$k$-nn and decision trees

CS 446
Today we’ll cover two standard machine learning methods.

Nearest neighbors.

Decision trees.
1. Nearest neighbor rule
Example: OCR for digits

1. Classify images of handwritten digits by the actual digits they represent.

2. Classification problem: $\mathcal{Y} = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ (a discrete set).

Digits from standard MNIST dataset (Lecun, Cortes, Burges).
Nearest neighbor (NN) classifier

Given: labeled examples $D := \{(x_i, y_i)\}_{i=1}^n$

Predictor: $\hat{f}_D : \mathcal{X} \rightarrow \mathcal{Y}$

On input $x$,

1. Find the point $x_i$ among $\{x_i\}_{i=1}^n$ that is “closest” to $x$ (the nearest neighbor).
2. Return $y_i$.

Remark. For nearest neighbor, there is no “fitting” procedure, other than memorizing the training data!
How to measure distance?

A default choice for distance between points in $\mathbb{R}^d$ is the Euclidean distance (also called $\ell_2$ distance):

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

(where $u = (u_1, u_2, \ldots, u_d)$ and $v = (v_1, v_2, \ldots, v_d)$).

Grayscale $28 \times 28$ pixel images.

Treat as vectors (of 784 real-valued features) that live in $[0, 1]^{784}$.

**Note.** Spatial information lost!
2. Evaluation
Example: OCR for digits with NN classifier

- Classify images of handwritten digits by the digits they depict.
Example: OCR for digits with NN classifier

- Classify images of handwritten digits by the digits they depict.

\[ \mathcal{X} = \mathbb{R}^{784}, \mathcal{Y} = \{0, 1, \ldots, 9\}. \]
Example: OCR for digits with NN classifier

- Classify images of handwritten digits by the digits they depict.

\[
0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9
\]

- \( \mathcal{X} = \mathbb{R}^{784} \), \( \mathcal{Y} = \{0, 1, \ldots, 9\} \).

- **Given**: labeled examples \( D := \{(x_i, y_i)\}_{i=1}^{n} \subset \mathcal{X} \times \mathcal{Y} \).
Example: OCR for digits with NN classifier

▶ Classify images of handwritten digits by the digits they depict.

0 1 2 3 4 5 6 7 8 9

▶ $\mathcal{X} = \mathbb{R}^{784}$, $\mathcal{Y} = \{0, 1, \ldots, 9\}$.

▶ **Given**: labeled examples $D := \{(x_i, y_i)\}_{i=1}^n \subset \mathcal{X} \times \mathcal{Y}$.

▶ Construct NN classifier $\hat{f}_D$ using $D$. 
Example: OCR for digits with NN classifier

- Classify images of handwritten digits by the digits they depict.

- \( \mathcal{X} = \mathbb{R}^{784}, \mathcal{Y} = \{0, 1, \ldots, 9\} \).

- **Given:** labeled examples \( D := \{(x_i, y_i)\}_{i=1}^{n} \subset \mathcal{X} \times \mathcal{Y} \).

- Construct NN classifier \( \hat{f}_D \) using \( D \).

- **Question:** Is this classifier any good?
Error rate

Error rate of classifier $f$ on a set of labeled examples $D$:

$$\text{err}(f; D) := \frac{\# \text{ of } (x, y) \in D \text{ such that } f(x) \neq y}{|D|}$$

(i.e., the fraction of $D$ on which $f$ disagrees with paired label).
Error rate

- **Error rate** of classifier $f$ on a set of labeled examples $D$:

$$\text{err}(f; D) := \frac{\# \text{ of } (x, y) \in D \text{ such that } f(x) \neq y}{|D|}$$

(i.e., the fraction of $D$ on which $f$ disagrees with paired label).

- **Question**: What is $\text{err}(\hat{f}_D; D)$?
Error rate

- **Error rate** of classifier $f$ on a set of labeled examples $D$:

$$\text{err}(f; D) := \frac{\# \text{ of } (x, y) \in D \text{ such that } f(x) \neq y}{|D|}$$

(i.e., the fraction of $D$ on which $f$ disagrees with paired label).

