# Academy of PhD Training in Statistics

## Statistical Machine Learning

Louis J.M. Aslett (louis.aslett@durham.ac.uk)

Trees, Forests & Boosting

# This Section

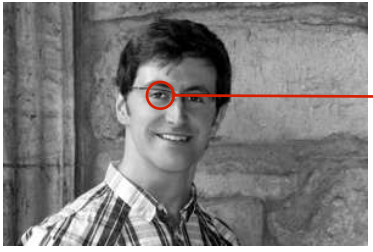
- Decision trees
  - Binary trees
  - CART
  - Pruning
- Bagging
  - Out Of Bag
- Random Forests
- Boosting
  - AdaBoost
  - Gradient boosting

# Tabular data

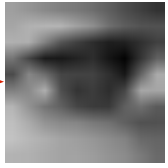
$$\mathbf{X} = \begin{pmatrix} 0.391 & 0.153 & \\ 0.628 & 0.394 & \\ \vdots & \vdots & \cdots \\ -1.556 & 2.421 & \end{pmatrix} \quad \tilde{\mathbf{x}}_{\cdot j} = \begin{pmatrix} \text{green} \\ \text{yellow} \\ \vdots \\ \text{blue} \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} \text{other features} & \begin{matrix} 0 & 1 \\ 0 & 0 \\ \vdots & \vdots \\ 1 & 0 \end{matrix} & \text{other features} \end{pmatrix}$$

$$\tilde{\mathbf{x}}_{\cdot j} = \begin{pmatrix} \text{medium} \\ \text{large} \\ \vdots \\ \text{small} \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} \text{other features} & \begin{matrix} 2 \\ 3 \\ \vdots \\ 1 \end{matrix} & \text{other features} \end{pmatrix}$$

# Unstructured data: eg Image



Me!
(300 x 200 pixels)



My eye!
(10 x 10 pixels)

$$= \begin{pmatrix} 0.70 & 0.76 & 0.75 & 0.71 & 0.50 & 0.29 & 0.46 & 0.69 & 0.65 & 0.62 \\ 0.69 & 0.72 & 0.67 & 0.58 & 0.54 & 0.53 & 0.27 & 0.69 & 0.70 & 0.58 \\ 0.67 & 0.69 & 0.58 & 0.46 & 0.69 & 0.43 & 0.11 & 0.51 & 0.71 & 0.54 \\ 0.64 & 0.65 & 0.54 & 0.38 & 0.31 & 0.27 & 0.10 & 0.34 & 0.71 & 0.50 \\ 0.64 & 0.62 & 0.53 & 0.28 & 0.20 & 0.27 & 0.14 & 0.25 & 0.64 & 0.47 \\ 0.62 & 0.59 & 0.48 & 0.24 & 0.24 & 0.26 & 0.18 & 0.18 & 0.56 & 0.43 \\ 0.58 & 0.55 & 0.45 & 0.23 & 0.18 & 0.19 & 0.09 & 0.12 & 0.44 & 0.37 \\ 0.55 & 0.51 & 0.38 & 0.38 & 0.28 & 0.09 & 0.02 & 0.13 & 0.37 & 0.31 \\ 0.48 & 0.42 & 0.32 & 0.44 & 0.66 & 0.40 & 0.21 & 0.19 & 0.33 & 0.25 \\ 0.42 & 0.40 & 0.32 & 0.36 & 0.65 & 0.67 & 0.38 & 0.25 & 0.32 & 0.20 \end{pmatrix}$$

$$\begin{pmatrix} 0.70 & 0.76 & 0.75 & 0.71 & \dots & 0.25 & 0.32 & 0.20 \end{pmatrix}$$

Each image flattened to a row vector (300x200=) 60,000 long.
ie each pixel is a feature … so a **lot** of columns in feature matrix representation!

# Unstructured data: eg Text

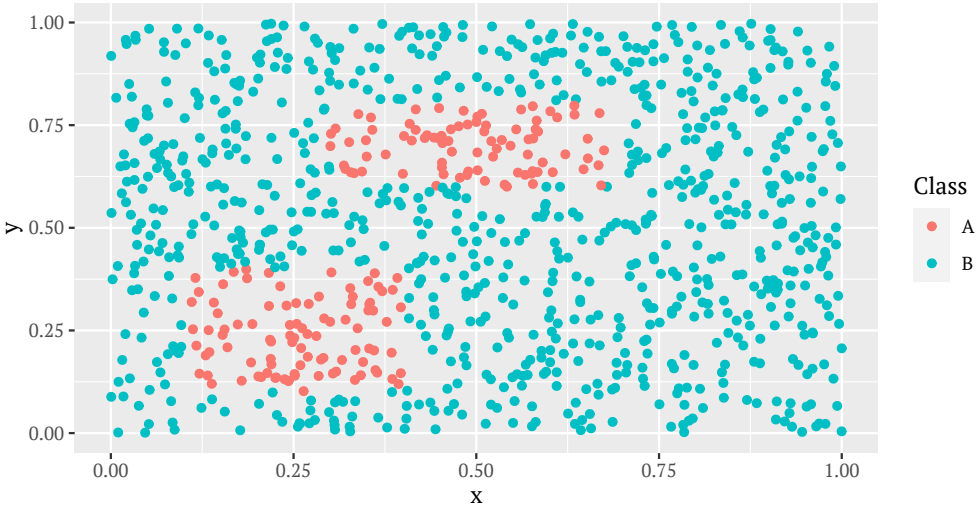
Each element of the input $\tilde{\mathbf{x}}_{\cdot j}$ is usually called a *document*, not an observation.

**Document term matrix**: is the most common encoding, assigning each word to a column containing counts, usually after removing common words ('stop words') such as 'the', 'and', 'a', etc.

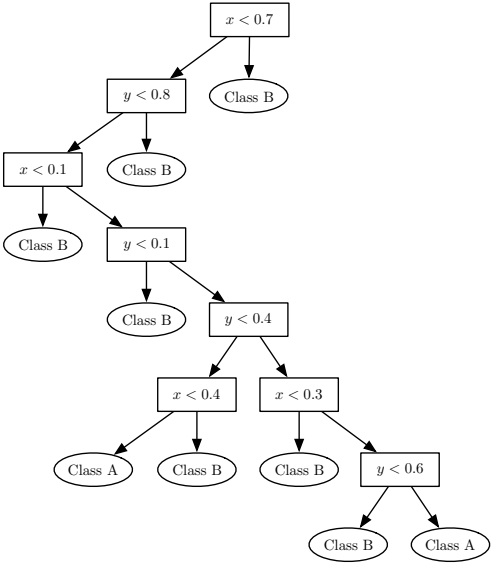
$$\tilde{\mathbf{x}}_{\cdot j} = \begin{pmatrix} \text{“It is a truth universally ...”} \\ \text{“Happy families are all alike; ...”} \\ \vdots \\ \text{“It was a bright cold day in ...”} \end{pmatrix} \quad \mathbf{X} = \begin{pmatrix} & 457 & 160 & \dots & 7 & \\ & 102 & 0 & \dots & 4 & \\ \text{other features} & \vdots & \vdots & \ddots & \vdots & \text{other features} \\ & 5285 & 54 & \dots & 0 & \end{pmatrix}$$

Also possibly $n$-grams. See `https://www.tidytextmining.com/`

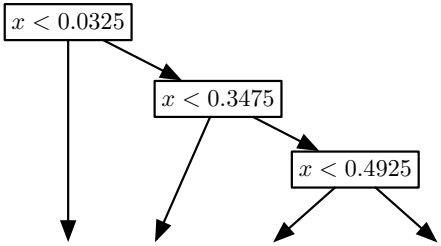
# Trees — a pictorial introduction (I)

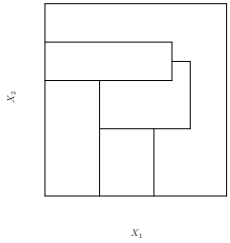


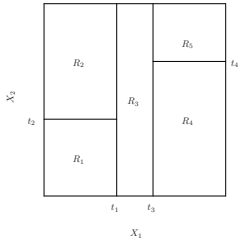
# Trees — a pictorial introduction (II)

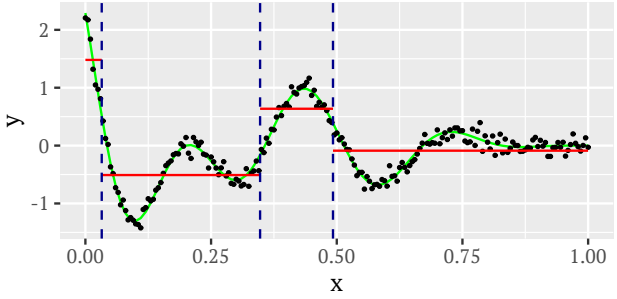


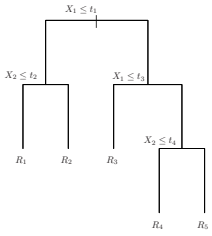
# Trees — a pictorial introduction (III)

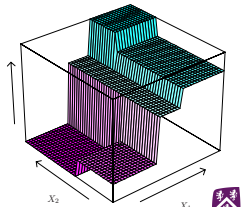


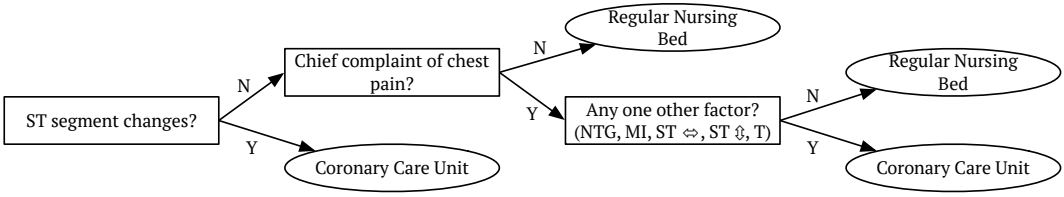



# Trees — a pictorial introduction (IV)

Classification trees mimic some decision making processes. For example, the following decision tree is used by A&E doctors to decide what bed type to assign:




This perhaps makes them popular for the feeling of interpretability and of being like a data-learned expert system.

