1. Nearest neighbor classification.
Nearest neighbor classification
THE OPTIMAL CLASSIFIER

Let \((X, Y) \sim P\).

The classifier \(f: \mathcal{X} \rightarrow \mathcal{Y}\) with the smallest prediction error

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

is the **Bayes classifier**

\[
f^*(x) = \arg \max_{y \in \mathcal{Y}} \Pr[Y = y \mid X = x].
\]
The optimal classifier

Let \((X, Y) \sim P\).

The classifier \(f : \mathcal{X} \rightarrow \mathcal{Y}\) with the smallest prediction error

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

is the **Bayes classifier**

\[
f^*(x) = \arg\max_{y \in \mathcal{Y}} \Pr[Y = y \mid X = x].
\]

Of course, we don’t know \(P\), and hence can’t generally get a handle of \(f^*\).
**The optimal classifier**

Let \( (X, Y) \sim P \).

The classifier \( f: \mathcal{X} \rightarrow \mathcal{Y} \) with the smallest prediction error

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

is the **Bayes classifier**

\[
f^*(x) = \arg \max_{y \in \mathcal{Y}} \Pr[Y = y \mid X = x].
\]

Of course, we don’t know \( P \), and hence can’t generally get a handle of \( f^* \).

**What can we do?**

- **Last lecture**: use “generative” models to approximate \( \Pr[Y = y \mid X = x] \).
THE OPTIMAL CLASSIFIER

Let \((X, Y) \sim P\).

The classifier \(f : \mathcal{X} \rightarrow \mathcal{Y}\) with the smallest prediction error

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

is the **Bayes classifier**

\[
f^*(x) = \arg \max_{y \in \mathcal{Y}} \Pr[Y = y \mid X = x].
\]

Of course, we don’t know \(P\), and hence can’t generally get a handle of \(f^*\).

**What can we do?**

- **Last lecture**: use “generative” models to approximate \(\Pr[Y = y \mid X = x]\).
- **This lecture**: directly approximate the decision boundaries of \(f^*\).
Nearest neighbor (NN) classifier

Given training data \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}\), construct \(\hat{f} : \mathcal{X} \to \mathcal{Y}\) as follows:

On input \(x\),

1. Let \(x_i\) be the point among \(x_1, x_2, \ldots, x_n\) that is closest to \(x\).
2. Return \(y_i\).

Question: how should we measure distance between points in \(\mathcal{X}\)?
Nearest neighbor (NN) classifier

Given training data \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}\), construct \(\hat{f} : \mathcal{X} \rightarrow \mathcal{Y}\) as follows:

On input \(x\),

1. Let \(x_i\) be the point among \(x_1, x_2, \ldots, x_n\) that is closest to \(x\).
2. Return \(y_i\).

**Question**: how should we measure distance between points in \(\mathcal{X}\)?
A default choice of distance for data in $\mathbb{R}^d$:

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

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

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

But there are many other options (which could be much better than $\ell_2$) . . .

- $\ell_p$ for $p \in [1, \infty]$:

  $$\|u - v\|_p := \left(\sum_{i=1}^{d} |u_i - v_i|^p\right)^{1/p}.$$ 

- Edit distance (for strings): how add/delete/substitutions are required to transform one string to the other.

- Shape distance (for images): figures out what “shape” is depicted in each image, then computes a distance based on how much “warping” is required to change one to the other.
Example: OCR with NN classifier

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

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

0 1 2 3 4 5 6 7 8 9

- 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 $28 \times 28$ images, treated as vectors in $\mathbb{R}^{784}$, with labels indicating the digit they represent.