**Question**: What is $\text{err}(\hat{f}_D; D)$?

**Note**: We can also write

$$\text{err}(f; D) = \frac{1}{|D|} \sum_{(x, y) \in D} 1[f(x) \neq y] = \frac{1}{|D|} \sum_{(x, y) \in D} \ell_{zo}(f(x), y),$$

where $\ell_{zo}(\hat{y}, y) = 1[\hat{y} \neq y]$ is the zero-one (classification) loss,
A better way to evaluate the classifier

- Split the labeled examples \( \{(x_i, y_i)\}_{i=1}^{n} \) into two sets (randomly).
  - *Training data* \( S \).
  - *Test data* \( T \).

\[ \text{Training error rate of } \hat{f}_S: \text{err}(\hat{f}_S; S) = 0\% \]

\[ \text{Test error rate of } \hat{f}_S: \text{err}(\hat{f}_S; T) = 3.09\% \]

Is this good?

This gap \( \text{err}(\hat{f}_S; T) - \text{err}(\hat{f}_S; S) = 3.09\% \) means we have overfit.
A better way to evaluate the classifier

- Split the labeled examples \( \{(x_i, y_i)\}_{i=1}^{n} \) into two sets (randomly).
  - \textit{Training data} \( S \).
  - \textit{Test data} \( T \).

- Only use \textit{training data} \( S \) to construct NN classifier \( \hat{f}_S \).

\[ \text{Training error rate of } \hat{f}_S: \text{err}(\hat{f}_S; S) = 0\% \]
\[ \text{Test error rate of } \hat{f}_S: \text{err}(\hat{f}_S; T) = 3.09\% \]

Is this good? This gap \( \text{err}(\hat{f}_S; T) - \text{err}(\hat{f}_S; S) = 3.09\% \) means we have overfit.
A better way to evaluate the classifier

▶ Split the labeled examples \( \{(x_i, y_i)\}_{i=1}^{n} \) into two sets (randomly).

▶ Training data \( S \).
▶ Test data \( T \).

▶ Only use training data \( S \) to construct NN classifier \( \hat{f}_S \).
▶ Training error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; S) = 0\% \).
A better way to evaluate the classifier

- Split the labeled examples \( \{(x_i, y_i)\}_{i=1}^{n} \) into two sets (randomly).
  - Training data \( S \).
  - Test data \( T \).

- Only use training data \( S \) to construct NN classifier \( \hat{f}_S \).
  - Training error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; S) = 0\% \).

- Use test data \( T \) to evaluate accuracy of \( \hat{f}_S \).
A better way to evaluate the classifier

- Split the labeled examples \( \{(x_i, y_i)\}_{i=1}^n \) into two sets (randomly).
  - Training data \( S \).
  - Test data \( T \).

- Only use training data \( S \) to construct NN classifier \( \hat{f}_S \).
  - Training error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; S) = 0\% \).

- Use test data \( T \) to evaluate accuracy of \( \hat{f}_S \).
  - Test error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; T) = 3.09\% \).
A better way to evaluate the classifier

- Split the labeled examples \( \{ (x_i, y_i) \}_{i=1}^{n} \) into two sets (randomly).
  - Training data \( S \).
  - Test data \( T \).

- Only use training data \( S \) to construct NN classifier \( \hat{f}_S \).
  - Training error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; S) = 0\% \).

- Use test data \( T \) to evaluate accuracy of \( \hat{f}_S \).
  - Test error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; T) = 3.09\% \).

Is this good?
A better way to evaluate the classifier

- Split the labeled examples \( \{(x_i, y_i)\}_{i=1}^{n} \) into two sets (randomly).
  - Training data \( S \).
  - Test data \( T \).

- Only use training data \( S \) to construct NN classifier \( \hat{f}_S \).
  - Training error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; S) = 0\% \).

- Use test data \( T \) to evaluate accuracy of \( \hat{f}_S \).
  - Test error rate of \( \hat{f}_S \): \( \text{err}(\hat{f}_S; T) = 3.09\% \).