# Trees — a pictorial introduction (V)

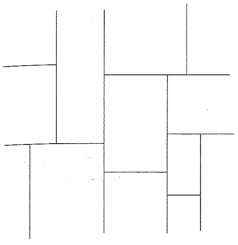


FIGURE 20.4. *Partition induced by an ordinary binary tree.*

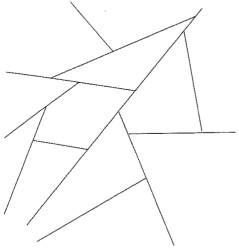


FIGURE 20.6. *Partition induced by a* BSP *tree.*

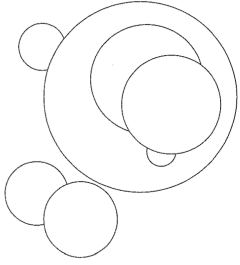


FIGURE 20.7. *Partition induced by a sphere tree.*

Figure 1: From *A Probabilistic Theory of Pattern Recognition* (1996).

# Trees

A tree is a data structure representing a partition of feature space achieved by recursive splitting. Only binary trees (ie splits into two parts) usually considered.

# Trees

A tree is a data structure representing a partition of feature space achieved by recursive splitting. Only binary trees (ie splits into two parts) usually considered.

Nomenclature:

- **Root node:** top of the tree, represents all of $\mathcal{X}$
- **Node:** some subset of $\mathcal{X}$ resulting from earlier splits
- **Left/right child:** each half of a split, creating a partition of a node
- **Leaf node:** a node which has no further splits applied
- **Depth:** length of path from the root node to reach this node
- **Height:** maximum depth among all nodes in the tree

# Tree challenges

1. if $\mathbf{x}_i \neq \mathbf{x}_j \; \forall \, i \neq j$ then can partition every observation into its own region: over-fitting?

2. infinitely many trees give identical training error (including test error for hold-out) when feature space continuous.

3. assume growth to fixed height $L$, $\implies$ $\sum_{\ell=0}^{L-1} 2^\ell = 2^L - 1$ nodes, each choosing among $d$ variables.

   $\therefore$ joint optimisation requires $O\left(d(2^L - 1)\right)$ operations *per iteration* of optimisation.

# Tree challenges

1. if $\mathbf{x}_i \neq \mathbf{x}_j \; \forall \, i \neq j$ then can partition every observation into its own region: over-fitting?

2. infinitely many trees give identical training error (including test error for hold-out) when feature space continuous.

3. assume growth to fixed height $L$, $\implies$ $\sum_{\ell=0}^{L-1} 2^\ell = 2^L - 1$ nodes, each choosing among $d$ variables.

   $\therefore$ joint optimisation requires $O\left(d(2^L - 1)\right)$ operations *per iteration* of optimisation.

Finding optimal tree is NP-complete! (Hyafil and Rivest, 1976)

# Binary tree notation

Let $\mathcal{R}_j^{(\ell)}$ denote the region specified by the $j$-th node at depth $\ell$, for $\ell \in \{1, 2, \dots\}$ and $j \in \{0, \dots, 2^\ell - 1\}$.

# Binary tree notation

Let $\mathcal{R}_j^{(\ell)}$ denote the region specified by the $j$-th node at depth $\ell$, for $\ell \in \{1, 2, \dots\}$ and $j \in \{0, \dots, 2^\ell - 1\}$.

**NB** index $j$ from $0 \implies$ represent with binary expansion. ie

$$5_{10} \equiv 101_2 \quad \implies \quad \mathcal{R}_5^{(4)} \equiv \mathcal{R}_{0101}$$

- Number of digits = $\ell$
- Binary value = $j$

# Binary tree notation

Let $\mathcal{R}_j^{(\ell)}$ denote the region specified by the $j$-th node at depth $\ell$, for $\ell \in \{1, 2, \dots\}$ and $j \in \{0, \dots, 2^\ell - 1\}$.

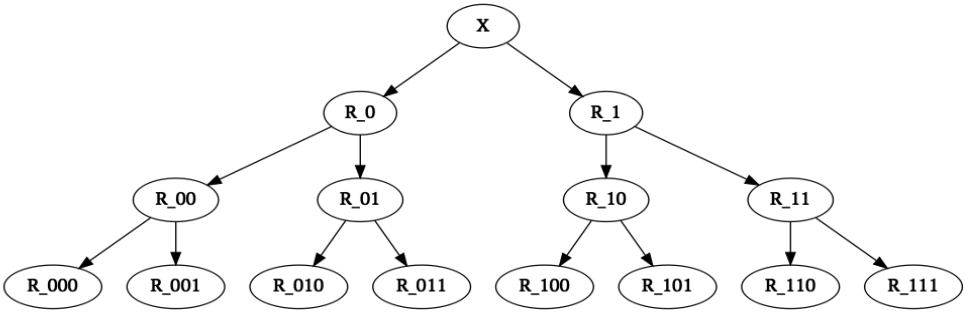
**NB** index $j$ from $0 \implies$ represent with binary expansion. ie

$$5_{10} \equiv 101_2 \quad \implies \quad \mathcal{R}_5^{(4)} \equiv \mathcal{R}_{0101}$$

- Number of digits = $\ell$
- Binary value = $j$

$\implies$ descendents of node identifiable as start with same binary sequence.

# Binary tree notation: binary expansion indexing




# Binary tree notation: tree paths

Path to depth $\ell$ denoted vector

$$\boldsymbol{\tau}_\ell = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}) \quad \tau_{\ell i} \in \{0, 1\}$$

Moving from node at depth $i-1$ to $i$:

- $\tau_{\ell i} = 0$ take left branch;
- $\tau_{\ell i} = 1$ take right branch.

$\mathcal{R}_{\boldsymbol{\tau}_\ell}$ defines region reached along that path, where $\boldsymbol{\tau}_\ell$ matches binary expansion indexing.

# Binary tree notation: tree paths

Path to depth $\ell$ denoted vector

$$\boldsymbol{\tau}_\ell = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}) \quad \tau_{\ell i} \in \{0, 1\}$$

Moving from node at depth $i-1$ to $i$:

- $\tau_{\ell i} = 0$ take left branch;
- $\tau_{\ell i} = 1$ take right branch.

$\mathcal{R}_{\boldsymbol{\tau}_\ell}$ defines region reached along that path, where $\boldsymbol{\tau}_\ell$ matches binary expansion indexing.

Split $\mathcal{R}_{\boldsymbol{\tau}_\ell}$ by obvious extension into $\mathcal{R}_{\boldsymbol{\tau}_\ell 0}$ and $\mathcal{R}_{\boldsymbol{\tau}_\ell 1}$, with

$$\mathcal{R}_{\boldsymbol{\tau}_\ell 0} \cap \mathcal{R}_{\boldsymbol{\tau}_\ell 1} = \emptyset \text{ and } \mathcal{R}_{\boldsymbol{\tau}_\ell 0} \cup \mathcal{R}_{\boldsymbol{\tau}_\ell 1} = \mathcal{R}_{\boldsymbol{\tau}_\ell}$$

# Binary tree notation: axis parallel splits

Split region $\mathcal{R}$ on dimension $i$ at split value $s$ by,

$$\overleftarrow{\mathcal{X}}(\mathcal{R}, i, s) := \{\mathbf{x} \in \mathcal{R} : x_i \leq s\}$$

$$\overrightarrow{\mathcal{X}}(\mathcal{R}, i, s) := \{\mathbf{x} \in \mathcal{R} : x_i > s\}$$

# Binary tree notation: axis parallel splits

Split region $\mathcal{R}$ on dimension $i$ at split value $s$ by,

$$\overleftarrow{\mathcal{X}}(\mathcal{R}, i, s) := \{\mathbf{x} \in \mathcal{R} : x_i \leq s\}$$

$$\overrightarrow{\mathcal{X}}(\mathcal{R}, i, s) := \{\mathbf{x} \in \mathcal{R} : x_i > s\}$$

Then, taking $\mathcal{R}_\emptyset := \mathcal{X}$, recursively define,

$$\mathcal{R}_{\boldsymbol{\tau}\ell 0} := \overleftarrow{\mathcal{X}}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$$

$$\mathcal{R}_{\boldsymbol{\tau}\ell 1} := \overrightarrow{\mathcal{X}}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$$

# Binary tree notation: axis parallel splits

Split region $\mathcal{R}$ on dimension $i$ at split value $s$ by,

$$\overleftarrow{\mathcal{X}}(\mathcal{R}, i, s) := \{\mathbf{x} \in \mathcal{R} : x_i \leq s\}$$

$$\overrightarrow{\mathcal{X}}(\mathcal{R}, i, s) := \{\mathbf{x} \in \mathcal{R} : x_i > s\}$$

Then, taking $\mathcal{R}_\emptyset := \mathcal{X}$, recursively define,

$$\mathcal{R}_{\boldsymbol{\tau}_\ell 0} := \overleftarrow{\mathcal{X}}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$$

$$\mathcal{R}_{\boldsymbol{\tau}_\ell 1} := \overrightarrow{\mathcal{X}}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$$

Tree fully defined by split indices, $i_{\boldsymbol{\tau}_\ell}$, and values, $s_{\boldsymbol{\tau}_\ell}$

Collection $\mathcal{T}$, # nodes $|\mathcal{T}|$, # leaves $\|\mathcal{T}\|_\Lambda$

# CART (Classification And Regression Trees) Breiman et al. (1984)

Corresponding data notation ...

