{z}) \leq 2d(\mathbf{x},\mathbf{z}) \ . \tag{8}$$

From (7) and (8) we are to obtain our second inequality, that is

$$d(\mathbf{x},\mathbf{z}) \geq \frac{1}{2}(k(\mathbf{x},\mathbf{x}) - k(\mathbf{z},\mathbf{z}) - k(\mathbf{y},\mathbf{y})) \quad \forall \mathbf{y} \ .$$

Since this holds for all $\mathbf{y}$ and we want that the right side to be as small as possible, we can use $k(\mathbf{y},\mathbf{y}) = 0$, therefore

$$d(\mathbf{x},\mathbf{z}) \geq \frac{1}{2}(k(\mathbf{x},\mathbf{x}) - k(\mathbf{z},\mathbf{z})) \ . \tag{9}$$

From inequalities (6) and (9) we would like to use the most *beneficial* one. First, since (9) contains only an addition and a multiplication by a constant, it is our recommendation at first glance. On the other hand, we want to find that formula which offers a *lower* upper bound on the Euclidean distance. This is because when comparing distances in k-NN we want a tighter bound to filter out as many distance calculations as we can. Using notations $x := k(\mathbf{x},\mathbf{x})$, $z := k(\mathbf{z},\mathbf{z})$, $t := \sqrt{\frac{x}{z}}$ we obtain the function $-t - \frac{3}{t} + 4$. By simple analysis it can be shown that this function, defined on $[0,\infty)$, takes positive values *only* in the interval $[1,3]$ and negatives in $[0,1) \cup (3,\infty)$, that is the right side of (9) is in most cases smaller than the right side of (6). So, following our second argument the winner formula is again (9). For visual demonstration, on Figure 2 the function $f(x,z) = -x - 3z + 4\sqrt{xz}$ was plot in the interval $[0,20]$.

References

[1] G. H. Bakir, J. Weston, and B. Schölkopf, "Learning to find pre-images," in *NIPS*. MIT Press, 2003.

[2] C. J. C. Burges, "Simplified support vector decision rules," in *Proc. 13th International Conference on Machine Learning*. Morgan Kaufmann, 1996, pp. 71–77.

[3] M. Charikar, "Similarity estimation techniques from rounding algorithms," in *STOC*, 2002, pp. 380–388.

[4] T. M. Cover and P. E. Hart, "Nearest neighbor pattern classification," *IEEE Transactions on Information Theory*, vol. IT-13, 1967.

[5] Datar, Immorlica, Indyk, and Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *COMPGEOM: Annual ACM Symposium on Computational Geometry*, 2004.

[6] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York: Wiley, 2001.

[7] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*. San Francisco: Morgan Kaufmann, Sep. 1999, pp. 518–529.

[8] M. X. Goemans and D. P. Williamson, "Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming," *Journal of the ACM*, vol. 42, no. 6, pp. 1115–1145, 1995.

[9] G. H. Golub and C. F. V. Loan, *Matrix Computations*, 3rd ed. Baltimore: Johns Hopkins University Press, 1996.

[10] D. Huang, Y. Tian, and F. D. la Torre, "Local isomorphism to solve the pre-image problem in kernel methods," in *CVPR*. IEEE, 2011, pp. 2761–2768.

[11] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *STOC*, 1998, pp. 604–613.

[12] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," in *ICCV*. IEEE, 2009, pp. 2130–2137.

[13] J. T.-Y. Kwok and I. W.-H. Tsang, "The pre-image problem in kernel methods," *IEEE Transactions on Neural Networks*, vol. 15, no. 6, pp. 1517–1525, 2004.

[14] S. Mika, B. Schlkopf, A. Smola, K. robert Mller, M. Scholz, and G. Rtsch, "Kernel pca and de-noising in feature spaces," in *Advances in Neural Information Processing Systems 11*. MIT Press, 1999, pp. 536–542.

[15] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *NIPS*. Curran Associates, Inc, 2009, pp. 1509–1517.

[16] D. Ravichandran, P. Pantel, and E. Hovy, "Randomized algorithms and NLP: Using locality sensitive hash functions for high speed noun clustering," in *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*. Ann Arbor, Michigan: Association for Computational Linguistics, Jun. 2005, pp. 622–629.

[17] R. Salakhutdinov and G. E. Hinton, "Semantic hashing," *Int. J. Approx. Reasoning*, vol. 50, no. 7, pp. 969–978, 2009.

[18] B. Schölkopf and A. J. Smola, *Learning with Kernels*. Cambridge, MA: MIT Press, 2002.

[19] B. Schölkopf, A. J. Smola, and K.-R. Müller, "Kernel principal component analysis," *Advances in kernel methods: support vector learning*, pp. 327–352, 1999.

[20] Y. Weiss, A. B. Torralba, and R. Fergus, "Spectral hashing," in *NIPS*. MIT Press, 2008, pp. 1753–1760.

[21] J. Weston, O. Chapelle, A. Elisseeff, B. Schölkopf, and V. Vapnik, "Kernel dependency estimation," in *NIPS*. MIT Press, 2002, pp. 873–880.

[22] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. F. M. Ng, B. Liu, P. S. Yu, Z.-H. Zhou, M. Steinbach, D. J. Hand, and D. Steinberg, "Top 10 algorithms in data mining," *Knowl. Inf. Syst*, vol. 14, no. 1, pp. 1–37, 2008.

[23] W. S. Zheng, J. H. Lai, and P. C. Yuen, "Weakly supervised learning on pre-image problem in kernel methods," in *ICPR*, 2006, pp. II: 711–715.