COMS 4771 Lecture 3

1. Nearest neighbor classification.

Nearest neighbor classification

THE OPTIMAL CLASSIFIER

Let $(X, Y) \sim P$.
The classifier $f \colon \mathcal{X} \to \mathcal{Y}$ with the smallest prediction error

$$\operatorname{err}(f) = \Pr[f(X) \neq Y]$$

is the **Bayes classifier**

$$f^{\star}(x) = \operatorname*{arg\,max}_{y \in \mathcal{Y}} \Pr\Big[Y = y \,\Big|\, X = x\Big].$$

The optimal classifier

Let $(X, Y) \sim P$.
The classifier $f : \mathcal{X} \to \mathcal{Y}$ with the smallest prediction error

$$\mathrm{err}(f) = \Pr[f(X) \neq Y]$$

is the **Bayes classifier**

$$f^\star(x) = \operatorname*{arg\,max}_{y \in \mathcal{Y}} \Pr\Big[Y = y \,\Big|\, X = x\Big].$$

Of course, we don't know P, and hence can't generally get a handle of $f^\star$.

The optimal classifier

Let $(X, Y) \sim P$.
The classifier $f\colon \mathcal{X} \to \mathcal{Y}$ with the smallest prediction error

$$\mathrm{err}(f) = \Pr[f(X) \neq Y]$$

is the **Bayes classifier**

$$f^\star(x) = \operatorname*{arg\,max}_{y \in \mathcal{Y}} \Pr\Big[Y = y \,\Big|\, X = x\Big].$$

Of course, we don't know P, and hence can't generally get a handle of $f^\star$.
What can we do?

- **Last lecture**: use "generative" models to approximate $\Pr[Y = y | X = x]$.

THE OPTIMAL CLASSIFIER

Let $(X, Y) \sim P$.
The classifier $f : \mathcal{X} \to \mathcal{Y}$ with the smallest prediction error

$$\mathrm{err}(f) = \Pr[f(X) \neq Y]$$

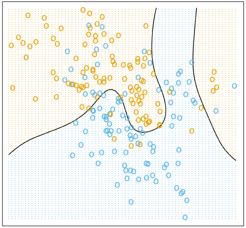
is the **Bayes classifier**

$$f^\star(x) = \arg\max_{y \in \mathcal{Y}} \Pr\Big[Y = y \,\Big|\, X = x\Big].$$

Of course, we don't know P, and hence can't generally get a handle of $f^\star$.
What can we do?

- **Last lecture**: use "generative" models to approximate $\Pr[Y = y | X = x]$.
- **This lecture**: directly approximate the **decision boundaries** of $f^\star$.

discriminative model



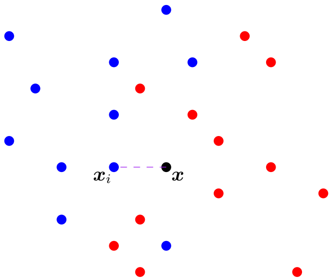
Text

Nearest neighbor (NN) classifier

Given **training data** $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, construct $\hat{f}\colon \mathcal{X} \to \mathcal{Y}$ as follows:

On input $\boldsymbol{x}$,

1. Let $\boldsymbol{x}_i$ be the point among $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ that is closest to $\boldsymbol{x}$.
2. Return y_i.

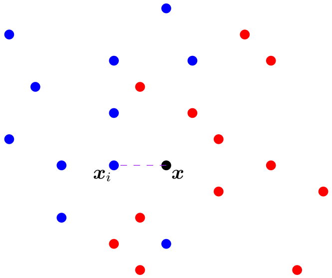


Nearest neighbor (NN) classifier

Given **training data** $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, construct $\hat{f} \colon \mathcal{X} \to \mathcal{Y}$ as follows:

On input $\boldsymbol{x}$,

1. Let $\boldsymbol{x}_i$ be the point among $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ that is closest to $\boldsymbol{x}$.
2. Return y_i.



Question: how should we measure distance between points in $\mathcal{X}$?

A default choice of distance for data in $\mathbb{R}^d$:

$$\text{Euclidean } (\ell_2) \text{ distance}: \quad \|\boldsymbol{u} - \boldsymbol{v}\|_2 := \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2}.$$

A default choice of distance for data in $\mathbb{R}^d$:

$$\text{Euclidean } (\ell_2) \text{ distance}: \quad \|\boldsymbol{u} - \boldsymbol{v}\|_2 := \sqrt{\sum_{i=1}^{d} (u_i - v_i)^2}.$$

But there are many other (and often better) options ...