Non-calligraphic versions of $\mathcal{X}$ and $\mathcal{R}$, $X$ and $R$, denote observations in region:

$$\overleftarrow{X}(\mathcal{R}, i, s) := \left\{(\mathbf{x}, y) : (\mathbf{x}, y) \in \mathcal{D}, \mathbf{x} \in \overleftarrow{\mathcal{X}}(\mathcal{R}, i, s)\right\}$$

$$R_{\boldsymbol{\tau}_\ell 0} := \overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$$

Similarly $\overrightarrow{X}$ and $R_{\boldsymbol{\tau}_\ell 1}$

# CART (Classification And Regression Trees) Breiman et al. (1984)

**Initialise:** $\mathcal{T} = \emptyset$, $\mathcal{S} = \{\boldsymbol{\tau}_1 = (0), \boldsymbol{\tau}_1 = (1)\}$

# CART (Classification And Regression Trees) Breiman et al. (1984)

**Initialise:** $\mathcal{T} = \emptyset$, $\mathcal{S} = \{\boldsymbol{\tau}_1 = (0), \boldsymbol{\tau}_1 = (1)\}$

**Repeat:** remove first $\boldsymbol{\tau}_\ell \in \mathcal{S}$ and for this node, solve to find $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$ as,

$$\arg\min_{i,s} \left[ C\left(\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) + C\left(\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) \right]$$

subject to $|\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$ and $|\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$, where $C(\cdot)$ a *cost function*.

# CART (Classification And Regression Trees) Breiman et al. (1984)

**Initialise:** $\mathcal{T} = \emptyset$, $\mathcal{S} = \{\boldsymbol{\tau}_1 = (0), \boldsymbol{\tau}_1 = (1)\}$

**Repeat:** remove first $\boldsymbol{\tau}_\ell \in \mathcal{S}$ and for this node, solve to find $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$ as,

$$\arg\min_{i,s} \left[ C\left(\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) + C\left(\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) \right]$$

subject to $|\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$ and $|\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$, where $C(\cdot)$ a *cost function*.

- No split satisfies constraint? Move to next index in $\mathcal{S}$.

# CART (Classification And Regression Trees) Breiman et al. (1984)

**Initialise:** $\mathcal{T} = \emptyset$, $\mathcal{S} = \{\boldsymbol{\tau}_1 = (0), \boldsymbol{\tau}_1 = (1)\}$

**Repeat:** remove first $\boldsymbol{\tau}_\ell \in \mathcal{S}$ and for this node, solve to find $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$ as,

$$\arg\min_{i,s} \left[ C\left(\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) + C\left(\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) \right]$$

subject to $|\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$ and $|\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$, where $C(\cdot)$ a *cost function*.

- No split satisfies constraint? Move to next index in $\mathcal{S}$.
- $\exists$ split satisfying constraint? Check *stopping criterion*:

# CART (Classification And Regression Trees) Breiman et al. (1984)

**Initialise:** $\mathcal{T} = \emptyset$, $\mathcal{S} = \{\boldsymbol{\tau}_1 = (0), \boldsymbol{\tau}_1 = (1)\}$

**Repeat:** remove first $\boldsymbol{\tau}_\ell \in \mathcal{S}$ and for this node, solve to find $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$ as,

$$\arg\min_{i,s} \left[ C\left(\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) + C\left(\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) \right]$$

subject to $|\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$ and $|\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$, where $C(\cdot)$ a *cost function*.

- No split satisfies constraint? Move to next index in $\mathcal{S}$.
- $\exists$ split satisfying constraint? Check *stopping criterion*:
  - the stopping criterion is not triggered, then:
    - $(i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$ is added to $\mathcal{T}$; and
    - $\boldsymbol{\tau}_{\ell+1} = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}, 0)$ and $\boldsymbol{\tau}_{\ell+1} = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}, 1)$ are added to $\mathcal{S}$.

# CART (Classification And Regression Trees) Breiman et al. (1984)

**Initialise:** $\mathcal{T} = \emptyset$, $\mathcal{S} = \{\boldsymbol{\tau}_1 = (0), \boldsymbol{\tau}_1 = (1)\}$

**Repeat:** remove first $\boldsymbol{\tau}_\ell \in \mathcal{S}$ and for this node, solve to find $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$ as,

$$\arg\min_{i,s} \left[ C\left(\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) + C\left(\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) \right]$$

subject to $|\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$ and $|\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$, where $C(\cdot)$ a *cost function*.

- No split satisfies constraint? Move to next index in $\mathcal{S}$.
- $\exists$ split satisfying constraint? Check *stopping criterion*:
  - the stopping criterion is not triggered, then:
    - $(i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$ is added to $\mathcal{T}$; and
    - $\boldsymbol{\tau}_{\ell+1} = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}, 0)$ and $\boldsymbol{\tau}_{\ell+1} = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}, 1)$ are added to $\mathcal{S}$.
  - the stopping criterion is triggered then discard $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$, resulting in $\mathcal{R}_{\boldsymbol{\tau}_\ell}$ becoming a leaf on that branch.

# CART (Classification And Regression Trees) Breiman et al. (1984)

**Initialise:** $\mathcal{T} = \emptyset$, $\mathcal{S} = \{\boldsymbol{\tau}_1 = (0), \boldsymbol{\tau}_1 = (1)\}$

**Repeat:** remove first $\boldsymbol{\tau}_\ell \in \mathcal{S}$ and for this node, solve to find $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$ as,

$$\arg\min_{i,s} \left[ C\left(\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) + C\left(\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)\right) \right]$$

subject to $|\overleftarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$ and $|\overrightarrow{X}(\mathcal{R}_{\boldsymbol{\tau}_\ell}, i, s)| > 0$, where $C(\cdot)$ a *cost function*.

- No split satisfies constraint? Move to next index in $\mathcal{S}$.
- $\exists$ split satisfying constraint? Check *stopping criterion*:
  - the stopping criterion is not triggered, then:
    - $(i_{\boldsymbol{\tau}_\ell}, s_{\boldsymbol{\tau}_\ell})$ is added to $\mathcal{T}$; and
    - $\boldsymbol{\tau}_{\ell+1} = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}, 0)$ and $\boldsymbol{\tau}_{\ell+1} = (\tau_{\ell 1}, \ldots, \tau_{\ell\ell}, 1)$ are added to $\mathcal{S}$.
  - the stopping criterion is triggered then discard $i_{\boldsymbol{\tau}_\ell}$ and $s_{\boldsymbol{\tau}_\ell}$, resulting in $\mathcal{R}_{\boldsymbol{\tau}_\ell}$ becoming a leaf on that branch.

Continue until $\mathcal{S}$ empty.

# Cost functions

Trees make same prediction for all observations in given region: cost function measures inhomogeneity of a region.

- Mean square error (regression),

$$C(R) := \sum_{(\mathbf{x},y)\in R} (y - \bar{y})^2$$

  where $\bar{y} = \frac{1}{|R|} \sum_{(\mathbf{x},y)\in R} y$

- Misclassification rate (classification)

$$C(R) := \sum_{(\mathbf{x},y)\in R} \mathbb{1}(y \neq \hat{y})$$

  where $\hat{y}$ is the most frequently occurring label in the set $R$.

# Cost functions

- Entropy/deviance (classification)

$$C(R) := \sum_{g=1}^{G} \hat{p}_g \log \hat{p}_g$$

where $\hat{p}_g$ is the empirical proportion of label $g$ in the set $R$.

- Gini index (classification)

$$C(R) := \sum_{g=1}^{G} \hat{p}_g (1 - \hat{p}_g)$$

where $\hat{p}_g$ is the empirical proportion of label $g$ in the set $R$.

# Stopping criterion

Grow a tree until every leaf contains single observation? Massive overfitting!

Possible stopping rules:

- Impose complexity parameter, specifying minimum amount a split must reduce cost.
- Apriori specify height for the tree, any split resulting in nodes with depth exceeding height is prevented.
- Require minimum number of observations falling inside a node.

# Stopping criterion

Grow a tree until every leaf contains single observation? Massive overfitting!

Possible stopping rules:

- Impose complexity parameter, specifying minimum amount a split must reduce cost.
- Apriori specify height for the tree, any split resulting in nodes with depth exceeding height is prevented.
- Require minimum number of observations falling inside a node.

All these are supported by `rpart`, the main decision tree function/package in R.

# Stopping criterion

Grow a tree until every leaf contains single observation? Massive overfitting!

