Blockcourse on High-Dimensional Computational Geometry Spring 2013

Lecture 8 — July 31, 2013

Wolfgang Mulzer & Helmut Alt *Scribe: Monica Blanco & Yannik Stein*

1 NN–Search by Locality Sensitive Hashing

Let's remind the definition of (r_1, r_2, p_1, p_2)–sensitivity:

Definition 1. *Let (X, D) be a metric space. A family $\mathcal{H} = \{h : X \to U\}$ is called (r_1, r_2, p_1, p_2)–sensitive if the following holds for all $x, y \in X$:*

$$D(x,y) \leq r_1 \Longrightarrow \underset{h\in\mathcal{H}}{Prob}\left[h(x) = h(y)\right] \geq p_1$$

$$D(x,y) \geq r_2 \Longrightarrow \underset{h\in\mathcal{H}}{Prob}\left[h(x) = h(y)\right] \leq p_2$$

with h chosen uniformly at random in $\mathcal{H}$.

For the near neighbour problem we take $r_1 = r$, $r_2 = cr$.

We now want to control the probabilities p_1 and p_2. We want the probability of ignoring a good point ($\geq 1 - p_1$) and the probability of accepting a bad point ($\leq p_2$) to be small. Hence we want p_1 to be close to 1 and p_2 to be close to 0.

Theorem 2. *Let (X, D) be a metric space and suppose that there exists a (r, cr, p_1, p_2)–sensitive family $\mathcal{H}$ of hash functions for such space.*
Set

$$\rho = \frac{log(\frac{1}{p_1})}{log(\frac{1}{p_2})}$$

and let $P \subseteq X$ have at most n points.

Then there exists a fully dynamic data structure for c–approximate r–NN over P:

- *A query requires $\Theta\big(\frac{n^\rho}{p_1}\big)$ distance computations and $\Theta(\frac{n^\rho}{p_1} \cdot log_{\frac{1}{p_2}} n)$ evaluations of hash functions.*
  The same for updates.
- *The space requirement is $\Theta\big(\frac{n^{1+\rho}}{p_1}\big)$ plus the space required to store P.*
- *The failure probability is at most $\frac{1}{3} + \frac{1}{e}$.*

Proof. Set

$$K = \lceil log_{\frac{1}{p_2}} n \rceil$$

$$L = \frac{n^\rho}{p_1}$$

We now define a new family of hash functions

$$\mathcal{G} = \{g_{I=(i_1,i_2,...,i_K)} = (h_{i_1}, h_{i_2}, ..., h_{i_k}) : X \to U^K \mid h_{i_j} \in \mathcal{H} \ \ \forall j = 1, ...k\}$$

where U^K is the cartesian product of the buckets, and $g_I \in \mathcal{G}$ is given by:

$$g_I(x) = g_{(i_1,i_2,...,i_K)}(x) = (h_{i_1}(x), h_{i_2}(x), ..., h_{i_k}(x)) \in U^K, \ \ x \in X$$

To take a random hash function in $\mathcal{G}$ is equivalent to taking K random hash functions in $\mathcal{H}$ and concatenating the results.

Since for all $x, y \in X$

$$\underset{g_I \in \mathcal{G}}{Prob}\left[g_I(x) = g_I(y)\right] =$$

$$= \underset{\substack{h_{i_j} \in \mathcal{H} \\ j=1,...,K}}{Prob}\left[h_{i_1}(x) = h_{i_1}(y) \ \wedge \ h_{i_2}(x) = h_{i_2}(y) \ \wedge \ ... \ \wedge \ h_{i_K}(x) = h_{i_K}(y)\right] \underset{*}{=}$$

$$= \prod_{j=1,..,K} \underset{h_{i_j} \in \mathcal{H}}{Prob}\left[h_{i_j}(x) = h_{i_j}(y)\right] \underset{**}{=}$$

$$= (\underset{h \in \mathcal{H}}{Prob}\left[h(x) = h(y)\right])^K$$

* independent events
** equally distributed

then we get that the family $\mathcal{G}$ is $(r, cr, \tilde{p_1}, \tilde{p_2})$–sensitive for (X, D), with $\tilde{p_1} = p_1^K$ and $\tilde{p_2} = p_2^K$: let $x, y \in X$, we have

