Fast Large-Scale Kernel k-Nearest Neighbors

Zalán Bodó, zbodo@cs.ubbcluj.ro

The k-nearest neighbor classification was introduced by Cover and Hart in [Cover and Hart, 1967], and since then it is claimed to be one of the most successful classification methods. By its simplicity and ease of implementation it is widely used in a wide variety of applications from text categorization to image retrieval. But its simplicity and the 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, i.e. N comparisons have to be made, that is N distance calculations. If for example the distance does not have a closed form like graph distances, moreover it is a complex one, there is little chance to apply it for large-scale data settings. It the last few years several novel algorithms were proposed for fast nearest neighbor search, namely Locality Sensitive Hashing (LSH) techniques [Andoni and Indyk, 2008]. However, most of them operate in the Euclidean (L^2) or L^1 space. In the recent literature the method was extended to support reproducing Hilbert spaces [Schölkopf and Smola, 2002]. In this research we want to analyze existent techniques, improve on its performance and develop other large-scale kernel-based methods for fast approximate k-nearest neighbors.

1 k-nearest neighbors

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

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

where the function f assigns a label to 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 kNN, where all the neighbors have the same influence on the predicted label.

In order to efficiently implement the kNN 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, that most of the time is Euclidean distance. The Euclidean distance can be rewritten in form of dot products as

$$\|\mathbf{x} - \mathbf{z}\|_2^2 = \langle \mathbf{x}, \mathbf{x} \rangle + \langle \mathbf{z}, \mathbf{z} \rangle - 2 \cdot \langle \mathbf{x}, \mathbf{z} \rangle$$

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

2 Distance inequalities

Since kNN 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.

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 in A.1 that (2) offers a better solution since most of the time it gives a lower upper bound the distance $d(\mathbf{x}, \mathbf{z})$.

In the training phase we store the data along with their squared norm in the feature space, that is $k(\mathbf{x}, \mathbf{x})$. Most kNN implementations store all the distances and when comparisons are over this list of (distance, index) pairs is sorted in increasing order, from which the first k neighbors are obtained. However another method is proposed here, which maintains a list of k (distance, index) pairs and stores the smallest distances in it. Since k is usually small – 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, store the (distance, index) pair in it on the next available position and update M containing the maximal distance value encountered so far. If it is full, we perform the following steps: check for whether $\frac{1}{2}(k(\mathbf{x}, \mathbf{x}) - k(\mathbf{z}, \mathbf{z})) \geq M$. If it evaluates to true, then skip and move the next training example. It M is less, then store the distance in the list on the position of M; go through the list and store the new M and its index.

Experiments show that this simple modification to the algorithm along with the application of the second formula can save up to 25% of comparisons.

3 Locality sensitive hashing

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

Let $h : X^d \to V$ denote the function. In order 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 $\|\mathbf{x} - \mathbf{z}\| \geq cR_1 = (1 + \epsilon)R_2$;
3. $p_1 > p_2$.

One such hash function is the dot-product based function of Charikar [Charikar, 2002] 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 such a random hyperplane is generated by the function

$$h_{\mathbf{r}}(\mathbf{x}) = \begin{cases} 1, & \text{if } \mathbf{r}'\mathbf{x} \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

It can be proven that if $\mathbf{r}$ is generated from a multivariate Gaussian $\mathcal{N}(\mathbf{0}, \mathbf{I})$, then

$$P(sgn(\mathbf{r}'\mathbf{x}_i) = sgn(\mathbf{r}'\mathbf{x}_j)) = 1 - \frac{1}{\pi}\cos^{-1}\left(\frac{\mathbf{x}_i'\mathbf{x}_j}{\|\mathbf{x}_i\|\|\mathbf{x}_j\|}\right)$$

The approximation can be controlled by adjusting the parameter L and defining the searching procedure. In the simplest method only one bucket is searched, the one with the obtained hash key. However, if L is large – i.e. one requires a faster but less accurate method – the neighboring hash keys also have to be checked. Neighboring is defined by Hamming distance.