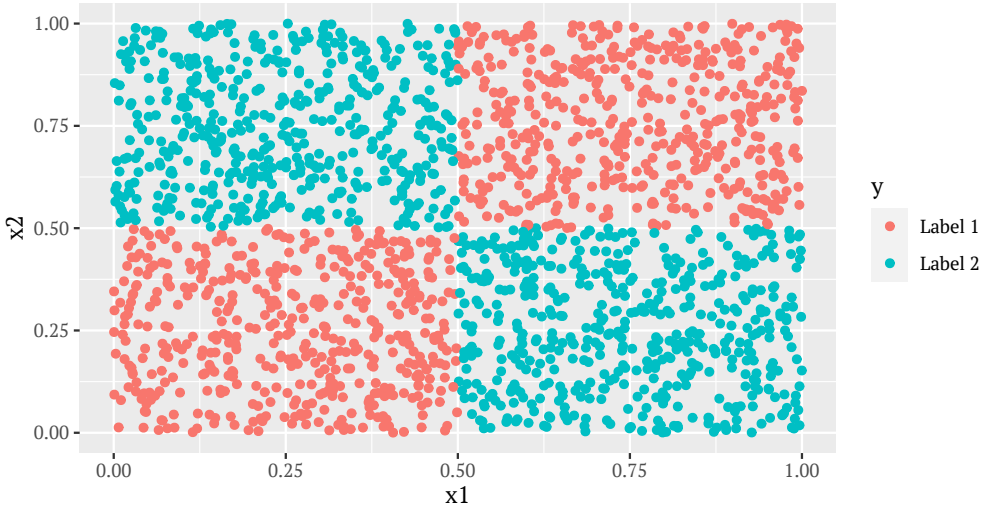
Possible stopping rules:

- Impose complexity parameter, specifying minimum amount a split must reduce cost.
- Apriori specify height for the tree, any split resulting in nodes with depth exceeding height is prevented.
- Require minimum number of observations falling inside a node.

All these are supported by `rpart`, the main decision tree function/package in R.

Complexity parameter seems most well founded perhaps?

# Stopping criterion: problem



# Pruning

Alternative approach: grow a big tree anyway, then prune the branches back!

In other words, identify one or more non-leaf nodes to remove all descendents from, turning them into leaf nodes.

# Pruning

Alternative approach: grow a big tree anyway, then prune the branches back!

In other words, identify one or more non-leaf nodes to remove all descendents from, turning them into leaf nodes.

Pruning $\mathcal{T}$ back to node index $\boldsymbol{\tau}_\ell$ is defined by:

$$\rho(\mathcal{T}, \boldsymbol{\tau}_\ell) := \mathcal{T} \setminus \{(i_{\boldsymbol{\epsilon}_k}, s_{\boldsymbol{\epsilon}_k}) : k \geq \ell \text{ and } \tau_{\ell i} = \epsilon_{ki} \, \forall \, i \in \{1, \ldots, \ell\}\}$$

# Pruning

Alternative approach: grow a big tree anyway, then prune the branches back!

In other words, identify one or more non-leaf nodes to remove all descendents from, turning them into leaf nodes.

Pruning $\mathcal{T}$ back to node index $\boldsymbol{\tau}_\ell$ is defined by:

$$\rho(\mathcal{T}, \boldsymbol{\tau}_\ell) := \mathcal{T} \setminus \{(i_{\boldsymbol{\epsilon}_k}, s_{\boldsymbol{\epsilon}_k}) : k \geq \ell \text{ and } \tau_{\ell i} = \epsilon_{ki} \; \forall \, i \in \{1, \ldots, \ell\}\}$$

**Cost-complexity pruning**: $\mathcal{T}_0$ full tree. Recursively prune $\mathcal{T}_i$ to $\mathcal{T}_{i+1} = \rho(\mathcal{T}_i, \boldsymbol{\tau}_\ell)$ by solving:

$$\boldsymbol{\tau}_\ell = \arg\min_{\boldsymbol{\epsilon}_k} \frac{\widehat{\text{Err}}(\rho(\mathcal{T}_i, \boldsymbol{\epsilon}_k)) - \widehat{\text{Err}}(\mathcal{T}_i)}{\|\mathcal{T}_i\|_\Lambda - \|\rho(\mathcal{T}_i, \boldsymbol{\epsilon}_k)\|_\Lambda}$$

Gives sequence of trees $\mathcal{T}_0, \mathcal{T}_1, \ldots$. Select one using held-out data.

# Pruning as regularisation

Note, reduction in error from a single pruning is:

$$\alpha_i = \frac{\widehat{\text{Err}}(\rho(\mathcal{T}_i, \boldsymbol{\tau}_\ell)) - \widehat{\text{Err}}(\mathcal{T}_i)}{\|\mathcal{T}_i\|_\Lambda - \|\rho(\mathcal{T}_i, \boldsymbol{\tau}_\ell)\|_\Lambda}$$

$$\implies \widehat{\text{Err}}(\rho(\mathcal{T}_i, \boldsymbol{\tau}_\ell)) + \alpha_i \|\rho(\mathcal{T}_i, \boldsymbol{\tau}_\ell)\|_\Lambda = \widehat{\text{Err}}(\mathcal{T}_i) + \alpha_i \|\mathcal{T}_i\|_\Lambda$$

$\therefore$ defines regularised path through tree space!

# Summary

**Pros**

- Very flexible model
- Yet still quite interpretable
- Computationally much more tractable than local methods

# Summary

**Pros**

- Very flexible model
- Yet still quite interpretable
- Computationally much more tractable than local methods

**Cons**

- Struggle to represent additive relationships, so can underperform linear models in simple settings
- Need to tune through pruning and optimal tree unachievable
- Tends to be a very high variance model!

# Summary

**Pros**

- Very flexible model
- Yet still quite interpretable
- Computationally much more tractable than local methods

**Cons**

- Struggle to represent additive relationships, so can underperform linear models in simple settings
- Need to tune through pruning and optimal tree unachievable
- Tends to be a very high variance model!

**Example in notes**

# Bagging (Breiman, 1996)

"Bagging" is an abbreviation of **B**ootstrap **AGG**regat**ING**.

# Bagging (Breiman, 1996)

"Bagging" is an abbreviation of **B**ootstrap **AGG**regat**ING**.

$\hat{f}(\cdot \mid \mathcal{D})$ model (eg trees) fitted to data set $\mathcal{D}$ (call this the *base learner*).

$\hat{f}$ turned into *bagged estimator* by repeatedly fitting to bootstrap resamples of data, then take average.

# Bagging (Breiman, 1996)

"Bagging" is an abbreviation of **B**ootstrap **AGG**regat**ING**.

$\hat{f}(\cdot \mid \mathcal{D})$ model (eg trees) fitted to data set $\mathcal{D}$ (call this the *base learner*).

$\hat{f}$ turned into *bagged estimator* by repeatedly fitting to bootstrap resamples of data, then take average.

Draw $B$ samples size $|\mathcal{D}|$ with replacement from $\mathcal{D}$, $\mathcal{D}^{\star 1}, \ldots, \mathcal{D}^{\star B}$. Then,

$$\hat{f}^{\text{bag}}(\mathbf{x}) := \frac{1}{B} \sum_{b=1}^{B} \hat{f}(\mathbf{x} \mid \mathcal{D}^{\star b})$$

# Bagging (Breiman, 1996)

"Bagging" is an abbreviation of **B**ootstrap **AGG**regat**ING**.

$\hat{f}(\cdot \mid \mathcal{D})$ model (eg trees) fitted to data set $\mathcal{D}$ (call this the *base learner*).

$\hat{f}$ turned into *bagged estimator* by repeatedly fitting to bootstrap resamples of data, then take average.

Draw $B$ samples size $|\mathcal{D}|$ with replacement from $\mathcal{D}$, $\mathcal{D}^{\star 1}, \ldots, \mathcal{D}^{\star B}$. Then,

$$\hat{f}^{\text{bag}}(\mathbf{x}) := \frac{1}{B} \sum_{b=1}^{B} \hat{f}(\mathbf{x} \mid \mathcal{D}^{\star b})$$

$$\hat{f}^{\text{bag}}(\mathbf{x}) = \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]$$

# Why bagging?

$$\mathbb{E}_{D_n}\left[\left(y - \hat{f}(\mathbf{x} \mid D_n)\right)^2\right] = y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)^2\right]$$

# Why bagging?

$$\begin{aligned}\mathbb{E}_{D_n}\left[\left(y-\hat{f}(\mathbf{x}\,|\,D_n)\right)^2\right] &= y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x}\,|\,D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x}\,|\,D_n)^2\right] \\ &\geq y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x}\,|\,D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x}\,|\,D_n)\right]^2\end{aligned}$$

# Why bagging?

$$\begin{aligned}
\mathbb{E}_{D_n}\left[\left(y - \hat{f}(\mathbf{x} \mid D_n)\right)^2\right] &= y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)^2\right] \\
&\geq y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]^2 \\
&= \left(y - \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]\right)^2
\end{aligned}$$

# Why bagging?

$$
\begin{aligned}
\mathbb{E}_{D_n}\left[\left(y - \hat{f}(\mathbf{x} \mid D_n)\right)^2\right] &= y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)^2\right] \\
&\geq y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]^2 \\
&= \left(y - \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]\right)^2 \\
&= \left(y - \hat{f}^{\text{bag}}(\mathbf{x})\right)^2
\end{aligned}
$$

# Why bagging?

$$
\begin{aligned}
\mathbb{E}_{D_n}\left[\left(y - \hat{f}(\mathbf{x} \mid D_n)\right)^2\right] &= y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)^2\right] \\
&\geq y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]^2 \\
&= \left(y - \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]\right)^2 \\
&= \left(y - \hat{f}^{\text{bag}}(\mathbf{x})\right)^2
\end{aligned}
$$

$$
\implies \mathbb{E}_{Y \mid X=\mathbf{x}}\left[\mathbb{E}_{D_n}\left[\left(Y - \hat{f}(\mathbf{x} \mid D_n)\right)^2\right]\right] \geq \mathbb{E}_{Y \mid X=\mathbf{x}}\left[\left(Y - \hat{f}^{\text{bag}}(\mathbf{x})\right)^2\right]
$$