$$D(x,y) \leq r_1 \Longrightarrow \underset{g \in \mathcal{G}}{Prob}\left[g(x) = g(y)\right] = (\underset{h \in \mathcal{H}}{Prob}\left[h(x) = h(y)\right])^K \geq p_1^K$$

$$D(x,y) \geq r_2 \Longrightarrow \underset{g \in \mathcal{G}}{Prob}\left[g(x) = g(y)\right] = (\underset{h \in \mathcal{H}}{Prob}\left[h(x) = h(y)\right])^K \leq p_2^K$$

This way, the ratio of probabilities:

$$\frac{\tilde{p_1}}{\tilde{p_2}} = \frac{p_1^K}{p_2^K} = (\frac{p_1}{p_2})^K$$

becomes larger.

To build now the Data Structure, we take L random hash functions $g_1, g_2, ..., g_L \in \mathcal{G}$, and we compute, for each $x \in P$, $g_1(x), g_2(x), ..., g_L(x)$, and store x in the appropriate bucket.

From here we derive the space storage conditions: we need at most

$$\underbrace{n}_{points\ in\ P} \cdot \underbrace{L}_{g_i\ for\ i=1,...,L} = n \cdot \frac{n^\rho}{p_1} = \frac{n^{\rho+1}}{p_1} = \Theta(\frac{n^{1+\rho}}{p_1})$$

To perform a query $q \in X$: evaluate $g_1(q), g_2(q), ..., g_L(q)$.
Let P_i, for each bucket containing a $g_i(q)$, be the points of P that are in the same bucket, i.e.

$$P_i = \{p \in P \mid g_i(p) = g_i(q)\}$$

Let $p \in P_i$ for some $i = 1, ..., L$. Compute the distance $D(p, q)$ and return p if this distance is at most cr.
Abort the algorithm and return $\perp$ if

- no such point is found or
- we have found $3L$ points in the P_i's with distance from q greater than cr.

From here we can derive the query time:

- At most $3L$ distance computations $\implies \Theta(L) = \Theta(\frac{n^\rho}{p_1})$
- $L \cdot K$ evaluations of hash functions $\implies \Theta(L \cdot K) = \Theta(\frac{n^\rho}{p_1} \cdot log_{\frac{1}{p_2}} n)$

Analysis: Fix query $q \in X$.

- If $D(q, P) > cr$, then we will not find a point p in the P_i's with $D(q, p) \leq cr$, and hence the algorithm will abort. Then in this case there is no probability of failure.
- Suppose now that $D(q, P) \leq cr$. Then the probability of failure here is divided in two:
  1. The probability that a good point p does not lie in any of the P_i's, and hence it is never taken.
     Let $p \in P$ with $D(q, p) \leq r$.

$$\underset{g \in \mathcal{G}}{Prob}\left[g(p) = g(q)\right] \geq p_1^K \geq p_1^{(log_{\frac{1}{p_2}} n)+1} = p_1 \cdot n^{\frac{logp_1}{log\frac{1}{p_2}}} = p_1 \cdot n^{-\frac{log\frac{1}{p_1}}{log\frac{1}{p_2}}} = p_1 \cdot n^{-\rho}$$

$$\implies \underset{g \in \mathcal{G}}{Prob}\left[g(p) \neq g(q)\right] \leq 1 - p_1 \cdot n^{-\rho}$$

$$\implies \underset{g_1,...,g_L \in \mathcal{G}}{Prob}\left[g_1(p) \neq g_1(q) \wedge \ ... \ \wedge \ g_L(p) \neq g_L(q)\right] \underset{*}{=} \prod_{i=1,..,L} \underset{g_i \in \mathcal{G}}{Prob}\left[g_i(p) \neq g_i(q)\right] \underset{**}{=}$$

$$= (\underset{g \in \mathcal{G}}{Prob}\left[g(p) \neq g(q)\right])^L \leq (1 - p_1 \cdot n^{-\rho})^L \leq e^{-p_1 n^{-\rho} L} = \frac{1}{e}$$