3.1 Kernelization

The method presented in the previous section is defined on the Euclidean space, however for some problems a better performance can be obtained by working in another space by the means of kernels. In the LSH literature there were developed only a few kernel-based techniques among which only a small percent can be used with an arbitrarily chosen kernel. One such method is presented in [Kulis and Grauman, 2009]. It extends on the previous method by choosing a relatively small random sample of the data set by which it approximates the inverse of the kernel and again, by randomly sampling from this sample, generates approximately Gaussian distributed vectors. The random hyperplane is then

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

where Φ is the data matrix in the feature space, $\mathbf{K}$ is the randomly sampled kernel matrix of p points, and S is a random index set of size t of the p points. The matrices Φ and $\mathbf{K}$ are centered using the centering formula known from kernel PCA [Schölkopf et al., 1999]:

$$\mathbf{K}_c = \mathbf{K} - \frac{1}{p}\mathbf{1}_{pp}\mathbf{K} - \frac{1}{p}\mathbf{K}\mathbf{1}_{pp} + \frac{1}{p^2}\mathbf{1}_{pp}\mathbf{K}\mathbf{1}_{pp}$$

We aim to analyze methods for efficiently generating Gaussian vectors in feature spaces and study and apply kernel pre-image techniques – if that is possible in this scenario [Schölkopf and Smola, 2002; Bakir et al., 2003].

We also plan study one-class Support Vector Machines (SVM) [Schölkopf and Smola, 2002] for locality sensitive hashing. The first method separates data from the origin by solving the

following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{w},\xi,\rho} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + \frac{1}{\nu N}\sum_{i=1}^{N}\xi_i - \rho \\ \text{subject to} \quad & \boldsymbol{w}'\phi(\mathbf{x}_i) \geq \rho - \xi_i, \xi_i \geq 0, \nu \in (0,1] \end{aligned}$$

The other common method is used in novelty detection [Tax and Duin, 1999] but it is the same optimization problem that is applied in support vector clustering [Ben-Hur et al., 2001]. It finds smallest balls that contains all the training points. The primal problem to solved is formulated as follows:

$$\begin{aligned} \min_{R,\xi,\mathbf{a}} \quad & R^2 + \frac{1}{\nu N}\sum_{i=1}^{N}\xi_i \\ \text{subject to} \quad & \|\phi(\mathbf{x}_i) - \mathbf{a}\|^2 \leq R^2 + \xi_i, \xi_i \geq 0, \nu \in (0,1] \end{aligned}$$

A Proofs

A.1 Derivation of the inequalities

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 real vectors $\mathbf{c}$. It is strictly positive definite if $\mathbf{c}'\mathbf{K}\mathbf{c} = 0$ iff $\mathbf{c}$ is the all zero vector.

For deriving the inequalities the positive definite property is used.

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

$$d(\mathbf{x},\mathbf{z}) \geq k(\mathbf{x},\mathbf{x}) + k(\mathbf{z},\mathbf{z}) - 2\sqrt{k(\mathbf{x},\mathbf{x})k(\mathbf{z},\mathbf{z})} \tag{4}$$

The second inequality is obtain in a similar way. 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 $\mathbf{c}'\mathbf{K}\mathbf{c}$ is greater than zero. In this way we get

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

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

$$d(\mathbf{x},\mathbf{y}) + d(\mathbf{x},\mathbf{z}) - d(\mathbf{y},\mathbf{z}) \geq k(\mathbf{x},\mathbf{x}) - k(\mathbf{y},\mathbf{y}) - k(\mathbf{z},\mathbf{z}) \tag{5}$$

The triangle inequality tells us that

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

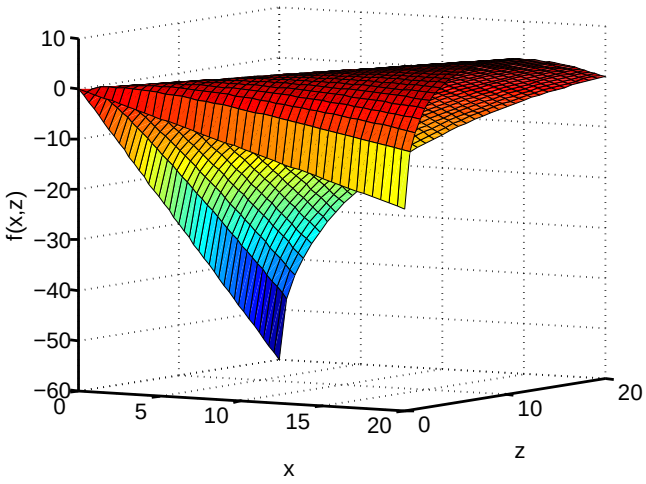

Figure 1: Plot of function $f(x,z) = -x - 3z + 4\sqrt{xz}$ in $[0,20]$.