Example: OCR with NN classifier

- **Handwritten digits data**: grayscale 28×28 images, treated as vectors in $\mathbb{R}^{784}$, with labels indicating the digit they represent.

Example: OCR with NN classifier

- **Handwritten digits data**: grayscale 28×28 images, treated as vectors in $\mathbb{R}^{784}$, with labels indicating the digit they represent.

- Split into training data S (60000 points) and test data T (10000 points).

Example: OCR with NN classifier

- **Handwritten digits data**: grayscale 28×28 images, treated as vectors in $\mathbb{R}^{784}$, with labels indicating the digit they represent.

- Split into training data S (60000 points) and test data T (10000 points).
- **Training error**: $\text{err}(\hat{f}, S) = 0$

Example: OCR with NN classifier

- **Handwritten digits data**: grayscale 28×28 images, treated as vectors in $\mathbb{R}^{784}$, with labels indicating the digit they represent.

- Split into training data S (60000 points) and test data T (10000 points).
- **Training error**: $\mathrm{err}(\hat{f}, S) = 0$
  Test error: $\mathrm{err}(\hat{f}, T) = 0.0309$
- Examples of mistakes (test point in T, nearest neighbor in S):

Example: OCR with NN classifier

- **Handwritten digits data**: grayscale 28×28 images, treated as vectors in $\mathbb{R}^{784}$, with labels indicating the digit they represent.

- Split into training data S (60000 points) and test data T (10000 points).
- **Training error**: $\mathrm{err}(\hat{f}, S) = 0$

  Test error: $\mathrm{err}(\hat{f}, T) = 0.0309$
- Examples of mistakes (test point in T, nearest neighbor in S):

- **Observation**: First mistake (correct label is '2') might've been avoided by looking at three nearest neighbors (whose labels are '8', '2', '2') . . .

test point three nearest neighbors

k-NEAREST NEIGHBORS CLASSIFIER

Given **training data** $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, construct $\hat{f}_k \colon \mathcal{X} \to \mathcal{Y}$ as follows:

On input $\boldsymbol{x}$,

1. Let $\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \ldots, \boldsymbol{x}_{i_k}$ be the k points among $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ that are closest to $\boldsymbol{x}$.
2. Return the plurality of $y_{i_1}, y_{i_2}, \ldots, y_{i_k}$.

(Break ties in both steps arbitrarily.)

k-NEAREST NEIGHBORS CLASSIFIER

Given **training data** $(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \ldots, (\boldsymbol{x}_n, y_n) \in \mathcal{X} \times \mathcal{Y}$, construct $\hat{f}_k \colon \mathcal{X} \to \mathcal{Y}$ as follows:

On input $\boldsymbol{x}$,

1. Let $\boldsymbol{x}_{i_1}, \boldsymbol{x}_{i_2}, \ldots, \boldsymbol{x}_{i_k}$ be the k points among $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ that are closest to $\boldsymbol{x}$.
2. Return the plurality of $y_{i_1}, y_{i_2}, \ldots, y_{i_k}$.

(Break ties in both steps arbitrarily.)

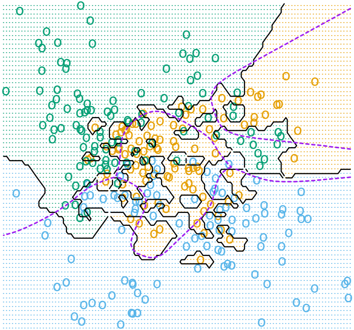
Example: OCR with k-NN classifier

k	1	3	5	7	9
$\mathrm{err}(\hat{f}_k, T)$	0.0309	0.0295	0.0312	0.0306	0.0341

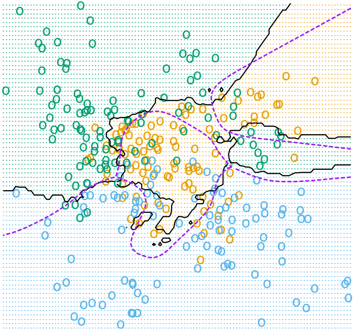
Effect of k

In general:

- Smaller $k \Rightarrow$ smaller training error. ($k = 1 \Rightarrow$ has zero training error.)
- Larger $k \Rightarrow$ predictions are more "stable" due to voting.



