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

Zalán Bodó and Lehel Csató

Faculty of Mathematics and Computer Science,
Babeș–Bolyai University, Cluj-Napoca/Kolozsvár

IJCNN/WCCI, Brisbane, Australia, June 2012

Contents

Introduction

- large databases – to find the k-NN of a point requires to compare the point to every training example = linear search time for one point
- how to improve on this?
- some existing approaches:
  - locality-sensitive hashing (LSH)
  - based on p-stable distributions
  - semantic hashing
  - spectral hashing
  - etc.
- using kernels – points can be mapped to another space, different from the input space; one can vary the similarity function without actually performing the mapping
- $\rightarrow$ kernelized LSH – problem: too many kernel computation

Our contributions:

- a slightly modified algorithm for k-NN search (needed for the next item)
- inequalities to omit certain comparisons
- kernel LSH using the pre-images of random feature space vectors

k-nearest neighbors

- k-NN:

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

  (We use $sim(\mathbf{z}, \mathbf{x}) = 1$)

- to determine the neighbors $N_k(\cdot)$ the Euclidean distance is used:

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

- using an arbitrary kernel – apply the kernel trick:

$$d(\mathbf{x}, \mathbf{z}) = k(\mathbf{x}, \mathbf{x}) + k(\mathbf{z}, \mathbf{z}) - 2k(\mathbf{x}, \mathbf{z})$$

$$k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})'\phi(\mathbf{z})$$

- usual k-NN algorithm: compute distances + sort

Modified k-NN

In order to omit some comparisons, we modify the algorithm by incorporating the sorting (for one point $\mathbf{x}$):

k-NN

1: $i \leftarrow 0$
2: $dmax \leftarrow -1$
3: $imax \leftarrow -1$
4: **for** every training point $\mathbf{z}_i$ **do**
5: **if** $i < k$ **then**
6: Compute/store $d(\mathbf{x}, \mathbf{z}_i)$ (and index) in a list.
7: **if** distance $> dmax$ **then**
8: $dmax \leftarrow d(\mathbf{x}, \mathbf{z}_i)$
9: $imax \leftarrow$ index of $dmax$ in the list
10: **end if**
11: $i \leftarrow i + 1$
12: Go to step 4.
13: **end if**
14: Compute distance $d(\mathbf{x}, \mathbf{z}_i)$.
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: $i \leftarrow i + 1$
20: **end for**

The kernel values ($k(\mathbf{x}, \mathbf{x})$) are calculated and stored before the running the search procedure; for test points has to be computed too.
Inequalities used:

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

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

(The second one is *better*, so we use that one.)

- *dmax* and *imax* contains the largest distance and its corresponding index
- $\frac{1}{2}(k(\mathbf{x}, \mathbf{x}) - k(\mathbf{z}, \mathbf{z})) \geq dmax$ has to be included before step 14: if that evaluates to true, the algorithm moves on to the next training example
- this means $d(\mathbf{x}, \mathbf{z}) \geq \frac{1}{2}(k(\mathbf{x}, \mathbf{x}) - k(\mathbf{z}, \mathbf{z})) \geq dmax$, so the distance of $\mathbf{x}$ is larger than the max. distance encountered so far