Is this good?

This gap \( \text{err}(\hat{f}_S; T) - \text{err}(\hat{f}_S; S) = 3.09\% \) means we have overfit.
Why does NN work on new data?

Consider any new point \( x \).
As training set size increases,
\( k \)th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point \( x \).
As training set size increases, 
\( k \)th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point $x$. As training set size increases, $k$:th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point $x$.
As training set size increases, $k$:th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point \( x \).
As training set size increases,
\( k \)th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point $x$.
As training set size increases, $k$th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point $x$. As training set size increases, $k$th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point $x$.
As training set size increases, $k$:th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point \( x \).
As training set size increases,
\( k \)th nearest neighbor becomes closer and closer.
Why does NN work on new data?

Consider any new point \( x \).
As training set size increases, 
\( k \)th nearest neighbor becomes closer and closer.

**Note.** Can prove this so long as \( k/n \to 0! \)
3. Upgrading the nearest neighbor rule
Diagnostics

- Some mistakes made by the NN classifier (test point in $T$, nearest neighbor in $S$):

  28

  3 5

  54
Some mistakes made by the NN classifier (test point in $T$, nearest neighbor in $S$):

First mistake (correct label is “2”) could’ve been avoided by looking at the three nearest neighbors (whose labels are “8”, “2”, and “2”).

-test point three nearest neighbors
Given: labeled examples $D := \{(x_i, y_i)\}_{i=1}^n$

Predictor: $\hat{f}_{D,k} : \mathcal{X} \rightarrow \mathcal{Y}$:

On input $x$,

1. Find the $k$ points $x_{i_1}, x_{i_2}, \ldots, x_{i_k}$ among $\{x_i\}_{i=1}^n$ “closest” to $x$ (the $k$ nearest neighbors).
2. Return the plurality of $y_{i_1}, y_{i_2}, \ldots, y_{i_k}$.

(Break ties in both steps arbitrarily.)
Effect of $k$

- Smaller $k$: smaller training error rate.
- Larger $k$: higher training error rate, but predictions are more "stable" due to voting.

<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>Test error rate</td>
<td>0.0309</td>
<td>0.0295</td>
<td>0.0312</td>
<td>0.0306</td>
<td>0.0341</td>
</tr>
</tbody>
</table>
Choosing $k$

The hold-out set approach

1. Pick a subset $V \subset S$ (hold-out set, a.k.a. validation set).
2. For each $k \in \{1, 3, 5, \ldots \}$:
   - Construct $k$-NN classifier $\hat{f}_{S \setminus V, k}$ using $S \setminus V$.
   - Compute error rate of $\hat{f}_{S \setminus V, k}$ on $V$ ("hold-out error rate").
3. Pick the $k$ that gives the smallest hold-out error rate.
Choosing $k$

**The hold-out set approach**

1. Pick a subset $V \subset S$ (*hold-out set, a.k.a. validation set*).
2. For each $k \in \{1, 3, 5, \ldots \}$:
   - Construct $k$-NN classifier $\hat{f}_{S \setminus V, k}$ using $S \setminus V$.
   - Compute error rate of $\hat{f}_{S \setminus V, k}$ on $V$ ("hold-out error rate").
3. Pick the $k$ that gives the smallest hold-out error rate.

(There are many other approaches.)
4. Other issues with nearest neighbor prediction
Better distance functions

- **Strings**: edit distance

  \[ \text{dist}(u, v) = \# \text{ insertions/deletions/mutations needed to change } u \text{ to } v. \]
Better distance functions

- **Strings**: edit distance
  \[
  \text{dist}(u, v) = \text{\# insertions/deletions/mutations needed to change } u \text{ to } v.
  \]

- **Images**: shape context distance
  \[
  \text{dist}(u, v) = \text{how much “warping” is required to change } u \text{ to } v.
  \]
Better distance functions

- **Strings**: edit distance
  \[ \text{dist}(u, v) = \# \text{ insertions/deletions/mutations needed to change } u \text{ to } v. \]