1-NN



15-NN

Purple dotted lines: Bayes classifier's decision boundaries.
Black solid lines: k-NN's decision boundaries.

Choosing k

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1, 3, 5, 7, 9\}$)?

Choosing k

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1, 3, 5, 7, 9\}$)?

- **Minimizer of test error**: $\hat{k} := \arg\min_k \mathrm{err}(\hat{f}_k, T)$.

Choosing k

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1, 3, 5, 7, 9\}$)?

- **Minimizer of test error**: $\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_k, T)$.

  Caveat: $\hat{f}_{\hat{k}}$ is no longer independent of T!

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1, 3, 5, 7, 9\}$)?

- **Minimizer of test error**: $\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_k, T)$.

  Caveat: $\hat{f}_{\hat{k}}$ is no longer independent of T!

  $\implies$ Test error is not an unbiased estimate of true error of $\hat{f}_{\hat{k}}$.

CHOOSING k

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1, 3, 5, 7, 9\}$)?

- **Minimizer of test error**: $\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_k, T)$.

  Caveat: $\hat{f}_{\hat{k}}$ is no longer independent of T!

  $\Longrightarrow$ Test error is not an unbiased estimate of true error of $\hat{f}_{\hat{k}}$.

- **Better alternatives**: For any set of labeled examples $A \subseteq \mathcal{X} \times \mathcal{Y}$, define $\hat{f}_{(A,k)}$ to be the k-NN classifier that searches for neighbors in A.

  1. **Minimizer of hold-out error**: fix $H \subseteq S$,

  $$\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_{(S \setminus H, k)}, H).$$

Choosing k

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1, 3, 5, 7, 9\}$)?

- **Minimizer of test error**: $\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_k, T)$.

  Caveat: $\hat{f}_{\hat{k}}$ is no longer independent of T!

  $\Longrightarrow$ Test error is not an unbiased estimate of true error of $\hat{f}_{\hat{k}}$.

- **Better alternatives**: For any set of labeled examples $A \subseteq \mathcal{X} \times \mathcal{Y}$, define $\hat{f}_{(A,k)}$ to be the k-NN classifier that searches for neighbors in A.

  1. **Minimizer of hold-out error**: fix $H \subseteq S$,

  $$\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_{(S \setminus H, k)}, H).$$

  2. **Minimizer of leave-one-out cross-validation error**:

  $$\hat{k} := \arg\min_k \frac{1}{|S|} \sum_{(x,y) \in S} \operatorname{err}(\hat{f}_{(S \setminus \{(x,y)\}, k)}, \{(x, y)\}).$$

Choosing k

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1, 3, 5, 7, 9\}$)?

- **Minimizer of test error**: $\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_k, T)$.

  Caveat: $\hat{f}_{\hat{k}}$ is no longer independent of T!

  $\Longrightarrow$ Test error is not an unbiased estimate of true error of $\hat{f}_{\hat{k}}$.

- **Better alternatives**: For any set of labeled examples $A \subseteq \mathcal{X} \times \mathcal{Y}$, define $\hat{f}_{(A,k)}$ to be the k-NN classifier that searches for neighbors in A.

  1. **Minimizer of hold-out error**: fix $H \subseteq S$,

  $$\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_{(S \setminus H, k)}, H).$$

  2. **Minimizer of leave-one-out cross-validation error**:

  $$\hat{k} := \arg\min_k \frac{1}{|S|} \sum_{(x,y) \in S} \operatorname{err}(\hat{f}_{(S \setminus \{(x,y)\}, k)}, \{(x, y)\}).$$

  $\Longrightarrow$ Test error is an unbiased estimate of true error of $\hat{f}_{\hat{k}}$.

Choosing k

Question: how do we choose k (say, from some subset of $\mathbb{N}$ like $\{1,3,5,7,9\}$)?

- **Minimizer of test error**: $\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_k, T)$.

Caveat: $\hat{f}_{\hat{k}}$ is no longer independent of T!

$\Longrightarrow$ Test error is not an unbiased estimate of true error of $\hat{f}_{\hat{k}}$.

- **Better alternatives**: For any set of labeled examples $A \subseteq \mathcal{X} \times \mathcal{Y}$, define $\hat{f}_{(A,k)}$ to be the k-NN classifier that searches for neighbors in A.