from which

$$d(\mathbf{x},\mathbf{y}) + d(\mathbf{x},\mathbf{z}) - d(\mathbf{y},\mathbf{z}) \leq 2d(\mathbf{x},\mathbf{z}) \tag{6}$$

From (5) and (6) we are to obtain our second inequality, more precisely

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

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

$$d(\mathbf{x},\mathbf{z}) \geq \frac{1}{2}(k(\mathbf{x},\mathbf{x}) - k(\mathbf{z},\mathbf{z})) \tag{7}$$

From inequalities (4) and (7) we would like to use the more beneficial one. First, since (7) contains only an addition and a multiplication by a constant, it is our recommendation at first glance. Second, we want to find that formula which offers a lower upper bound on the Euclidean distance. This is because when comparing distances in kNN we want a tighter bound to filter out as many distance calculations as we may. Using the 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 (7) is in most cases smaller than the right side of (4). So, following our second argument the winner formula is again (7). For visual demonstration, on figure 1 the function $f(x,z) = -x - 3z + 4\sqrt{xz}$ was plot in the interval $[0,20]$.

A.2 Random Gaussian vectors in feature space

The problem to be solved is to generate Gaussian vectors in the feature space defined by a kernel. We follow here the idea from [Kulis and Grauman, 2009], where the random vector is constructed as a weighted sum of data set points. We define the mean of feature t vectors by $\mathbf{z}_t = \frac{1}{t}\sum_{i\in S}\phi(\mathbf{x}_i)$, where S is the index set of the t vectors. By the central limit theorem $\frac{1}{\sqrt{t}}\sum_{i\in S}\phi(\mathbf{x}_i)$ is distributed according to $\mathcal{N}(\mathbf{0},\Sigma)$, provided that the data is centered in the feature space. Here Σ is the unknown covariance matrix of the data. By using the whitening

transform the vector can be made to be distributed according to $\mathcal{N}(\mathbf{0}, \mathbf{I})$. Thus we need to compute the vector $\frac{1}{\sqrt{t}}\Sigma^{-\frac{1}{2}} \sum_{i \in S} \phi(\mathbf{x}_i)$. We know that the mapped data, the covariance and the kernel matrix can be written as

$$\begin{aligned} \Phi &= \mathbf{U}\mathbf{S}\mathbf{V}' \\ \Sigma &= \Phi\Phi' = \mathbf{U}\mathbf{S}^2\mathbf{U}' \\ \mathbf{K} &= \Phi'\Phi = \mathbf{V}\mathbf{S}^2\mathbf{V}' \end{aligned}$$

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

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

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

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

References

Andoni and Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *CACM: Communications of the ACM*, 51, 2008.

G. H. Bakir, J. Weston, and B. Schölkopf. Learning to find pre-images. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *NIPS*. MIT Press, 2003. ISBN 0-262-20152-6.

A. Ben-Hur, D. Horn, H. T. Siegelmann, and V. Vapnik. Support vector clustering. *Journal of Machine Learning Research*, 2:125–137, 2001. URL `http://www.jmlr.org/papers/v2/horn01a.html`.

M. Charikar. Similarity estimation techniques from rounding algorithms. In *STOC*, pages 380–388, 2002. URL `http://doi.acm.org/10.1145/509907.509965`.

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

B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *ICCV*, pages 2130–2137. IEEE, 2009. URL `http://dx.doi.org/10.1109/ICCV.2009.5459466`.

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

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

D. M. J. Tax and R. P. W. Duin. Data domain description using support vectors. In *ESANN*, pages 251–256, 1999. URL `http://www.dice.ucl.ac.be/Proceedings/esann/esannpdf/es1999-458.pdf`.