- **Images**: shape context distance
  \[ \text{dist}(u, v) = \text{how much “warping” is required to change } u \text{ to } v. \]

- **Audio waveforms**: dynamic time warping
- **Etc.**
Better distance functions

- **Strings**: edit distance
  \[ \text{dist}(u, v) = \# \text{ insertions/deletions/mutations needed to change } u \text{ to } v. \]

- **Images**: shape context distance
  \[ \text{dist}(u, v) = \text{how much “warping” is required to change } u \text{ to } v. \]

- **Audio waveforms**: dynamic time warping

- Etc.

---

**OCR digits classification**

<table>
<thead>
<tr>
<th>Distance</th>
<th>$\ell_2$</th>
<th>$\ell_3$</th>
<th>Tangent</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test error rate</td>
<td>3.09%</td>
<td>2.83%</td>
<td>1.10%</td>
<td>0.63%</td>
</tr>
</tbody>
</table>
**Caution**: nearest neighbor classifier can be broken by bad/noisy features!

![Diagram of Feature 1 and Feature 2 with y = 0 and y = 1](image)

**Curse of dimension.** Given poly($d$) random unit norm points in $\mathbb{R}^d$, with probability $> 99\%$, each is squared distance $2 \pm O\left(\frac{1}{\sqrt{d}}\right)$ from all others.

**Looking ahead:** if we train a deep network on other data, we can chop it in the middle, and those ($k \ll d$) outputs become NN features!
Naïve method for computing NN predictions: 
\[ O(n) \] distance computations.
Naïve method for computing NN predictions:
\[ O(n) \] distance computations.

Better: organize training data in a data structure to improve look-up time.
Naïve method for computing NN predictions:

\[ O(n) \] distance computations.

Better: organize training data in a data structure to improve look-up time.

- Space: \[ O(nd) \] for \( n \) points in \( \mathbb{R}^d \).
- Query time: \[ O(2^d \log n) \] time in worst-case.
Naïve method for computing NN predictions: 
\(O(n)\) distance computations.

Better: organize training data in a data structure to improve look-up time.

- Space: \(O(nd)\) for \(n\) points in \(\mathbb{R}^d\).
- Query time: \(O(2^d \log n)\) time in worst-case.

Finding an “approximate” NN can be more efficient.
Naïve method for computing NN predictions:
\[ O(n) \] distance computations.

Better: organize training data in a data structure to improve look-up time.
- Space: \( O(nd) \) for \( n \) points in \( \mathbb{R}^d \).
- Query time: \( O(2^d \log n) \) time in worst-case.

Finding an “approximate” NN can be more efficient.

E.g., how to quickly find a point among the top-1% closest points?
Naïve method for computing NN predictions:
\( O(n) \) distance computations.

Better: organize training data in a data structure to improve look-up time.

- Space: \( O(nd) \) for \( n \) points in \( \mathbb{R}^d \).
- Query time: \( O(2^d \log n) \) time in worst-case.

Finding an “approximate” NN can be more efficient.

E.g., how to quickly find a point among the top-1\% closest points?

- Popular technique: Locality sensitive hashing
5. Decision trees
**Decision trees**

Directly optimize tree structure for good classification.

A decision tree is a function $f : \mathcal{X} \rightarrow \mathcal{Y}$, represented by a binary tree in which:

- Each **tree node** is associated with a splitting rule $g : \mathcal{X} \rightarrow \{0, 1\}$.
- Each **leaf node** is associated with a label $\hat{y} \in \mathcal{Y}$.
Decision trees

Directly optimize tree structure for good classification.

A decision tree is a function \( f: \mathcal{X} \rightarrow \mathcal{Y} \), represented by a binary tree in which:

- Each tree node is associated with a splitting rule \( g: \mathcal{X} \rightarrow \{0, 1\} \).
- Each leaf node is associated with a label \( \hat{y} \in \mathcal{Y} \).