1. **Minimizer of hold-out error**: fix $H \subseteq S$,

the difference is at k's value!!!

$$\hat{k} := \arg\min_k \operatorname{err}(\hat{f}_{(S\setminus H,k)}, H).$$

hold one point out for test

2. **Minimizer of leave-one-out cross-validation error**:

$$\hat{k} := \arg\min_k \frac{1}{|S|} \sum_{(x,y)\in S} \operatorname{err}(\hat{f}_{(S\setminus\{(x,y)\},k)}, \{(x,y)\}).$$

$\Longrightarrow$ Test error is an unbiased estimate of true error of $\hat{f}_{\hat{k}}$.

More on this later in the course.

Consistency of k-NN

Say a learning algorithm is **consistent** if

$$\lim_{n \to \infty} \mathbb{E}\Big[\text{error of learned classifier with training sample size } n\Big] \;=\; \mathrm{err}(f^{\star}).$$

Consistency of k-NN

Say a learning algorithm is **consistent** if

the best classifier!

$$\lim_{n\to\infty} \mathbb{E}\Big[\text{error of learned classifier with } \boxed{\text{training sample size } n}\Big] = \boxed{\operatorname{err}(f^\star).}$$

k-**NN is consistent** provided that $k := k_n$ is chosen as an increasing but sublinear function of n:

$$\lim_{n\to\infty} k_n = \infty, \quad \lim_{n\to\infty} \frac{k_n}{n} = 0$$

(some other mild conditions might also have to hold).

as the size of training sample grows up. the error of the learned classifier getting closer to err(f).

Consistency of k-NN

Say a learning algorithm is **consistent** if

$$\lim_{n\to\infty} \mathbb{E}\Big[\text{error of learned classifier with training sample size } n\Big] \;=\; \mathrm{err}(f^\star).$$

k-**NN is consistent** provided that $k := k_n$ is chosen as an increasing but sublinear function of n:

$$\lim_{n\to\infty} k_n = \infty, \quad \lim_{n\to\infty} \frac{k_n}{n} = 0$$

(some other mild conditions might also have to hold).

1-**NN is not consistent unless** $\boxed{\mathrm{err}(f^\star) = 0,}$ although

$$\lim_{n\to\infty} \mathbb{E}\Big[\mathrm{err}(\hat{f}_1)\Big] \;\le\; 2\,\mathrm{err}(f^\star)\cdot\left(1 - \frac{K}{2(K-1)}\,\mathrm{err}(f^\star)\right).$$

Nearest neighbor search

- Naïve implementation of NN classifiers uses n distance computations to compute $\hat{f}_k(\boldsymbol{x})$ for any test point $\boldsymbol{x} \in \mathcal{X}$.

NEAREST NEIGHBOR SEARCH

- Naïve implementation of NN classifiers uses n distance computations to compute $\hat{f}_k(\boldsymbol{x})$ for 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.

Nearest neighbor search

- Naïve implementation of NN classifiers uses n distance computations to compute $\hat{f}_k(\boldsymbol{x})$ for 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.
- **Alternatives**:
  1. Settle for an approximate nearest neighbor using *locality sensitive hash functions*.
  2. Store the n training data in a geometric data structure that permits fast NN queries.

Locality sensitive hash functions

(Informally:) A family $\mathcal{H}$ of hash functions from $\mathbb{R}^d$ to $\mathbb{Z}$ is a **locality-sensitive hash family** if

- For any points $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathcal{X}$ with $\|\boldsymbol{a} - \boldsymbol{b}\|_2 \ll \|\boldsymbol{a} - \boldsymbol{c}\|_2$,

$$|\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{b})\}| \;\gg\; |\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{c})\}|.$$

(Informally:) A family $\mathcal{H}$ of hash functions from $\mathbb{R}^d$ to $\mathbb{Z}$ is a **locality-sensitive hash family** if

- For any points $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathcal{X}$ with $\|\boldsymbol{a} - \boldsymbol{b}\|_2 \ll \|\boldsymbol{a} - \boldsymbol{c}\|_2$,

$$|\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{b})\}| \;\gg\; |\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{c})\}|.$$

It turns out there are such hash families!