# Why bagging?

$$\begin{aligned}
\mathbb{E}_{D_n}\left[\left(y - \hat{f}(\mathbf{x} \mid D_n)\right)^2\right] &= y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)^2\right] \\
&\geq y^2 - 2y\,\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right] + \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]^2 \\
&= \left(y - \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]\right)^2 \\
&= \left(y - \hat{f}^{\text{bag}}(\mathbf{x})\right)^2
\end{aligned}$$

$$\implies \mathbb{E}_{Y \mid X=\mathbf{x}}\left[\mathbb{E}_{D_n}\left[\left(Y - \hat{f}(\mathbf{x} \mid D_n)\right)^2\right]\right] \geq \mathbb{E}_{Y \mid X=\mathbf{x}}\left[\left(Y - \hat{f}^{\text{bag}}(\mathbf{x})\right)^2\right]$$

Note that $\mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)^2\right] - \mathbb{E}_{D_n}\left[\hat{f}(\mathbf{x} \mid D_n)\right]^2 = \text{Var}_{D_n}\left(\hat{f}(\mathbf{x} \mid D_n)\right)$

# Bagging benefits

- Biggest gains when base learner variance high
  - eg Trees!
  - Grow large trees with minimal (or no) pruning. Relies on bagging procedure directly to avoid overfit.
  - or; prune each tree as was described in the last lecture. **Note:** now we don't need to do cross-validation, because we know on average 36.8% of original data will not be in bootstrap sample so can be used as a test set for pruning.

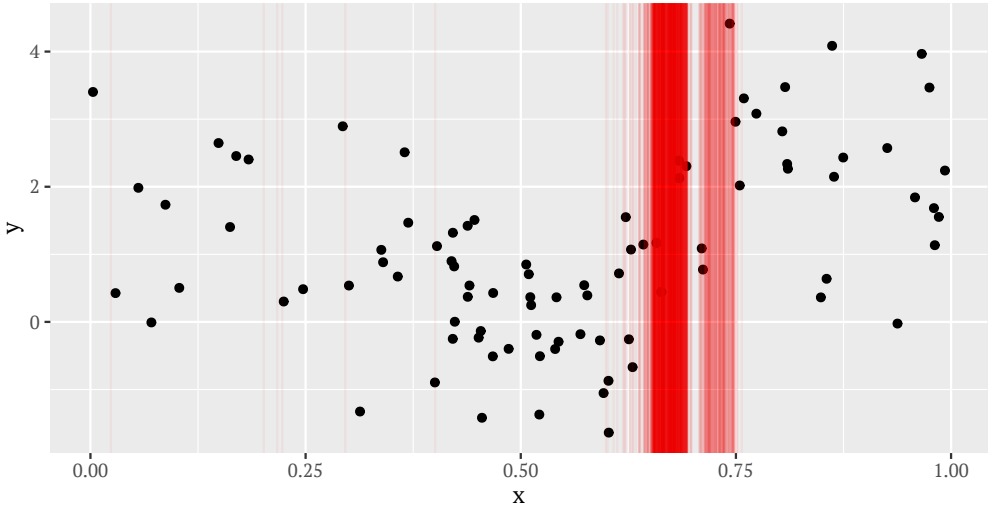
# Bagging benefits

- Biggest gains when base learner variance high
  - eg Trees!
  - Grow large trees with minimal (or no) pruning. Relies on bagging procedure directly to avoid overfit.
  - or; prune each tree as was described in the last lecture. **Note:** now we don't need to do cross-validation, because we know on average 36.8% of original data will not be in bootstrap sample so can be used as a test set for pruning.
- Note can use for any model, including say knn
  - May be additional theory involved
  - Hall and Samworth (2005) show bootstrap resample size must be at most 69%

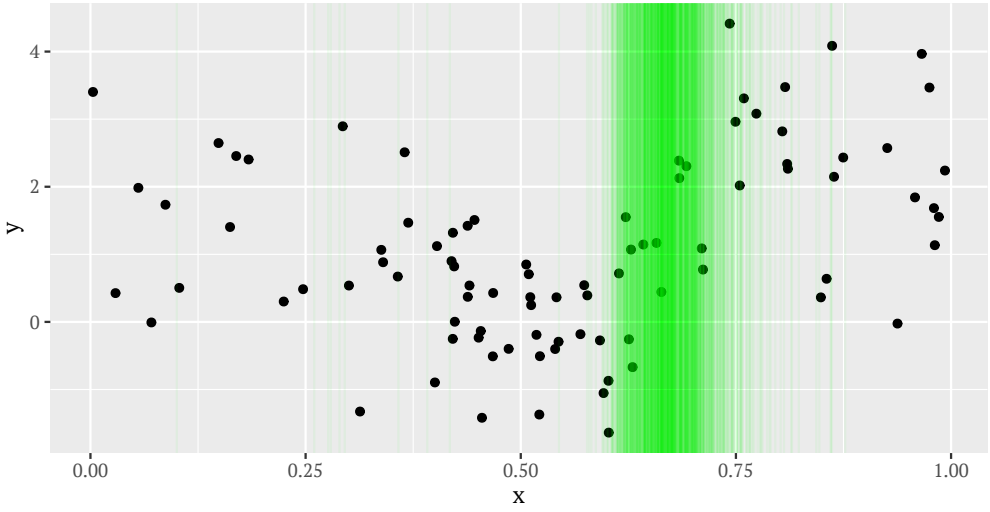
# Bagging benefits

- Biggest gains when base learner variance high
  - eg Trees!
  - Grow large trees with minimal (or no) pruning. Relies on bagging procedure directly to avoid overfit.
  - or; prune each tree as was described in the last lecture. **Note:** now we don't need to do cross-validation, because we know on average 36.8% of original data will not be in bootstrap sample so can be used as a test set for pruning.
- Note can use for any model, including say knn
  - May be additional theory involved
  - Hall and Samworth (2005) show bootstrap resample size must be at most 69%
- Full theoretical gain won't be achieved since we're only approximating $\mathbb{E}_{D_n}$ by bootstrapping

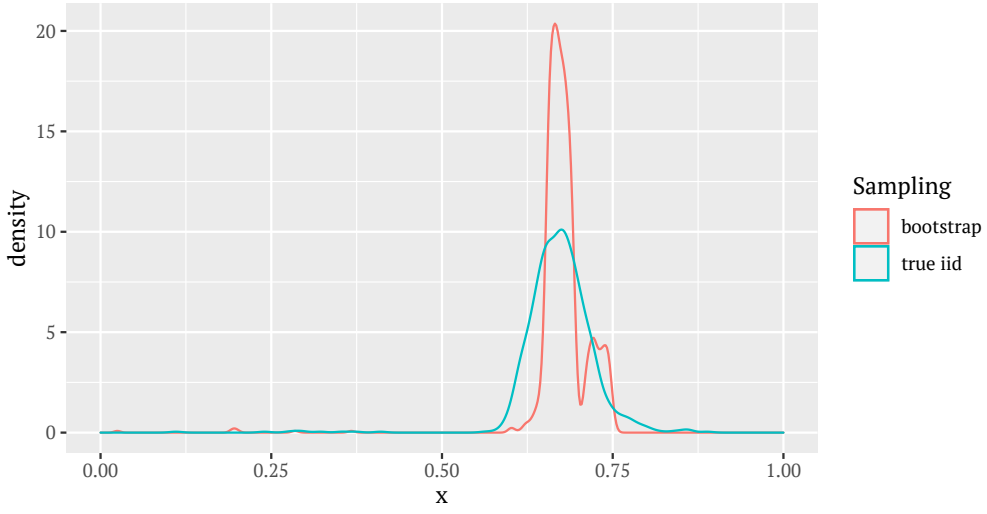
# Bootstrap and Trees — bootstrap split locations



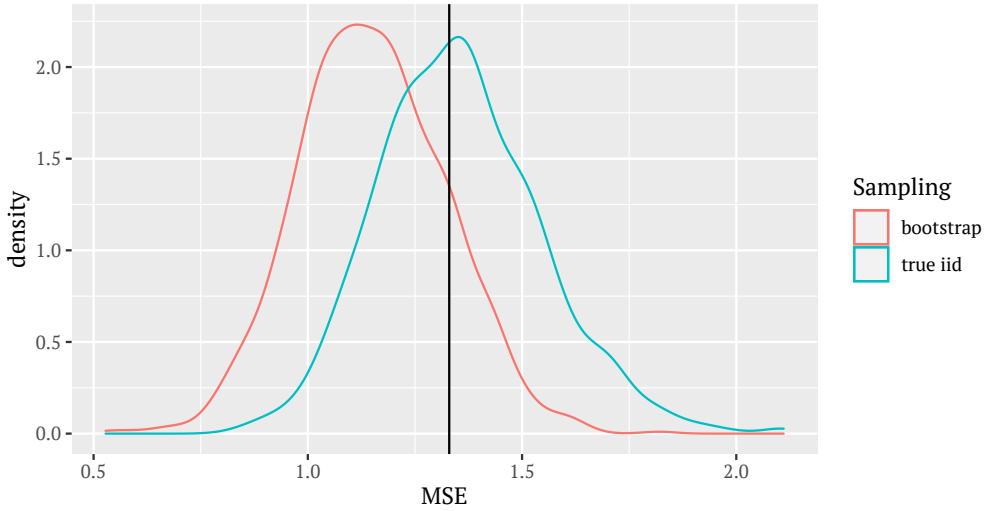
# Bootstrap and Trees — true iid split locations



# Bootstrap and Trees — split location densities



# Bootstrap and Trees — MSE densities



# Bagged classification

Then, to classify a new observation $\mathbf{x}$:

$$\hat{f}^{\text{bag}}(\mathbf{x}) = \arg\max_j \sum_{b=1}^{B} \mathbb{1}\left(\arg\max_k \left(\hat{f}_k(\mathbf{x} \mid \mathcal{D}^{\star b})\right) = j\right)$$

This chooses the class label for which the most bootstrapped trees ‘voted’.