  0 1 2 3 4 5 6 7 8 9

- Split into training data $S$ (60000 points) and test data $T$ (10000 points).
- **Training error**: $\text{err}(\hat{f}, S) = 0$
  
  **Test error**: $\text{err}(\hat{f}, T) = 0.0309$

- Examples of mistakes (test point in $T$, nearest neighbor in $S$):

  28 35 54 41
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: $\text{err}(\hat{f}, S) = 0$
  Test error: $\text{err}(\hat{f}, T) = 0.0309$

- 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]
Given training data \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}\), construct \(\hat{f}_k : \mathcal{X} \rightarrow \mathcal{Y}\) as follows:

On input \(x\),

1. Let \(x_{i_1}, x_{i_2}, \ldots, x_{i_k}\) be the \(k\) points among \(x_1, x_2, \ldots, x_n\) that are closest to \(x\).

2. Return the plurality of \(y_{i_1}, y_{i_2}, \ldots, y_{i_k}\).

(Break ties in both steps arbitrarily.)
Given training data \((x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n) \in \mathcal{X} \times \mathcal{Y}\), construct \(\hat{f}_k : \mathcal{X} \to \mathcal{Y}\) as follows:

On input \(x\),

1. Let \(x_{i_1}, x_{i_2}, \ldots, x_{i_k}\) be the \(k\) points among \(x_1, x_2, \ldots, x_n\) that are closest to \(x\).
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

<table>
<thead>
<tr>
<th>(k)</th>
<th>1</th>
<th>3</th>
<th>5</th>
<th>7</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{err}(\hat{f}_k, T))</td>
<td>0.0309</td>
<td>0.0295</td>
<td>0.0312</td>
<td>0.0306</td>
<td>0.0341</td>
</tr>
</tbody>
</table>
**Effect of \( k \)**

**In general:**

- Smaller \( k \) ⇒ smaller training error. \( (k = 1 \Rightarrow \text{has zero training error}) \)
- Larger \( k \) ⇒ predictions are more “stable” due to voting.

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:** \( \hat{k} := \text{arg min}_k \text{err}(\hat{f}_k, T). \)

\[\text{Caveat: } \hat{f}_{\hat{k}} \text{ is no longer independent of } T \Rightarrow \text{Test error is not an unbiased estimate of true error of } \hat{f}_{\hat{k}}.\]

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

1. **Minimizer of hold-out error:** fix $H \subseteq S$, $\hat{k} := \text{arg min}_k \text{err}(\hat{f}(S \setminus H, k), H)$.
2. **Minimizer of leave-one-out cross-validation error:** $\hat{k} := \text{arg min}_k 1/|S| \sum_{(x,y) \in S} \text{err}(\hat{f}(S \setminus \{(x,y)\}, k), \{(x, y)\})$.

\[\Rightarrow \text{Test error is an unbiased estimate of true error of } \hat{f}_{\hat{k}}.\]

More on this later in the course.
**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:** \( \hat{k} := \arg \min_k \text{err}(\hat{f}_k, T) \).

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

More on this later in the course.
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:** $\hat{k} := \arg\min_k \text{err}(\hat{f}_k, T)$.

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

$\implies$ Test error is not an unbiased estimate of true error of $\hat{f}_{\hat{k}}$. More on this later in the course.
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:** $\hat{k} := \arg \min_k \text{err}(\hat{f}_k, T)$.

  **Caveat:** $\hat{f}_k$ is no longer independent of $T$!

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

- **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 $H \subseteq S$,

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


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:** $\hat{k} := \arg \min_k \text{err}(\hat{f}_k, T)$.

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

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

- **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 $H \subseteq S$,

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

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

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

More on this later in the course.
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:** $\hat{k} := \arg \min_k \text{err}(\hat{f}_k, T)$.

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

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

- **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 $H \subseteq S$,

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

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

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

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

More on this later in the course.
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:** $\hat{k} := \arg \min_k \text{err}(\hat{f}_k, T)$.