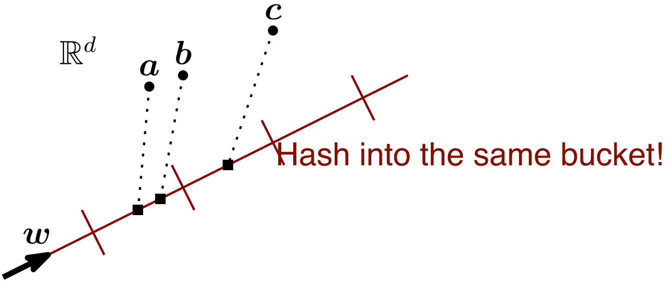
A locality sensitive hash family

A LSH family based on projections to one-dimensional subspaces

For $\boldsymbol{w} \in \mathbb{R}^d$ with $\|\boldsymbol{w}\|_2 = 1$, $r \in \{2^i : i \in \mathbb{Z}\}$, $s \in [0, r]$:

$$h_{\boldsymbol{w},r,s}(\boldsymbol{x}) := \left\lfloor \frac{\boldsymbol{w}^\top \boldsymbol{x} + s}{r} \right\rfloor .$$

- $\boldsymbol{w}$ determines the one-dimensional subspace,
- r determines a distance resolution, and
- s determines a shift of the bucket boundaries.



Locality sensitive hash functions

(Informally:) A family $\mathcal{H}$ of hash functions from $\mathbb{R}^d$ to $\mathbb{Z}$ is a **locality-sensitive hash family** if

- For any points $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^d$ with $\|\boldsymbol{a} - \boldsymbol{b}\|_2 \ll \|\boldsymbol{a} - \boldsymbol{c}\|_2$,

$$|\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{b})\}| \gg |\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{c})\}|.$$

Procedure:

- Select a hash function $h \in \mathcal{H}$ at random.
  (In practice, some parameters of the hash function, like r and s, may be tuned via hold-out or cross-validation.)
- Create pointer from buckets $j \in \mathbb{N}$ to points $\boldsymbol{x} \in S$ such that $h(\boldsymbol{x}) = j$.
- Given test point $\boldsymbol{x}$, search bucket $h(\boldsymbol{x})$ for nearest neighbor.
  (The bucket will generally contain far fewer than n points.)

Locality sensitive hash functions

(Informally:) A family $\mathcal{H}$ of hash functions from $\mathbb{R}^d$ to $\mathbb{Z}$ is a **locality-sensitive hash family** if

- For any points $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c} \in \mathbb{R}^d$ with $\|\boldsymbol{a} - \boldsymbol{b}\|_2 \ll \|\boldsymbol{a} - \boldsymbol{c}\|_2$,

$$|\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{b})\}| \gg |\{h \in \mathcal{H} : h(\boldsymbol{a}) = h(\boldsymbol{c})\}|.$$

Procedure:

- Select a hash function $h \in \mathcal{H}$ at random.

  (In practice, some parameters of the hash function, like r and s, may be tuned via hold-out or cross-validation.)
- Create pointer from buckets $j \in \mathbb{N}$ to points $\boldsymbol{x} \in S$ such that $h(\boldsymbol{x}) = j$.
- Given test point $\boldsymbol{x}$, search bucket $h(\boldsymbol{x})$ for nearest neighbor.

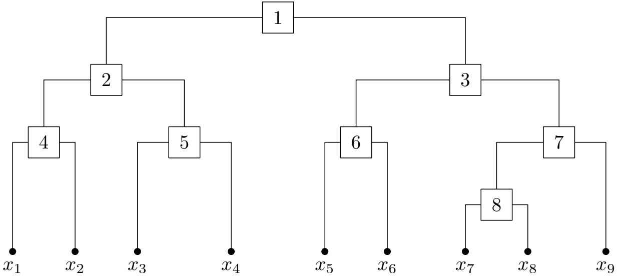
  (The bucket will generally contain far fewer than n points.)

Do this with several hash functions from $\mathcal{H}$ to boost the chances that you find close neighbors.