# Bagging class probabilities

Care is required when constructing a probability estimate. Is this a good idea?

$$\hat{p}_j^{\text{bag}}(\mathbf{x}) \stackrel{?}{:=} \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}\left(\arg\max_k \left(\hat{f}_k(\mathbf{x} \mid \mathcal{D}^{\star b})\right) = j\right)$$

# Bagging class probabilities

Care is required when constructing a probability estimate. Is this a good idea?

$$\hat{p}_j^{\text{bag}}(\mathbf{x}) \stackrel{?}{:=} \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}\left(\arg\max_k \left(\hat{f}_k(\mathbf{x} \mid \mathcal{D}^{\star b})\right) = j\right)$$

No! This is proportion of trees voting for given class.

If $p_\ell(\mathbf{x}) = 0.8$, say, and all bootstrapped trees correctly identify $C_\ell$ as most likely response then above estimates the probability as 1!

# Bagging class probabilities

Care is required when constructing a probability estimate. Is this a good idea?

$$\hat{p}_j^{\text{bag}}(\mathbf{x}) \stackrel{?}{:=} \frac{1}{B}\sum_{b=1}^{B} \mathbb{1}\left(\arg\max_k \left(\hat{f}_k(\mathbf{x} \,|\, \mathcal{D}^{\star b})\right) = j\right)$$

No! This is proportion of trees voting for given class.

If $p_\ell(\mathbf{x}) = 0.8$, say, and all bootstrapped trees correctly identify $C_\ell$ as most likely response then above estimates the probability as 1!

Instead, directly average tree output probabilities

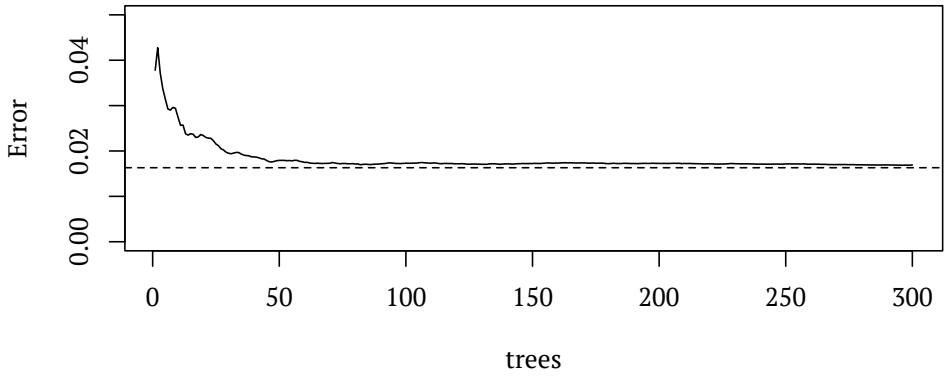
$$\hat{p}_j^{\text{bag}}(\mathbf{x}) := \frac{1}{B}\sum_{b=1}^{B} \hat{f}_j(\mathbf{x} \,|\, \mathcal{D}^{\star b})$$

# Out of bag error

Not every observation included in every bootstrap sample:

$$
\begin{aligned}
&\mathbb{P}\left((\mathbf{x}_i, y_i) \in \mathcal{D}^{\star b}\right) \\
&\quad = 1 - \mathbb{P}\left((\mathbf{x}_i, y_i) \notin \mathcal{D}^{\star b}\right) \\
&\quad = 1 - \mathbb{P}\left((\mathbf{x}_1^{\star b}, y_1^{\star b}) \neq (\mathbf{x}_i, y_i) \cap \cdots \cap (\mathbf{x}_n^{\star b}, y_n^{\star b}) \neq (\mathbf{x}_i, y_i)\right) \\
&\quad = 1 - \prod_{j=1}^{n} \mathbb{P}\left((\mathbf{x}_j^{\star b}, y_j^{\star b}) \neq (\mathbf{x}_i, y_i)\right) \qquad \text{by indep sampling with replacement} \\
&\quad = 1 - \left(1 - \frac{1}{n}\right)^n \\
&\quad \approx 1 - e^{-1} \approx 0.632
\end{aligned}
$$

# OOB




# When bagging?

**Pros**

- If you have a lot of data, bagging makes sense because the bootstrapped distribution will be good approx of population distribution.

# When bagging?

**Pros**

- If you have a lot of data, bagging makes sense because the bootstrapped distribution will be good approx of population distribution.
- Bagging is not restricted to trees!

# When bagging?

**Pros**

- If you have a lot of data, bagging makes sense because the bootstrapped distribution will be good approx of population distribution.
- Bagging is not restricted to trees!
- Most useful when it is desirable to reduce the variance of a predictor.
  - under the (inaccurate) independence assumption, variance will reduce by a factor of $\sim 1/B$ for $B$ bootstrap resamples.

# When bagging?

**Pros**

- If you have a lot of data, bagging makes sense because the bootstrapped distribution will be good approx of population distribution.
- Bagging is not restricted to trees!
- Most useful when it is desirable to reduce the variance of a predictor.
  - under the (inaccurate) independence assumption, variance will reduce by a factor of $\sim 1/B$ for $B$ bootstrap resamples.
- We can estimate the error in bagging *without* needing cross-validation, since any given observation will not be used in around 36.8% of the models (some care needed: see ESL, p.251). "Out Of Bag" (OOB) error estimation (cf 3-CV)

# When bagging?

**Pros**

- If you have a lot of data, bagging makes sense because the bootstrapped distribution will be good approx of population distribution.
- Bagging is not restricted to trees!
- Most useful when it is desirable to reduce the variance of a predictor.
  - under the (inaccurate) independence assumption, variance will reduce by a factor of $\sim 1/B$ for $B$ bootstrap resamples.
- We can estimate the error in bagging *without* needing cross-validation, since any given observation will not be used in around 36.8% of the models (some care needed: see ESL, p.251). "Out Of Bag" (OOB) error estimation (cf 3-CV)

**Cons**

- We lose interpretability as the final estimate is not a tree.

# When bagging?

**Pros**

- If you have a lot of data, bagging makes sense because the bootstrapped distribution will be good approx of population distribution.
- Bagging is not restricted to trees!
- Most useful when it is desirable to reduce the variance of a predictor.
  - under the (inaccurate) independence assumption, variance will reduce by a factor of $\sim 1/B$ for $B$ bootstrap resamples.
- We can estimate the error in bagging *without* needing cross-validation, since any given observation will not be used in around 36.8% of the models (some care needed: see ESL, p.251). "Out Of Bag" (OOB) error estimation (cf 3-CV)

**Cons**

- We lose interpretability as the final estimate is not a tree.
- Computational cost is multiplied by a factor of at least $B$.

# Silk purse or sow's ear?

*"Bagging goes a ways toward making a silk purse out of a sow's ear, especially if the sow's ear is twitchy. It is a relatively easy way to improve an existing method, since all that needs adding is a loop in front that selects the bootstrap sample and sends it to the procedure and a back end that does the aggregation. What one loses, with the trees, is a simple and interpretable structure. What one gains is increased accuracy."*

*— Breiman (1996), p.137*

# Random forests

Bagging improved on single CART model by reducing the variance through resampling. But, in fact the bagged models are still quite correlated.

$\Longrightarrow$ less variance reduction achieved by averaging.

# Random forests

Bagging improved on single CART model by reducing the variance through resampling. But, in fact the bagged models are still quite correlated.

$\Longrightarrow$ less variance reduction achieved by averaging.

Can we make them more independent in order to produce a better variance reduction?

# Random forests

Bagging improved on single CART model by reducing the variance through resampling. But, in fact the bagged models are still quite correlated.

$\implies$ less variance reduction achieved by averaging.

Can we make them more independent in order to produce a better variance reduction?

**Random forests** achieve this by not only resampling observations, but by restricting the model to *random subspaces*, $\mathcal{X}' \subset \mathcal{X}$.

$\implies$ Random forests $=$ Bagging $+$ Random Subspaces

# Random forest algorithm

1. Take a bootstrap resample $\mathcal{D}^\star$ of the data, as for bagging.
2. Fit the CART algorithm to bootstrap sample, *but* each time a split is made, only optimise over a subset of $m < d$ variables.
   - Perhaps prune individual trees
   - Many algorithms (eg `ranger` in R) grow to purity without pruning
3. Average the prediction of all $B$ trees

# Random forest algorithm

1. Take a bootstrap resample $\mathcal{D}^\star$ of the data, as for bagging.
2. Fit the CART algorithm to bootstrap sample, *but* each time a split is made, only optimise over a subset of $m < d$ variables.
   - Perhaps prune individual trees
   - Many algoriths (eg `ranger` in R) grow to purity without pruning
3. Average the prediction of all $B$ trees

The random subspaces causes less correlation in the trees ... combined with bagging this means the trees are a lot less correlated and variance is reduced substantially.