When \( \mathcal{X} = \mathbb{R}^d \), typically only consider splitting rules of the form

\[
g(x) = \mathbf{1}\{x_i > t\}
\]

for some \( i \in [d] \) and \( t \in \mathbb{R} \).

Called axis-aligned or coordinate splits.

(Notation: \( [d] := \{1, 2, \ldots, d\} \).)
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width
Decision tree example

Classifying irises by sepal and petal measurements

\[ \mathcal{X} = \mathbb{R}^2, \mathcal{Y} = \{1, 2, 3\} \]

\[ x_1 = \text{ratio of sepal length to width} \]

\[ x_2 = \text{ratio of petal length to width} \]
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

Decision tree example

$X_1 > 1.7$
Decision tree example

Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 =$ ratio of sepal length to width
- $x_2 =$ ratio of petal length to width

$y = 1$ if $x_1 > 1.7$

$y = 3$ if $x_1 \leq 1.7$
Classifying irises by sepal and petal measurements

- \( X = \mathbb{R}^2 \), \( Y = \{1, 2, 3\} \)
- \( x_1 = \text{ratio of sepal length to width} \)
- \( x_2 = \text{ratio of petal length to width} \)

Decision tree example
Classifying irises by sepal and petal measurements

- $\mathcal{X} = \mathbb{R}^2$, $\mathcal{Y} = \{1, 2, 3\}$
- $x_1 = \text{ratio of sepal length to width}$
- $x_2 = \text{ratio of petal length to width}$

Decision tree example

- $x_1 > 1.7$
- $x_2 > 2.8$
- $\hat{y} = 1$
- $\hat{y} = 2$
- $\hat{y} = 3$
Basic decision tree learning algorithm

Basic “top-down” greedy algorithm

- Initially, tree is a single leaf node containing all (training) data.
- Loop:
  - Pick the leaf $\ell$ and rule $h$ that maximally reduces uncertainty.
  - Split data in $\ell$ using $h$, and grow tree accordingly.

\[ \text{... until some stopping criterion is satisfied.} \]

[Label of a leaf is the plurality label among the data contained in the leaf.]
Notions of uncertainty: binary case ($\mathcal{Y} = \{0, 1\}$)

Suppose in a set of examples $S \subseteq \mathcal{X} \times \{0, 1\}$, a $p$ fraction are labeled as 1.

1. **Classification error:**
   $$u(S) := \min\{p, 1 - p\}$$

2. **Gini index:**
   $$u(S) := 2p(1 - p)$$

3. **Entropy:**
   $$u(S) := p \log \frac{1}{p} + (1 - p) \log \frac{1}{1 - p}$$

Gini index and entropy (after some rescaling) are concave upper-bounds on classification error.
Notions of uncertainty: general case ($\mathcal{Y} = \{1, 2, \ldots, K\}$)

Suppose in $S \subseteq \mathcal{X} \times \mathcal{Y}$, a $p_y$ fraction are labeled as $y$ (for each $y \in \mathcal{Y}$).

1. **Classification error**:

   $$u(S) := 1 - \max_{y \in \mathcal{Y}} p_y$$

2. **Gini index**:

   $$u(S) := 1 - \sum_{y \in \mathcal{Y}} p_y^2$$

3. **Entropy**:

   $$u(S) := \sum_{y \in \mathcal{Y}} p_y \log \frac{1}{p_y}$$
Notions of uncertainty: general case ($\mathcal{Y} = \{1, 2, \ldots, K\}$)

Suppose in $S \subseteq \mathcal{X} \times \mathcal{Y}$, a $p_y$ fraction are labeled as $y$ (for each $y \in \mathcal{Y}$).

1. **Classification error:**
   
   $$u(S) := 1 - \max_{y \in \mathcal{Y}} p_y$$

2. **Gini index:**
   
   $$u(S) := 1 - \sum_{y \in \mathcal{Y}} p_y^2$$

3. **Entropy:**
   
   $$u(S) := \sum_{y \in \mathcal{Y}} p_y \log \frac{1}{p_y}$$

   Each is *maximized* when $p_y = 1/K$ for all $y \in \mathcal{Y}$ (i.e., equal numbers of each label in $S$).