Tree structures for one-dimensional data

A data structure for fast NN search in $\mathbb{R}^1$

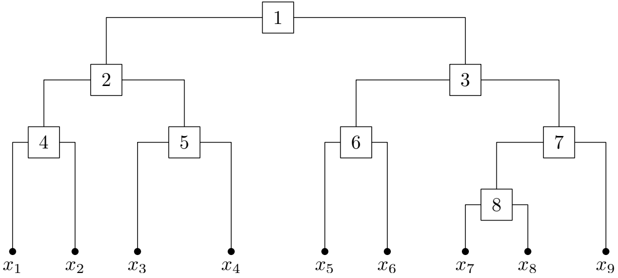
Sort training data so that $x_1 \leq x_2 \leq \cdots \leq x_n$, then construct binary tree:



Text

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:

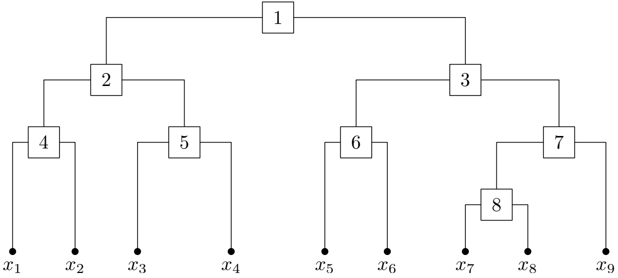


With each tree node, remember **midpoint** between rightmost point in left child, and leftmost point in right child.

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:

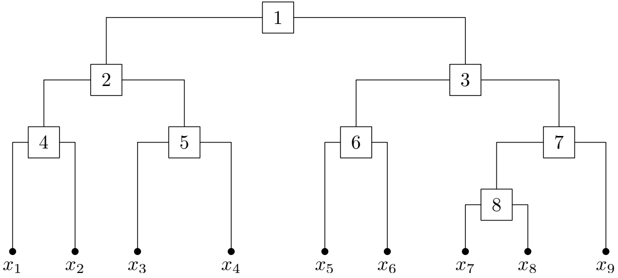


With each tree node, remember **midpoint** between rightmost point in left child, and leftmost point in right child. **This permits very efficient NN search.**

Tree structures for one-dimensional data

A data structure for fast NN search in $\mathbb{R}^1$

Sort training data so that $x_1 \le x_2 \le \cdots \le x_n$, 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**.

Tree structures for multi-dimensional data

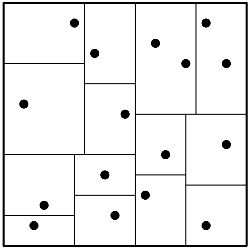
A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

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.



Text

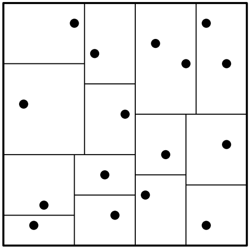
A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

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), but how about NN search?

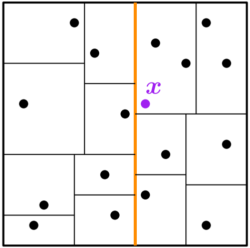
A data structure for fast NN search in $\mathbb{R}^d$, $d > 1$
Many options, but a popular one is the **K-D tree**.

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), but how about NN search?

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

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 most optimistic 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 $\boldsymbol{x}_R$.

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 most optimistic 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.

Generic NN search procedure for binary space partition trees

Given a test point $\boldsymbol{x}$ and a tree node v (initially $v = \text{root}$):

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

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

   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?

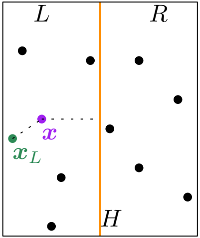
- **Note**: it's correct (though computationally wasteful) to declare "false" in Step 2 even if $(\star)$ turns out to be true.

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

Using geometric properties

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

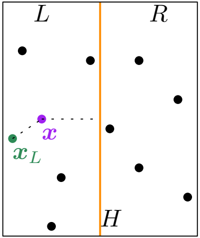
Suppose test point $\boldsymbol{x}$ is in L, and the NN of $\boldsymbol{x}$ in L is $\boldsymbol{x}_L$.



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,

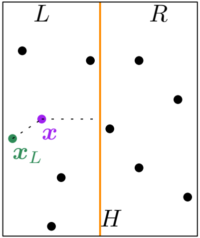
$$\min_{\boldsymbol{x}' \in R} \|\boldsymbol{x} - \boldsymbol{x}'\|_2 \geq \text{distance from } \boldsymbol{x} \text{ to } H$$

$$= |x_j - m|.$$

Using geometric properties

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

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



By geometry,

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

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

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

once there is point at left part

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.

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!

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? (Next lecture.)