# Breiman (2001) Theory

1. As number of trees $B \to \infty$, forest converges to limiting value of generalisation error for base learners ( $\implies$ won't overfit by adding trees)

# Breiman (2001) Theory

1. As number of trees $B \to \infty$, forest converges to limiting value of generalisation error for base learners ( $\implies$ won't overfit by adding trees)
2. Generalisation error asymptotically bounded by

$$\mathcal{E} \leq \frac{\bar{\rho}(1 - S^2)}{S^2}$$

where $S$ is how 'strong', and $\bar{\rho}$ is how correlated, randomised base learner models in the forest are.

# Breiman (2001) Theory

1. As number of trees $B \to \infty$, forest converges to limiting value of generalisation error for base learners ( $\implies$ won't overfit by adding trees)
2. Generalisation error asymptotically bounded by

$$\mathcal{E} \leq \frac{\bar{\rho}(1 - S^2)}{S^2}$$

where $S$ is how 'strong', and $\bar{\rho}$ is how correlated, randomised base learner models in the forest are.

**Aside:** Really?

# Breiman (2001) Theory

1. As number of trees $B \to \infty$, forest converges to limiting value of generalisation error for base learners ( $\implies$ won't overfit by adding trees)
2. Generalisation error asymptotically bounded by

$$\mathcal{E} \leq \frac{\bar{\rho}(1 - S^2)}{S^2}$$

where $S$ is how 'strong', and $\bar{\rho}$ is how correlated, randomised base learner models in the forest are.

**Aside:** Really? Consider binary classification Assume at feature value $\mathbf{x}$ we have $B$ *independent* classifiers, each with a 0-1 loss of 0.4.

$\implies$ distribution of number of votes for the correct label is Binomial$(B, 0.6)$. Then,

$$\mathbb{P}(> B/2 \text{ votes for correct class}) \to 1 \text{ as } B \to \infty$$

# Variable importance from forests (I)

**Option 1**

For each feature $\mathbf{x}_{\cdot j}, j = 1, \ldots, d$, loop over each tree in the forest

- find all nodes that make a split on $\mathbf{x}_{\cdot j}$
- compute the improvement in loss criterion the split causes (eg accuracy/Gini/etc)
- sum improvements across nodes in the tree

Finally, sum improvement across all trees.

# Variable importance from forests (I)

**Option 1**

For each feature $\mathbf{x}_{\cdot j}, j = 1, \ldots, d$, loop over each tree in the forest

- find all nodes that make a split on $\mathbf{x}_{\cdot j}$
- compute the improvement in loss criterion the split causes (eg accuracy/Gini/etc)
- sum improvements across nodes in the tree

Finally, sum improvement across all trees.

This is the method used by `randomForest::varImpPlot`.

# Variable importance from forests (II)

**Option 2**

Already discussed that bagging enables out of bag error estimation.

To compute out of bag *variable importance*, for each feature $\mathbf{x}_{\cdot j}, j = 1, \ldots, d$

- for each tree, take the out of bag samples, $\mathcal{D}^{\text{oob}}$, and compute the predictive accuracy for that tree;
- next, take $\mathcal{D}^{\text{oob}}$ and *randomly permute* all the entries for $j$th variable across observations;
- predict for this permuted $\mathcal{D}^{\text{oob}}$ and compute change in predictive accuracy for that tree.

Finally average the decrease in accuracy over all trees.

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Has an integrated measure of variable importance

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Has an integrated measure of variable importance
- Works well with high dimensional data

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Has an integrated measure of variable importance
- Works well with high dimensional data
- Extensive tuning not really needed

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Has an integrated measure of variable importance
- Works well with high dimensional data
- Extensive tuning not really needed

**Cons**

- Often quite suboptimal for regression

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Has an integrated measure of variable importance
- Works well with high dimensional data
- Extensive tuning not really needed

**Cons**

- Often quite suboptimal for regression
- Same extrapolation issue as trees

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Has an integrated measure of variable importance
- Works well with high dimensional data
- Extensive tuning not really needed

**Cons**

- Often quite suboptimal for regression
- Same extrapolation issue as trees
- Memory hungry model

# Random Forest discussion

**Pros**

- Inherits the advantages of bagging and trees
- Very easy to parallelise
- Has an integrated measure of variable importance
- Works well with high dimensional data
- Extensive tuning not really needed

**Cons**

- Often quite suboptimal for regression
- Same extrapolation issue as trees
- Memory hungry model
- Prediction slow for big ensembles

# Boosting

Q: Michael Kearns (1988): *"is it possible for weak learners be combined to create a strong learner with low bias?"* He call this the "Hypothesis Boosting Problem".

# Boosting

Q: Michael Kearns (1988): *"is it possible for weak learners be combined to create a strong learner with low bias?"* He call this the "Hypothesis Boosting Problem".

A: Yes! (Schapire, 1990) But, AdaBoost (Freund and Schapire, 1997) first practically usable approach.

# Boosting

Q: Michael Kearns (1988): *"is it possible for weak learners be combined to create a strong learner with low bias?"* He call this the "Hypothesis Boosting Problem".

A: Yes! (Schapire, 1990) But, AdaBoost (Freund and Schapire, 1997) first practically usable approach.

Again, build ensemble, but this time each classifier very *weak* and build *iteratively* to improve:

$$\hat{F}^{(B)}(\mathbf{x}) = \sum_{b=1}^{B} \alpha_b \hat{f}^{(b)}(\mathbf{x})$$

($\hat{F}^{(B)}$ is ensemble, not CDF!)

# AdaBoost (binary classification, $\mathcal{Y} = \{-1, +1\}$)

$\mathbf{w} = (w_1, \ldots, w_n)$, for each obs in $\mathcal{D}$. Initialise, $w_i = 1/n \; \forall \, i$ and $\hat{f}(\mathbf{x}) = 0 \; \forall \, \mathbf{x} \in \mathcal{X}$. For iterations $b = 1, \ldots, B$:

1. Fit tree to weighted dataset /w loss

$$\hat{f}^{(b)} = \arg \min_{f \in \mathcal{T}_{(h)}} \sum_{\{i : f(\mathbf{x}_i) \neq y_i\}} w_i$$

2. Compute weighted error:

$$\varepsilon_b = \sum_{i=1}^{n} w_i \mathbb{1} \left\{ \hat{f}^{(b)}(\mathbf{x}_i) \neq y_i \right\}$$

3. If $\varepsilon \geq \frac{1}{2}$, discard $\hat{f}^{(b)}$ and terminate. Else, if $\varepsilon < \frac{1}{2}$, retain, set coefficient

$$\alpha_b = \frac{1}{2} \log \left( \frac{1 - \varepsilon}{\varepsilon} \right) \quad \text{and update weights:} \quad w_i \leftarrow \frac{w_i \exp \left( -\alpha_b \hat{f}^{(b)}(\mathbf{x}_i) y_i \right)}{2\sqrt{\varepsilon(1 - \varepsilon)}}$$

# AdaBoost (binary classification, $\mathcal{Y} = \{-1, +1\}$)

Upon termination, up to $B$ weak learners $\hat{f}^{(b)}$ and coefficient weights $\alpha_b$, so full ensemble,

$$\hat{F}(\mathbf{x}) = \sum_b \alpha_b \hat{f}^{(b)}(\mathbf{x})$$

# AdaBoost (binary classification, $\mathcal{Y} = \{-1, +1\}$)

Upon termination, up to $B$ weak learners $\hat{f}^{(b)}$ and coefficient weights $\alpha_b$, so full ensemble,

$$\hat{F}(\mathbf{x}) = \sum_b \alpha_b \hat{f}^{(b)}(\mathbf{x})$$

Scoring classifier, so sign indicates label.

Early termination due to $\varepsilon_b \geq \frac{1}{2} \implies$ weak learner no better than guessing, so don't add more weak learners.

# AdaBoost (binary classification, $\mathcal{Y} = \{-1, +1\}$)

Upon termination, up to $B$ weak learners $\hat{f}^{(b)}$ and coefficient weights $\alpha_b$, so full ensemble,

$$\hat{F}(\mathbf{x}) = \sum_b \alpha_b \hat{f}^{(b)}(\mathbf{x})$$

Scoring classifier, so sign indicates label.

Early termination due to $\varepsilon_b \geq \frac{1}{2} \implies$ weak learner no better than guessing, so don't add more weak learners.

AdaBoost aggressively drives training/apparent error to zero in $O(\log n)$ iterations.

# Boosting = greedy forward stagewise (Friedman et al., 2000)

Set $\varepsilon_i = y_i$ for $i \in \{1, \ldots, n\}$. For $b = 1, \ldots, B$, iterate:

1. Fit a model (eg tree) $\hat{f}^{(b)}(\cdot)$ and multiplier $\lambda_b$ to the response $\varepsilon_1, \ldots, \varepsilon_n$ as,

$$\{\hat{f}^{(b)}(\cdot), \lambda_b\} \leftarrow \arg \min_{f \in \mathcal{T}_{(h)}, \lambda_b \in \mathbb{R}} \sum_{i=1}^{n} \left(\varepsilon_i - \lambda_b \hat{f}^{(b)}(\mathbf{x})\right)^2$$

2. Update residuals

$$\varepsilon_i \leftarrow \varepsilon_i - \lambda_b \hat{f}^{(b)}(\mathbf{x}) \quad \forall\, i$$

# Boosting = greedy forward stagewise (Friedman et al., 2000)

Set $\varepsilon_i = y_i$ for $i \in \{1, \ldots, n\}$. For $b = 1, \ldots, B$, iterate:

1. Fit a model (eg tree) $\hat{f}^{(b)}(\cdot)$ and multiplier $\lambda_b$ to the response $\varepsilon_1, \ldots, \varepsilon_n$ as,

$$\{\hat{f}^{(b)}(\cdot), \lambda_b\} \leftarrow \arg \min_{f \in \mathcal{T}_{(h)}, \lambda_b \in \mathbb{R}} \sum_{i=1}^{n} \left(\varepsilon_i - \lambda_b \hat{f}^{(b)}(\mathbf{x})\right)^2$$

2. Update residuals

$$\varepsilon_i \leftarrow \varepsilon_i - \lambda_b \hat{f}^{(b)}(\mathbf{x}) \quad \forall\, i$$

Output as final boosted model

$$\hat{F}^{(B)}(\mathbf{x}) = \sum_{b=1}^{B} \lambda_b \hat{f}^{(b)}(\mathbf{x})$$

# Gradient boosting machines (I)

Gradient descent:

$$\theta_{i+1} = \theta_i - \gamma \nabla g(\theta)\Big|_{\theta=\theta_i}$$

# Gradient boosting machines (I)

Gradient descent:

$$\theta_{i+1} = \theta_i - \gamma \nabla g(\theta)\Big|_{\theta=\theta_i}$$

$$\implies \hat{\theta} = \theta_0 - \sum \gamma \nabla g(\theta)\Big|_{\theta=\theta_i}$$

Very evocative of boosting!

