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 \( x \in \mathcal{X} \).

- If using Euclidean distance in \( \mathbb{R}^d \), then each distance computation is \( O(d) \) operations.

\[ \Rightarrow O(dn) \text{ operations per test point.} \]

- **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:

```
1
2 3
4 5 6 7
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8 x_9
```

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 \( \{x_j : x \in S\} \).
3. Split points into halves:

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

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 \( x \in \mathbb{R}^d \) 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 $x$ and a tree node $v$ (initially $v = \text{root}$):

1. Pick most optimistic child $L$, recursively find NN of $x$ in $L$ (call it $x_L$).
2. Let $R$ be the other child. If $\|x - x_L\|_2 < \min_{x' \in R} \|x - x'\|_2$ (*)
   then return $x_L$.
3. Otherwise recursively find NN of $x$ in $R$ (call it $x_R$); return the closer of $x_L$ and $x_R$.

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

Question: How do you check if (*) is true?

▶ Note: it’s correct (though computationally wasteful) to declare “false” in Step 2 even if (*) turns out to be true.

Using geometric properties

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

Suppose test point $x$ is in $L$, and the NN of $x$ in $L$ is $x_L$.

By geometry,
\[
\min_{x' \in R} \|x - x'\|_2 \geq \text{distance from } x \text{ to } H = |x_j - m|.
\]

A valid check: if $\|x - x_L\|_2 < |x_j - m|$, then
\[
\|x - x_L\|_2 < \min_{x' \in R} \|x - x'\|_2.
\]

In this case, we can skip searching $R$ and immediately return $x_L$.

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?