  2. The probability that a point $p \in P$ with $D(q, p) \leq cr$ lies in some P_i, but it is not taken after finding $3L$ bad points. This probability is the probability of finding at least $3L$ bad points in the P_i's.
     Let $p \in P$ be a point with $D(q, p) > cr$:

$$\underset{g \in \mathcal{G}}{Prob}\left[g(p) = g(q)\right] \leq p_2^K \leq p_2^{log_{\frac{1}{p_2}} n} = p_2^{\frac{logn}{log\frac{1}{p_2}}} = p_2^{-\frac{logn}{logp_2}} = n^{-\frac{logp_2}{logp_2}} = \frac{1}{n}$$

Then the probability that a certain bad point in P is in some bucket P_i is at most $\frac{L}{n}$:

$$\underset{g_1,...,g_L \in \mathcal{G}}{Prob}\left[g_1(p) = g_1(q) \vee \ ... \ \vee \ g_L(p) = g_L(q)\right] \underset{*}{=} \sum_{i=1,..,L} \underset{g_i \in \mathcal{G}}{Prob}\left[g_i(p) = g_i(q)\right] \underset{**}{=}$$

$$= L \cdot \underset{g \in \mathcal{G}}{Prob}\left[g(p) = g(q)\right] \leq \frac{L}{n}$$

Then we should expect to find at most L bad points in the L buckets P_i's:

$$E[\#\{p \in P \setminus B(q, cr) \ with \ g_i(p) = g_i(q) \ for \ some \ i \in \{1, ..., L\}\}] =$$

$$= \sum_{p \in P} \underset{g_1,...,g_L \in \mathcal{G}}{Prob}\left[g_1(p) = g_1(q) \vee \ ... \ \vee \ g_L(p) = g_L(q)\right] \underset{**}{\leq} \sum_{p \in P} \frac{L}{n} = n \cdot \frac{L}{n} = L$$

By Markov's Inequality,

$$\underset{g_1,...,g_L \in \mathcal{G}}{Prob}[\#\{p \in P \setminus B(q, cr) \ with \ g_i(p) = g_i(q) \ for \ some \ i \in \{1, ..., L\}\} \geq 3L] \leq \frac{1}{3}$$

So the probability of finding at least $3L$ bad points in the P_i's is at most $\frac{1}{3}$.

And hence the probability of failure is at most $\frac{1}{3} + \frac{1}{e}$

□

2 Examples

Examples of LSH families:

- Hamming metric

  Metric space (X, D) with $X = \{0, 1\}^d$ and $D(x, y) = \#\{\text{coordinates in which } x \text{ and } y \text{ differ}\}$, for $x, y \in X$.
  Define

  $$h_i(x = (x_1, ..., x_d)) = x_i$$

  as the projection onto the i–th coordinate of x, for $i = 1, ...d$, and let

  $$\mathcal{H} = \{h_1, ..., h_d\}$$

  be the family of hash functions.

  Claim 3. *$\mathcal{H}$ is $(r, cr, 1 - \frac{r}{d}, 1 - \frac{cr}{d})$–sensitive.*

  Proof. Let $x, y \in X$.

– If $D(x,y) \leq r \implies$ at most r coordinates are different in x and y
$\implies$ at least $d-r$ coordinates are the same in x and y
$\implies$ the probability that a random bit is the same is $\geq \frac{1}{d} \cdot (d-r) = 1 - \frac{r}{d}$

– If $D(x,y) \geq cr \implies$ at least cr coordinates are different in x and y
$\implies$ at most $d-cr$ coordinates are the same in x and y
$\implies$ the probability that a random bit is the same is $\leq \frac{1}{d} \cdot (d-cr) = 1 - \frac{cr}{d}$

□

Fact 4. *We can go from l_2 (euclidean) to l_1 (Manhattan) without losing too much. The same way, we can go from l_1 to D (Hamming).*

- LSH family for euclidean metric in $\mathbb{R}^d$, a fixed parameter r to be determined

We define

$$\mathcal{H} = \{h_{\vec{v},t} \mid \vec{v} \in \mathbb{R}^d \text{ is a vector and } t \in [0,r]\}$$

