Nearest neighbor search

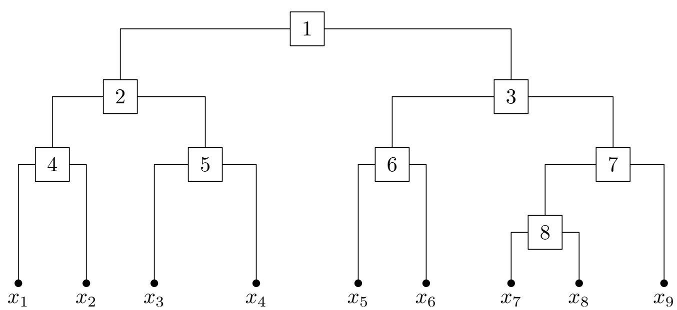
Nearest neighbor search

- Naïve implementation of NN classifiers based on n labeled examples requires n distance computations to compute the prediction on any test point $\boldsymbol{x} \in \mathcal{X}$.
  - If using Euclidean distance in $\mathbb{R}^d$, then each distance computation is $O(d)$ operations.

  $\Longrightarrow O(dn)$ operations per test point.
- **Solution**: store the labeled examples in a special data structure that permits fast NN queries.

Tree structures for one-dimensional data

A data structure for fast NN search in $\mathbb{R}^1$
Sort training data so that $x_1 \leq x_2 \leq \cdots \leq x_n$, then construct binary tree:



With each tree node, remember **midpoint** between rightmost point in left child, and leftmost point in right child. **This permits very efficient NN search.**

If tree is (approximately) balanced, then $O(\log(n))$ time to find NN!

Tree structures for multi-dimensional data

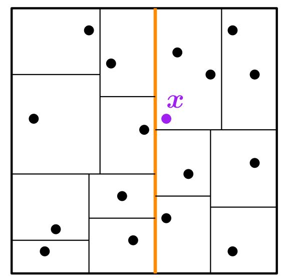
A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

Construction procedure
Given points $S \subset \mathbb{R}^d$:

1. Pick a coordinate $j \in \{1, 2, \ldots, d\}$.
2. Let m be the median of $\{x_j : \boldsymbol{x} \in S\}$.
3. Split points into halves:

$$L := \{\boldsymbol{x} \in S : x_j < m\},$$
$$R := \{\boldsymbol{x} \in S : x_j \geq m\}.$$

4. Recurse on L and R.



Easy to lookup points in S (in $O(\log(n))$ time).

What about new points (not in S)?

Same $O(\log(n))$-time routing of a test point $\boldsymbol{x} \in \mathbb{R}^d$ (called *defeatest search*) is **overly optimistic**: might not yield the NN!

Searching general tree structures

Generic NN search procedure for binary space partition trees
Given a test point $\boldsymbol{x}$ and a tree node v (initially $v =$ root):

1. Pick one child L, recursively find NN of $\boldsymbol{x}$ in L (call it $\boldsymbol{x}_L$).
2. Let R be the other child. If

$$\|\boldsymbol{x} - \boldsymbol{x}_L\|_2 < \min_{\boldsymbol{x}' \in R} \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \qquad (\star)$$

   then return $\boldsymbol{x}_L$.
3. Otherwise recursively find NN of $\boldsymbol{x}$ in R (call it $\boldsymbol{x}_R$); return the closer of $\boldsymbol{x}_L$ and $\boldsymbol{x}_R$.

Note: can't always guarantee $O(\log(n))$ search time due to Step 3.

Question: How do you check if $(\star)$ is true?

- **Note**: it is okay (though wasteful) to declare "false" in Step 2 even if $(\star)$ turns out to be true.

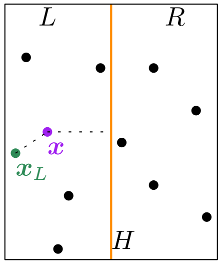
Using geometric properties

For K-D trees:
L and R are separated by a hyperplane $H = \left\{ \boldsymbol{z} \in \mathbb{R}^d : z_j = m \right\}$.

Suppose test point $\boldsymbol{x}$ is in L, and the NN of $\boldsymbol{x}$ in L is $\boldsymbol{x}_L$.

By geometry,

$$\min_{\boldsymbol{x}' \in R} \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \geq \text{distance from } \boldsymbol{x} \text{ to } H = |x_j - m| .$$



A valid check: if $\|\boldsymbol{x} - \boldsymbol{x}_L\|_2 < |x_j - m|$, then

$$\|\boldsymbol{x} - \boldsymbol{x}_L\|_2 < \min_{\boldsymbol{x}' \in R} \|\boldsymbol{x} - \boldsymbol{x}'\|_2 .$$

In this case, we can skip searching R and immediately return $\boldsymbol{x}_L$.

Efficient NN search?

For certain kinds of binary space partition trees (similar to K-D trees), enough pruning will happen so NN search typically completes in $O(2^d \log(n))$ time.

- Very fast in low dimensions.
- But can be slow in high dimensions.

But NN search is only means to an end—ultimate goal is good classification.
K-D tree construction doesn't even look at the labels!

Question: Can we use trees to directly build good classifiers?

Key takeaways

1. Efficient data structure for NN search in $\mathbb{R}^1$.
2. Construction of K-D trees in $\mathbb{R}^d$, $d > 1$.
3. NN search in K-D trees.