# Gradient boosting machines (II)

$$\mathcal{E}(F) = \mathbb{E}_{XY}\mathcal{L}(Y, F(X)) = \mathbb{E}_X\left[\mathbb{E}_{Y\,|\,X=\mathbf{x}}\left[\mathcal{L}(Y, F(\mathbf{x}))\right]\right]$$

# Gradient boosting machines (II)

$$\mathcal{E}(F) = \mathbb{E}_{XY}\mathcal{L}(Y, F(X)) = \mathbb{E}_X\left[\mathbb{E}_{Y \mid X=\mathbf{x}}\left[\mathcal{L}(Y, F(\mathbf{x}))\right]\right]$$
$$\mathcal{E}(F(\mathbf{x})) := \mathbb{E}_{Y \mid X}\mathcal{L}(Y, F(\mathbf{x}))$$

Do gradient descent on $\mathcal{E}(F(\mathbf{x}))$ as a function of $F(\mathbf{x})$ ... ie in function space.

# Gradient boosting machines (II)

$$\mathcal{E}(F) = \mathbb{E}_{XY}\mathcal{L}(Y, F(X)) = \mathbb{E}_X\left[\mathbb{E}_{Y\,|\,X=\mathbf{x}}\left[\mathcal{L}(Y, F(\mathbf{x}))\right]\right]$$
$$\mathcal{E}(F(\mathbf{x})) := \mathbb{E}_{Y\,|\,X}\mathcal{L}(Y, F(\mathbf{x}))$$

Do gradient descent on $\mathcal{E}(F(\mathbf{x}))$ as a function of $F(\mathbf{x})$ ... ie in function space.

$$\hat{F}^{(B+1)}(\mathbf{x}) = \hat{F}^{(B)}(\mathbf{x}) - \gamma\nabla\mathcal{E}(F(\mathbf{x}))\Big|_{F(\mathbf{x})=\hat{F}^{(B)}(\mathbf{x})}$$

# Gradient boosting machines (II)

$$\mathcal{E}(F) = \mathbb{E}_{XY}\mathcal{L}(Y, F(X)) = \mathbb{E}_X\left[\mathbb{E}_{Y\,|\,X=\mathbf{x}}\left[\mathcal{L}(Y, F(\mathbf{x}))\right]\right]$$
$$\mathcal{E}(F(\mathbf{x})) := \mathbb{E}_{Y\,|\,X}\mathcal{L}(Y, F(\mathbf{x}))$$

Do gradient descent on $\mathcal{E}(F(\mathbf{x}))$ as a function of $F(\mathbf{x})$ ... ie in function space.

$$\hat{F}^{(B+1)}(\mathbf{x}) = \hat{F}^{(B)}(\mathbf{x}) - \gamma\nabla\mathcal{E}(F(\mathbf{x}))\Big|_{F(\mathbf{x})=\hat{F}^{(B)}(\mathbf{x})}$$

Notation,

$$\eta_B(\mathbf{x}) := \nabla\mathcal{E}(F(\mathbf{x}))\Big|_{F(\mathbf{x})=\hat{F}^{(B)}(\mathbf{x})} = \mathbb{E}_{Y\,|\,X}\left[\frac{\partial\,\mathcal{L}(Y, F(\mathbf{x}))}{\partial\,F(\mathbf{x})}\Big|_{F(\mathbf{x})=\hat{F}^{(B)}(\mathbf{x})}\right]$$

# Gradient boosting machines (III)

We're doing boosting, not gradient descent really, so after computing gradient at current location, compute coefficient:

$$\alpha_{B+1} = \arg\min_{\alpha} \mathbb{E}_{XY} \mathcal{L}\left(Y, \hat{F}^{(B)}(X) - \alpha \eta_B(X)\right)$$

and add $\hat{f}^{(B+1)}(\mathbf{x}) = \eta_B(\mathbf{x})$ and $\alpha_{B+1}$ to the ensemble.

# Gradient boosting machines (III)

We're doing boosting, not gradient descent really, so after computing gradient at current location, compute coefficient:

$$\alpha_{B+1} = \arg\min_{\alpha} \mathbb{E}_{XY} \mathcal{L}\left(Y, \hat{F}^{(B)}(X) - \alpha \eta_B(X)\right)$$

and add $\hat{f}^{(B+1)}(\mathbf{x}) = \eta_B(\mathbf{x})$ and $\alpha_{B+1}$ to the ensemble.

**BUT** we can't compute this gradient!

# Gradient boosting machines (III)

We're doing boosting, not gradient descent really, so after computing gradient at current location, compute coefficient:

$$\alpha_{B+1} = \arg\min_{\alpha} \mathbb{E}_{XY} \mathcal{L}\left(Y, \hat{F}^{(B)}(X) - \alpha \eta_B(X)\right)$$

and add $\hat{f}^{(B+1)}(\mathbf{x}) = \eta_B(\mathbf{x})$ and $\alpha_{B+1}$ to the ensemble.

**BUT** we can't compute this gradient! Compute instead:

$$\hat{\eta}_B(\mathbf{x}_i) = \left.\frac{\partial \mathcal{L}\left(y_i, F(\mathbf{x})\right)}{\partial F(\mathbf{x})}\right|_{F(\mathbf{x})=\hat{F}^{(B)}(\mathbf{x}_i)} \qquad \forall\, i$$

and fit base learner to $(-\hat{\eta}_B(\mathbf{x}_1), \ldots, -\hat{\eta}_B(\mathbf{x}_n)) \in \mathbb{R}^n$ by ERM for square loss.

# Gradient boosting machines (IV)

In other words, add to ensemble:

$$\hat{f}^{(B+1)} = \arg \min_{\hat{f} \in \mathcal{F}, \beta \in \mathbb{R}} \sum_{i=1}^{n} \left( -\hat{\eta}_B(\mathbf{x}_i) - \beta \hat{f}(\mathbf{x}_i) \right)^2$$

$$\alpha_{B+1} = \arg \min_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} \mathcal{L} \left( y_i, \hat{F}^{(B)}(\mathbf{x}_i) + \alpha \hat{f}^{(B+1)}(\mathbf{x}_i) \right)$$

# Gradient boosting machines (IV)

In other words, add to ensemble:

$$\hat{f}^{(B+1)} = \arg \min_{\hat{f} \in \mathcal{F}, \beta \in \mathbb{R}} \sum_{i=1}^{n} \left( -\hat{\eta}_B(\mathbf{x}_i) - \beta \hat{f}(\mathbf{x}_i) \right)^2$$

$$\alpha_{B+1} = \arg \min_{\alpha \in \mathbb{R}} \sum_{i=1}^{n} \mathcal{L} \left( y_i, \hat{F}^{(B)}(\mathbf{x}_i) + \alpha \hat{f}^{(B+1)}(\mathbf{x}_i) \right)$$

General and powerful approach: just convex differentiable loss required, since gradient approx by arbitrary base learner via least squares.

# References I

Breiman, L. (2001). Random forests. *Machine Learning* **45**, 5–32. DOI: 10.1023/A:1010933404324

Breiman, L. (1996). Bagging predictors. *Machine Learning* **24**, 123–140. DOI: 10.1007/BF00058655

Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J. (1984). Classification and regression trees, 1st ed. Chapman & Hall/CRC. ISBN: 978-1351460484

Freund, Y., Schapire, R.E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences* **55**(1), 119–139. DOI: 10.1006/jcss.1997.1504

Friedman, J., Hastie, T., Tibshirani, R. (2000). Additive logistic regression: A statistical view of boosting. *The Annals of Statistics* **28**(2), 337–407. DOI: 10.1214/aos/1016218223

# References II

Hall, P., Samworth, R.J. (2005). Properties of bagged nearest neighbour classifiers. *Journal of the Royal Statistical Society: Series B* **67**(3), 363–379. DOI: 10.1111/j.1467-9868.2005.00506.x

Hyafil, L., Rivest, R.L. (1976). Constructing optimal binary decision trees is NP-complete. *Information Processing Letters* **5**(1), 15–17. DOI: 10.1016/0020-0190(76)90095-8

Kearns, M. (1988). Thoughts on hypothesis boosting. *Machine Learning Class Project (unpublished)* 1–9. URL `http://www.cis.upenn.edu/~mkearns/papers/boostnote.pdf`

Schapire, R.E. (1990). The strength of weak learnability. *Machine Learning* **5**, 197–227. DOI: 10.1023/A:1022648800760