Where $h_{\vec{v},t} : \mathbb{R}^d \to \mathbb{N}$ is defined as:

$$h_{\vec{v},t}(p) = \lfloor \frac{< \vec{v}, p > + t}{r} \rfloor$$

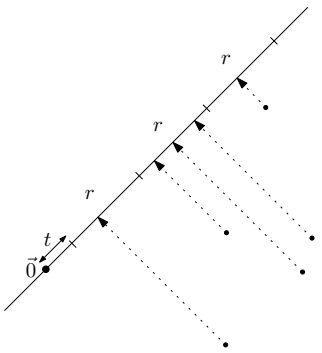

Figure 1: example of $h_{\vec{v},t}$ for two-dimensional input.

See figure 1 for an example.

The randomness of picking $h \in \mathcal{H}$ is determined by:

– Taking $t \in [0,r]$ with the uniform distribution $U[0,r]$
– Taking $\vec{v} \in \mathbb{R}^d$ with the normal distribution in dimension d, $\aleph^d(0,1)$

The idea of how this works is that close points will lie close together in any line (hash function), and that distant points will be together on the line for just few lines. See figure 2 for an example.

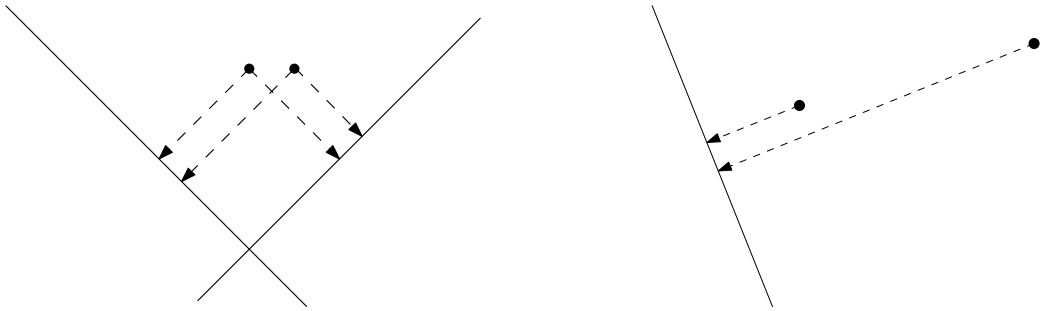

Figure 2: close points projected onto any hyperplane are again close. Depending on the hyperplane, far away points can also be close after projection.

Fact 5. *The parameter r can be picked such that $\mathcal{H}$ is $(1, 1+\epsilon, p_1, p_2)$–sensitive with*

$$\rho = \frac{1}{1+\epsilon}$$

With this family we can perform an $(1+\epsilon)$–approximate NN search with

- *Space* $\Theta(n^{1+\frac{1}{1+\epsilon}}\ poly\ log)$
- *Query time* $\Theta(n^{\frac{1}{1+\epsilon}}\ poly\ log)$

In 2006, this was improved to

- *Space* $\Theta(n^{1+\frac{1}{(1+\epsilon)^2}}\ poly\ log)$
- *Query time* $\Theta(n^{\frac{1}{(1+\epsilon)^2}}\ poly\ log)$

3 Low distortion embedding of finite metric spaces

We start with a motivation by applications in pattern recognition and analysis: *objects* that are represented by some of their *features* have to be recognized/analyzed.

object: e.g., image, geometric shape, character, text

feature: e.g., ad hoc histogram, avg. color, #e's in text, or more sophisticated: certain frequencies of Fourier transformation

By *feature extraction*, we mean the mapping of objects to *feature vectors*. The aim in pattern recognition is to *classify* objects, e.g., in character recognition:

$$\text{object b/w pixel image} \to \text{class of a finite set of classes}$$

A *pattern recognition algorithm* has the following scheme

$$\text{Object} \underset{\text{feature extraction}}{\longrightarrow} \text{feature vector} \underset{\text{classification}}{\longrightarrow} \text{class}$$

3.1 Classification

3.1.1 Nearest Neighbor Classification

For each class, a prototype feature vector $v \in \mathbb{R}^d$ is given. A query vector $q \in \mathbb{R}^d$ is classified by the "nearest neighbor" among the prototypes. This is called *NN-classification*.

