Finding Nearby Neighbors and Locality-Sensitive Hashing

A problem commonly appearing in multiple areas of data analysis is the problem of finding neighbors in high dimensional data. Consider a set $S = \{p_1 \ldots p_n\}$, of points, where each point, all taken from a high-dimensional space. Let $R$ be a given parameter. After preprocessing the points, we would like to answer queries such as

1. given a query point $q$, find the nearest point of $S$. That is, the point $p_i \in S$ whose distance from $q$ is the smallest.
2. given a query point $q$, find all the points of $S$ which are of distance $\leq R$.
3. etc

Two useful techniques, commonly used together are projections on small number of lines, and bucketing (hashing) on each line.

Rather than working in $\mathbb{R}^d$, it is of course much more convenient to work with points on a one-dimensional line. Let $\ell$ be a line (we discuss later how to pick it). Let $p'_i = \pi_\ell(p_i)$ be the orthogonal projection of $p_i$ on $\ell$ and let $S' = \{p'_1 \ldots p'_n\}$. Now let $q$ be a query point, and let $q' = \pi_\ell(q)$. Consider points to be “near” $q$ if their distance is $\leq R$ and far otherwise. Obviously $\|p - q\| \geq \|p' - q'\|$. Meaning that if $\|p'_i - q'\| \geq R$ (their projection are far), then we are assume that $\|p_i - q\| \geq R$ (the original point is far, and $p_i$ is filter out from further consideration. However, using more than a single line ($\ell_1$ and $\ell_2$ in this example), we could filter out a point $p_i$ because its projection on one of the lines $\ell_1$ or $\ell_2$ are too far from the projection of $q$ on these lines. For example, the projections on $\ell_2$ of $p_4, p_5$ and $q$ are close to each other. However, when projected on $\ell_1$, we see that $q$ must be quite far from $p_4$, since their projections on $\ell_1$ are distant.

To expedite the search for nearby points near $q'$s projection on each of the lines, a standard bucketing and hashing techniques could be used for each line. Namly, for each line $\ell$, we assign disjoint ‘buckets’ on $\ell$. The length of each bucket is $R$. (See Fig 1). For each bucket, we store which points of $S$ are projected to this bucket. For example, on $\ell_2$, both $p_4$ and $p_5$ are stored in bucket $b_3$. Since the number of buckets is unbounded, we must use hashing to map each location on $\ell$ to a bucket, using the same hashing technique we used in the closest-pair algorithm.

An obvious questions is how frequently do we have ‘false negative’ where points which are far from each other while their projections are close. Homework 4 indicates that the answer is ‘not too frequently’. If we pick the orientation of the line $\ell$ at random, and declare a pair of points $p, q$ to be ‘near’ (with respect to $\ell$) if the distance $\|\pi_\ell(p) - \pi_\ell(q)\|$ between their projections on $\ell$ is $\geq \|p - q\|/2$. The homework indicates that a random line will preserve these far and near relationships for many of the pairs of the points (and formal discussion here is outside the focus of the course).
**Computing the projection of a point on a line.** (this part is not required for the exam).

Finally, assume a a line $\ell$ is given, and assume the points $\vec{p}_0$ and $\vec{p}_1$ are on $\ell$. See Figure 2 Right. For simplicity, assume that $\vec{p}_0$ is the origin $(0,0)$. We are also given another point $\vec{q} \not\in \ell$. Then we could find the point $\vec{p}^* \in \ell$ which is the closest point to $\vec{q}$. This is exactly the orthogonal projection of $\vec{q}$ on $\ell$. Note that the the vectors $\vec{p}^*$ (the vector emerging from $\vec{p}_0$ toward $\vec{p}^*$) and the vectors $\vec{q} - \vec{p}^*$ (the vector emerging from $\vec{p}^*$ toward $\vec{q}$) are orthogonal to each other.

We know that each point $\vec{p}$ on $\ell$ could bd described as $\vec{p} = t \cdot \vec{p}_1$, for some scalar $t$ (time). You could imaging a car driving from $\vec{p}_0$ toward $\vec{p}_1$, moving at speed of $|\vec{p}_1|$ (the distance between $\vec{p}_0$ and $\vec{p}_1$) miles per hour. After one hour ($t = 1$) this car would be at $\vec{p}_1$. After 2 hours ($t = 2$) this car would be on $\ell$, and its distance from $\vec{p}_0$ is 2 times $|\vec{p}_1|$ and so on. Hence, to find $\vec{p}^*$, we only need to find at which time $t^*$ this car would be at $\vec{p}^*$. Then $\vec{p}^* = t^* \cdot \vec{p}_0$.

![Figure 2:](image)

Note that for any pair of non-zero vectors $\vec{v}, \vec{u}$, it is known that

$$\vec{v} \cdot \vec{u} = |\vec{v}| \cdot |\vec{u}| \cos(\beta)$$

Where $\vec{v} \cdot \vec{u}$ is the dot product of $\vec{v}$ and $\vec{u}$, and $\beta$ is the angle between them. See Fig 2 left. Hence, $\vec{v} \cdot \vec{u} = 0$ iff $\vec{u} \perp \vec{v}$. So $t^* \cdot \vec{p}_1$ is orthogonal to $\vec{q} - t^* \vec{p}_1$. Or

$$(t^* \cdot \vec{p}_1) \cdot (\vec{q} - t^* \vec{p}_1) = 0 \quad \text{or} \quad t^* = \left\{ \frac{\vec{p} \cdot \vec{q}}{\vec{p} \cdot \vec{p}} \right\}$$