   Each is *minimized* when $p_y = 1$ for a single label $y \in \mathcal{Y}$ (so $S$ is *pure* in label).
Limitations of uncertainty notions

Suppose $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{\text{red}, \text{blue}\}$, and the data is as follows:

Every split of the form $1\{x_i > t\}$ provides no reduction in uncertainty (whether based on classification error, Gini index, or entropy).

![Data representation](image-url)
Limitations of uncertainty notions

Suppose \( \mathcal{X} = \mathbb{R}^2 \) and \( \mathcal{Y} = \{ \text{red}, \text{blue} \} \), and the data is as follows:

Every split of the form \( \mathbb{1}\{x_i > t\} \) provides no reduction in uncertainty (whether based on classification error, Gini index, or entropy).

**Upshot:**
Zero reduction in uncertainty may not be a good stopping condition.
Basic decision tree learning algorithm

Basic “top-down” greedy algorithm

- Initially, tree is a single leaf node containing all (training) data.
- Loop:
  - Pick the leaf \( \ell \) and rule \( h \) that maximally reduces uncertainty.
  - Split data in \( \ell \) using \( h \), and grow tree accordingly.

\[ \text{... until some stopping criterion is satisfied.} \]

[Label of a leaf is the plurality label among the data contained in the leaf.]
Stopping criterion

Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
   Involves setting additional “tuning parameters” (similar to $k$ in $k$-NN).

2. Stop when every leaf is pure. (More common.)
   Serious danger of overfitting spurious structure due to sampling.
Stopping criterion

Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
Stopping criterion

Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.

   Involves setting additional “tuning parameters” (similar to $k$ in $k$-NN).

2. Stop when every leaf is pure.

   (More common.)

   Serious danger of overfitting spurious structure due to sampling.
Stopping criterion

Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
   Involves setting additional “tuning parameters” (similar to $k$ in $k$-NN).

2. Stop when every leaf is pure. (More common.)
Stopping criterion

Many alternatives; two common choices are:

1. Stop when the **tree reaches a pre-specified size**.
   Involves setting additional “tuning parameters” (similar to $k$ in $k$-NN).

2. Stop when **every leaf is pure**. (More common.)
   Serious danger of **overfitting** spurious structure due to sampling.
Stopping criterion

Many alternatives; two common choices are:

1. Stop when the tree reaches a pre-specified size.
   
   Involves setting additional “tuning parameters” (similar to \( k \) in \( k \)-NN).

2. Stop when every leaf is pure. (More common.)
   
   Serious danger of overfitting spurious structure due to sampling.
Stopping criterion

Many alternatives; two common choices are:

1. **Stop when the tree reaches a pre-specified size.**
   Involves setting additional “tuning parameters” (similar to $k$ in $k$-NN).

2. **Stop when every leaf is pure.** (More common.)
   Serious danger of **overfitting** spurious structure due to sampling.
Many alternatives; two common choices are:

1. **Stop when the tree reaches a pre-specified size.**
   
   Involves setting additional “tuning parameters” (similar to $k$ in $k$-NN).

2. **Stop when every leaf is pure.** (More common.)
   
   Serious danger of **overfitting** spurious structure due to sampling.
Training error goes to zero as the number of nodes in the tree increases.

True error decreases initially, but eventually increases due to overfitting.
What can be done about overfitting?

Preventing overfitting

Split training data $S$ into two parts, $S'$ and $S''$:

- Use first part $S'$ to **grow the tree until all leaves are pure**.
- Use second part $S''$ to **choose a good pruning of the tree**.
What can be done about overfitting?

Preventing overfitting

Split training data $S$ into two parts, $S'$ and $S''$:

- Use first part $S'$ to **grow the tree until all leaves are pure**.
- Use second part $S''$ to **choose a good pruning of the tree**.

**Pruning algorithm**

Loop:

- Replace any tree node by a leaf node if it improves the error on $S''$.

... until no more such improvements possible.
What can be done about overfitting?