We obtain prototypes either directly or by *k-NN classification* in the *learning phase/training phase* ($k \in \mathbb{N}$ is chosen experimentally):

1. The input is a set of feature vectors F, which contains several vectors per class.
2. Given a query vector q, we search in F for the k nearest neighbors. The class of q is then the most frequent class of the k nearest neighbors.

This is also called *supervised learning*.

3.1.2 Linear Discrimant Analysis (LDA)

We use a large sample of feature vectors with specified classes as training set to find linear constraints that can be used to classify query points. For two classes, we can try to find a hyperplane that separates the classes "well". See figure 3 for an example.

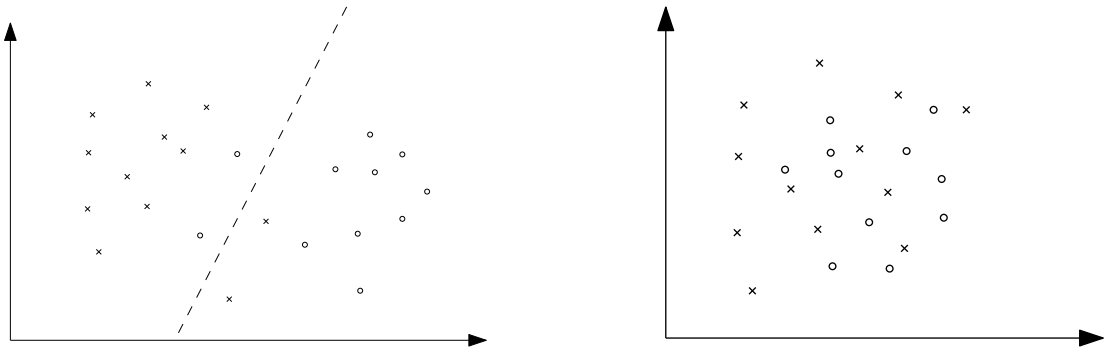

Figure 3: example of LDA with two classes. In the left figure, we can find a hyperplane that separates both classes "well". In the right figure, this is not possible.

What is done: We assume the feature vectors for each class to be distributed according to the normal distribution, where the parameters are (approximately) determined by the samples. We then try to find a hyperplane that separates the normal distributions well.

If there are many classes, we can apply

- statistical methods: set of hyperplanes separating the normal distributions well
- all $\binom{c}{2}$ separating hyperplanes of all pairs that are somehow combined
- one against the rest: recursively apply 2–class technique to one normal distribution and the rest

or *dimension reduction*: project all points into a subspace with low distortion, i.e., relative distances are preserved approximately.

3.2 Feature Selection

The problem is to find d "good" features among n possible ones. For example in character recognition, we have to find possible feature values of b/w pixels $\in \{0, 1\}$. Let $P \subset \mathbb{R}^n$ be a training set with all n features. We can try to find good features by searching a d–dimensional subspace which approximately preserves relative distances and thus simplifies classification (figure 4).

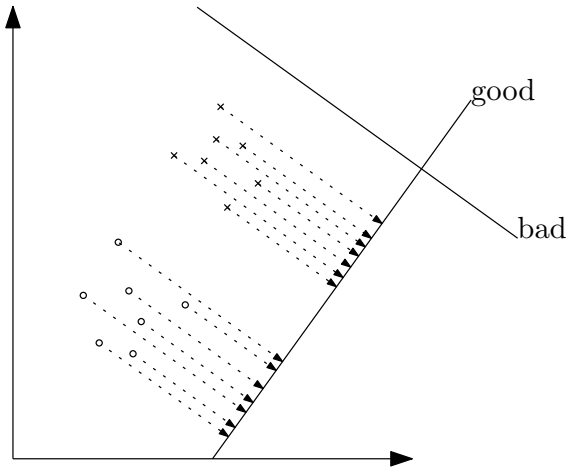

Figure 4: projection of the training set onto a 1–dimensional subspace.