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

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

- **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 $H \subseteq S$,

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

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

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

$\implies$ 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

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

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

$$\lim_{n \to \infty} \mathbb{E}[\text{err}(\hat{f}_1)] \leq 2 \cdot \text{err}(f^*) \cdot (1 - \mathcal{K}_2(\mathcal{K} - 1) \cdot \text{err}(f^*)).$$
Say a learning algorithm is **consistent** if

\[
\lim_{n \to \infty} \mathbb{E}\left[\text{error of learned classifier with training sample size } n\right] = \text{err}(f^*).
\]

**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).
Consistency of $k$-NN

Say a learning algorithm is **consistent** if

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

$k$-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 $\text{err}(f^*) = 0$, although

$$
\lim_{n \to \infty} \mathbb{E}\left[\text{err}(\hat{f}_1)\right] \leq 2 \text{err}(f^*) \cdot \left(1 - \frac{K}{2(K - 1)} \text{err}(f^*)\right).
$$
Naïve implementation of NN classifiers uses $n$ distance computations to compute $\hat{f}_k(x)$ for any test point $x \in \mathcal{X}$.
Naïve implementation of NN classifiers uses \( n \) distance computations to compute \( \hat{f}_k(x) \) for any test point \( x \in \mathcal{X} \).

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

\[ \Rightarrow O(dn) \text{ operations per test point.} \]
Naïve implementation of NN classifiers uses $n$ distance computations to compute $\hat{f}_k(x)$ for any test point $x \in \mathcal{X}$.

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

$$\implies O(dn) \text{ operations per test point.}$$

**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 \( a, b, c \in X \) with \( \|a - b\|_2 \ll \|a - c\|_2 \),

\[
|\{h \in \mathcal{H} : h(a) = h(b)\}| \gg |\{h \in \mathcal{H} : h(a) = h(c)\}|.
\]
Locality Sensitive Hash Functions

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

- For any points \( a, b, c \in X \) with \( \|a - b\|_2 \ll \|a - c\|_2 \),

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

It turns out there are such hash families!
A LSH family based on projections to one-dimensional subspaces

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

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

- \( \mathbf{w} \) determines the one-dimensional subspace,
- \( r \) determines a distance resolution, and
- \( s \) determines a shift of the bucket boundaries.
Localilty-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 $a, b, c \in \mathbb{R}^d$ with $\|a - b\|_2 \ll \|a - c\|_2$,

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

**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 $x \in S$ such that $h(x) = j$.

- Given test point $x$, search bucket $h(x)$ 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 $a, b, c \in \mathbb{R}^d$ with $\|a - b\|_2 \ll \|a - c\|_2$,

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

**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 $x \in S$ such that $h(x) = j$.

- Given test point $x$, search bucket $h(x)$ 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.
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
 / \   / \ \\
 x1 x2 x3 x4 x5 x6 x7 x8 x9
```
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.
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.**
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!
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 $\{x_j : x \in S\}$.
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 $R$. 
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), 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 $\{x_j : x \in S\}$.
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), but how about NN search?

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 \( x_R \).

\[\text{Note: can't always guarantee } O(\log(n)) \text{ search time due to Step 3.}\]

\[\text{Question: How do you check if } (\star) \text{ is true?} \]

\[\text{\(\rightarrow\text{ Note: it's correct (though computationally wasteful) to declare "false" in Step 2 even if } (\star) \text{ turns out to be true.}\)}\]
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
   \[ \|x - x_L\|_2 < \min_{x' \in R} \|x - x'\|_2 \]  \hspace{1cm} (\star)
   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.
**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
   \[
   \|x - x_L\|_2 < \min_{x' \in R} \|x - x'\|_2 \quad \text{(**)}
   \]
   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.
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, 
\[
\min_{x' \in \mathbb{R}} \| x - x' \|_2 \geq \text{distance from } x \text{ to } H = |x_j - m|.
\]

A valid check: if $\| x - x_L \|_2 < |x_j - m|$, then $\| x - x_L \|_2 < \min_{x' \in \mathbb{R}} \| x - x' \|_2$. In this case, we can skip searching $R$ and immediately return $x_L$. 
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 $x_L$. 

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 $x_L$. 
**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,

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

$$= |x_j - m|.$$
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.
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!**
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.)