**Preventing overfitting**

Split training data $S$ into two parts, $S'$ and $S''$:

- Use first part $S'$ to **grow the tree until all leaves are pure**.
- Use second part $S''$ to **choose a good pruning of the tree**.

**Pruning algorithm**

Loop:

- Replace any tree node by a leaf node if it improves the error on $S''$.

...until no more such improvements possible.

This can be done efficiently using **dynamic programming** (bottom-up traversal of the tree).
What can be done about overfitting?

Preventing overfitting
Split training data $S$ into two parts, $S'$ and $S''$:

- Use first part $S'$ to **grow the tree until all leaves are pure.**
- Use second part $S''$ to **choose a good pruning of the tree.**

Pruning algorithm
Loop:

- Replace any tree node by a leaf node if it improves the error on $S''$. …until no more such improvements possible.

This can be done efficiently using **dynamic programming** (bottom-up traversal of the tree).

**Independence of** $S'$ and $S''$ **make it unlikely for spurious structures in each to perfectly align.**
With both $k$-nn and DT, overfitting (large test-vs-train gap) when:

► $k$ in $k$-nn is too large, or
With both \( k \)-nn and DT, overfitting (large test-vs-train gap) when:

- \( k \) in \( k \)-nn is too large, or
- DT’s number of leaves (or a surrogate like depth) are too large.
Overfitting and (model complexity)

With both \( k \)-nn and DT, overfitting (large test-vs-train gap) when:

- \( k \) in \( k \)-nn is too large, or
- DT’s number of leaves (or a surrogate like depth) are too large.

More generally, high model complexity leads to overfitting.
With both $k$-nn and DT, overfitting (large test-vs-train gap) when:

- $k$ in $k$-nn is too large, or
- DT’s number of leaves (or a surrogate like depth) are too large.

More generally, high model complexity leads to overfitting.

Looking forward. In neural networks, the complexity notion is less obvious.
Example: Spam filtering

Data

- 4601 e-mail messages, 39.4% are spam.
- $\mathcal{Y} = \{\text{spam, not spam}\}$
- E-mails represented by 57 features:
  - 48: percentage of e-mail words that is specific word (e.g., “free”, “business”)
  - 6: percentage of e-mail characters that is specific character (e.g., “!”).
  - 3: other features (e.g., average length of ALL-CAPS words).

Results

Using variant of greedy algorithm to grow tree; prune tree using validation set.

Chosen tree has just **17 leaves**. Test error is 9.3%.

<table>
<thead>
<tr>
<th></th>
<th>$\hat{y} = \text{not spam}$</th>
<th>$\hat{y} = \text{spam}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$y = \text{not spam}$</td>
<td>57.3%</td>
<td>4.0%</td>
</tr>
<tr>
<td>$y = \text{spam}$</td>
<td>5.3%</td>
<td>33.4%</td>
</tr>
</tbody>
</table>
Example: Spam filtering

The pruned tree for the spam example. The split variables are shown in blue on the branches, and the classification is shown in every node. The numbers under the terminal nodes indicate misclassification rates on the test data.
6. Final remarks
Nearest neighbors and decision trees

Today we covered two standard machine learning methods.

![Diagram of nearest neighbors and decision trees]

**Nearest neighbors.**  
*Training/fitting*: memorize data.  
*Testing/predicting*: find $k$ closest memorized points, return plurality label.  
*Overfitting*? Vary $k$.

**Decision trees.**  
*Training/fitting*: greedily partition space, reducing “uncertainty”.  
*Testing/predicting*: traverse tree, output leaf label.  
*Overfitting*? Limit or prune tree.

**Note**: both methods can output real numbers (*regression*, not *classification*); return median/mean of \{ neighbors, points reaching leaf \}. 
Key takeaways

1. Two methods: $k$-nn and decision trees.
   - For $k$-nn, we have specified the complete method for fixed $k$.
   - To choose $k$, we’ve given the heuristic of using a validation set.
   - For decision tree, we have not fully specified the training procedure, only how to make decisions.

2. Training error, test error, overfitting.

3. Features.