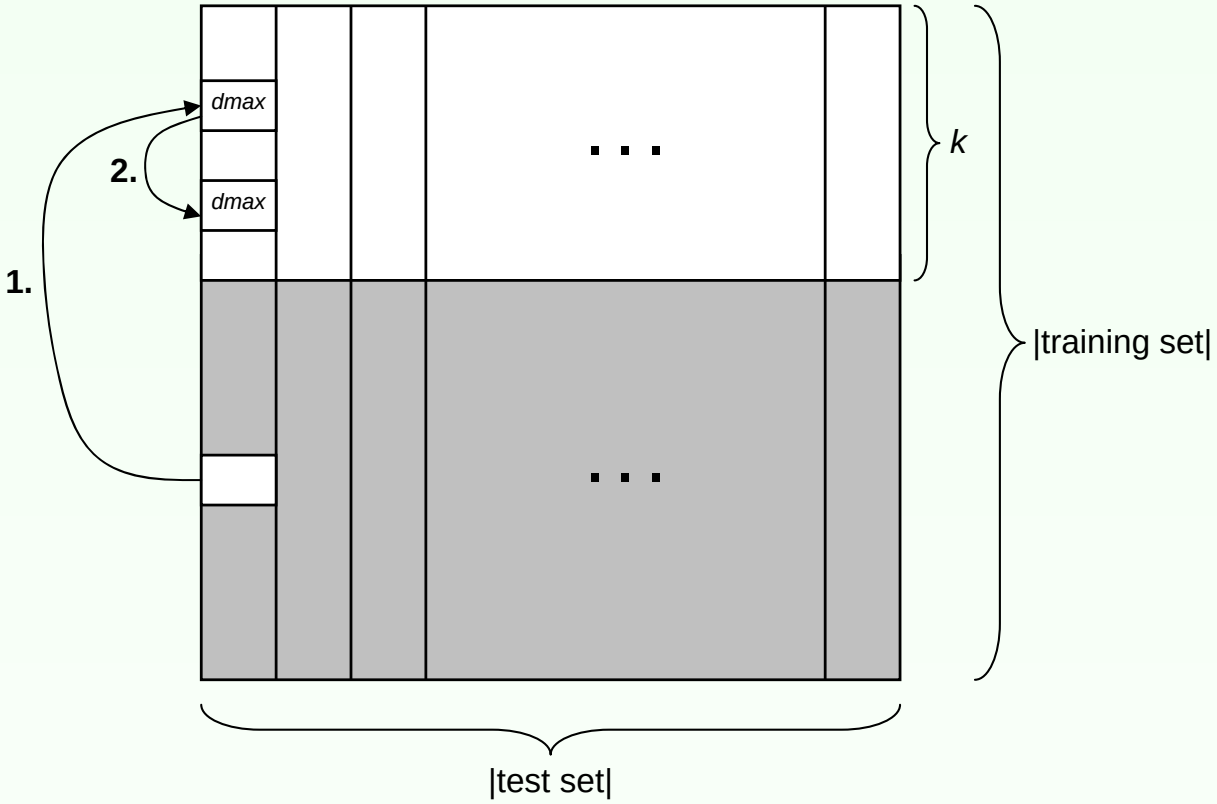

Figure: Schematic description of the modified k-NN search. For each training example two steps are performed: 1. distance is computed; if distance $<$ *dmax*, then the distance is stored at the position of *dmax*; 2. the new *dmax* is determined.

1st inequality

2×2 kernel matrix is positive definite $\Rightarrow$ determinant is non-negative

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

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

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

2nd inequality

Use the 3×3 kernel matrix.
We know that $\mathbf{c}'\mathbf{K}\mathbf{c} \geq 0$ and choose $\mathbf{c} = \begin{pmatrix} -1 & 1 & 1 \end{pmatrix}$:

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

Add $k(\mathbf{x},\mathbf{x}) - k(\mathbf{y},\mathbf{y}) - k(\mathbf{z},\mathbf{z})$:

$$d(\mathbf{x},\mathbf{y}) + d(\mathbf{x},\mathbf{z}) - d(\mathbf{y},\mathbf{z}) \geq k(\mathbf{x},\mathbf{x}) - k(\mathbf{y},\mathbf{y}) - k(\mathbf{z},\mathbf{z}) \qquad (a)$$

From the triangle inequality:

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

$$d(\mathbf{x},\mathbf{y}) + d(\mathbf{x},\mathbf{z}) - d(\mathbf{y},\mathbf{z}) \leq 2d(\mathbf{x},\mathbf{z}) \qquad (b)$$

From (a) and (b)

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

Smallest right side: $k(\mathbf{y},\mathbf{y}) = 0$

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

Locality-sensitive hashing

- hash nearby points to the same value $\Rightarrow$ partition the dataset into buckets holding similar points
- LSH exploits hash collisions
- use random hyperplanes with normal $\mathbf{r}$:

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

- hash key: $[h_{\mathbf{r}_1}(\mathbf{x}), \ldots, h_{\mathbf{r}_L}(\mathbf{x})]$
- if $\mathbf{r}$ is generated from a multivariate Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$ then

$$\begin{aligned} P(h_{\mathbf{r}}(\mathbf{x}_i) = h_{\mathbf{r}}(\mathbf{x}_j)) &= 1 - \frac{1}{\pi} \arccos\left(\frac{\mathbf{x}_i'\mathbf{x}_j}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}\right) \\ &= 1 - \frac{1}{\pi}\theta(\mathbf{x}_i, \mathbf{x}_j) \end{aligned}$$

Kernelization

- underlying idea:
  - choose a relatively small random sample of the dataset (p)
  - approximate the fractional exponent $-\frac{3}{2}$ of the kernel
  - again sample randomly from this sample, generating approximately normally distributed vectors (t)
- 3 important parameters: p – sample size, t – size of the smaller sample, L – hash sequence length
- mean of t randomly sampled feature vectors: $\mathbf{z}_t = \frac{1}{t}\sum_{i\in\mathcal{S}}\phi(\mathbf{x}_i)$
- central limit theorem: $\frac{1}{\sqrt{t}}\sum_{i\in\mathcal{S}}\phi(\mathbf{x}_i)$ is distributed as $\mathcal{N}(\mathbf{0},\mathbf{\Sigma})$ (provided the data is centered in the feature space)
- compute $\frac{1}{\sqrt{t}}\mathbf{\Sigma}^{-\frac{1}{2}}\sum_{i\in\mathcal{S}}\phi(\mathbf{x}_i)$ (whitening)
- one can show: $\mathbf{\Sigma}^{-\frac{1}{2}} = \mathbf{\Phi}\mathbf{K}^{-\frac{3}{2}}\mathbf{\Phi}'$ (**!typo in paper!**)
- random Gaussian vector in feature space:

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

- centering:

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

$$\mathbf{\Phi}_c = \mathbf{\Phi}(\mathbf{I} - \frac{1}{p}\mathbf{1}\mathbf{1}'); \quad \phi_c(\mathbf{x}) = \phi(\mathbf{x}) - \frac{1}{p}\sum_{j=1}^{p}\phi(\mathbf{x}_j)$$

- the desired dot product:

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

The underbraced expression can be precomputed for each $\mathbf{r}$ random vector; the offsets $\frac{1}{p}\sum_j \mathbf{k}_{\mathbf{x}_j}$ likewise.

Kernel LSH with pre-images

- goal: minimize the number of kernel computations
- how: use the pre-images of feature space points
- pre-images: given the image ψ find its (approximate) pre-image $\phi(\mathbf{g}^*)$ by solving

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

- our approach: find the pre-image map $\Gamma_j : \mathcal{H} \to \mathcal{X}$ by solving

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

- we use: $\Gamma_j(\psi) = \mathbf{w}_j'\psi$, squared error, no regularization:

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

with x_{ij} denoting the j-th dimension of vector $\mathbf{x}_i$

- solution:

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

- we want to obtain the pre-image $\mathbf{g}^{\mathbf{r}}$ of a random Gaussian feature space vector $\mathbf{r}$:

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

- the final dot product:

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

- centering:
  - $\mathbf{K}$ is centered
  - $\mathbf{g}^{\mathbf{r}}$ is centered
  - need to center $\phi(\mathbf{x})$:

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

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 **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

Prediction:

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

Experiments

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: Properties of the datasets used in the experiments.

Dataset	k in kNN	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: Parameters for kLSH and kLSHp experiments. The indicated kernel settings were used in kernel k-NN search too.

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

Table: Experimental results obtained for the Optdigits, DNA and Adult datasets: ineq – k-NN using inequality (2), kLSH – k-NN with kernel LSH, kLSHp – k-NN with kernel LSH using pre-images (our method).

Thank you!

Questions?

Supported by the Romanian Ministry of Education and Research via grant PN-II-RU-TE-2011-3-0278
and