# ORIE 4741: Learning with Big Messy Data

## Trees

Professor Udell

Operations Research and Information Engineering
Cornell

October 21, 2021

# Announcements 10/19/21

- section this week: data structures
- hw4 out (except last problem), due 10/28
  - save slip days for emergencies
- begin work on project midterm report

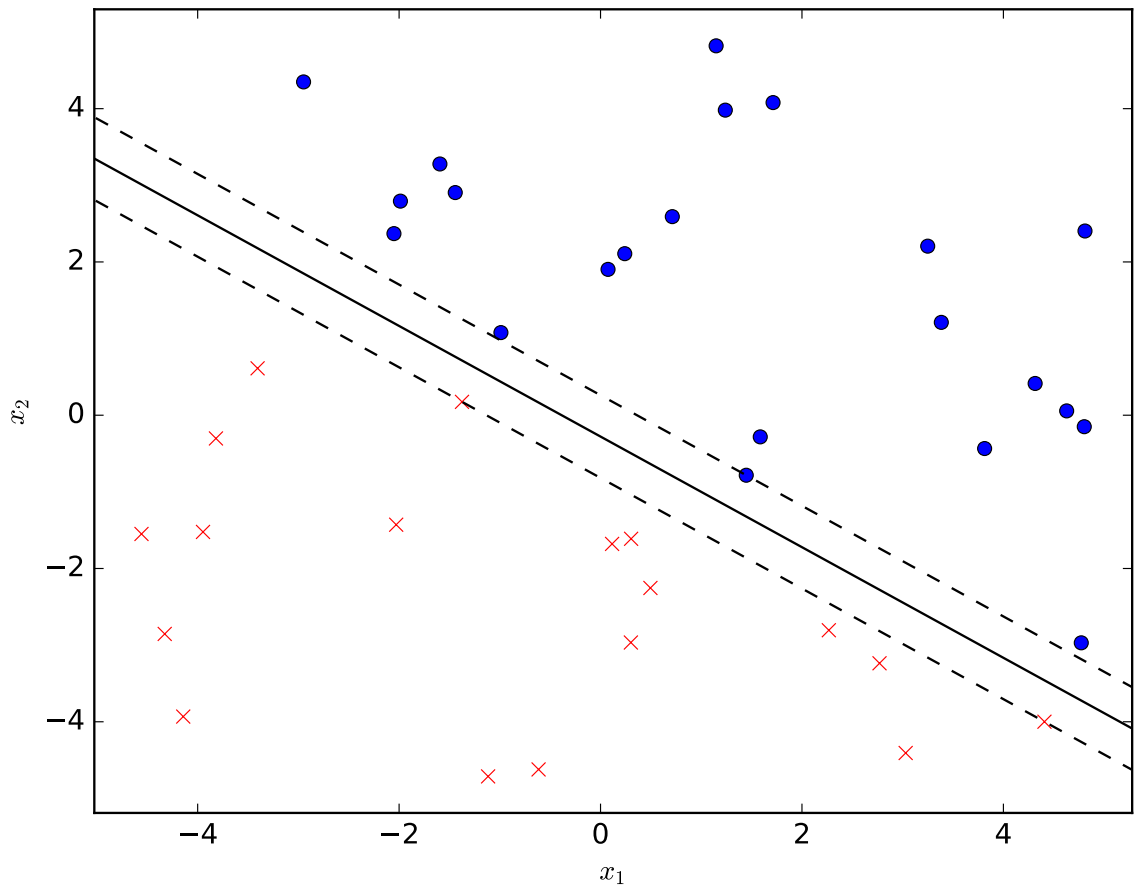
# Announcements 10/21/21

- hw4 out (except last problem) due 10am 11/1
  - save slip days for emergencies
- project midterm report due 11:59pm 11/1

# Outline

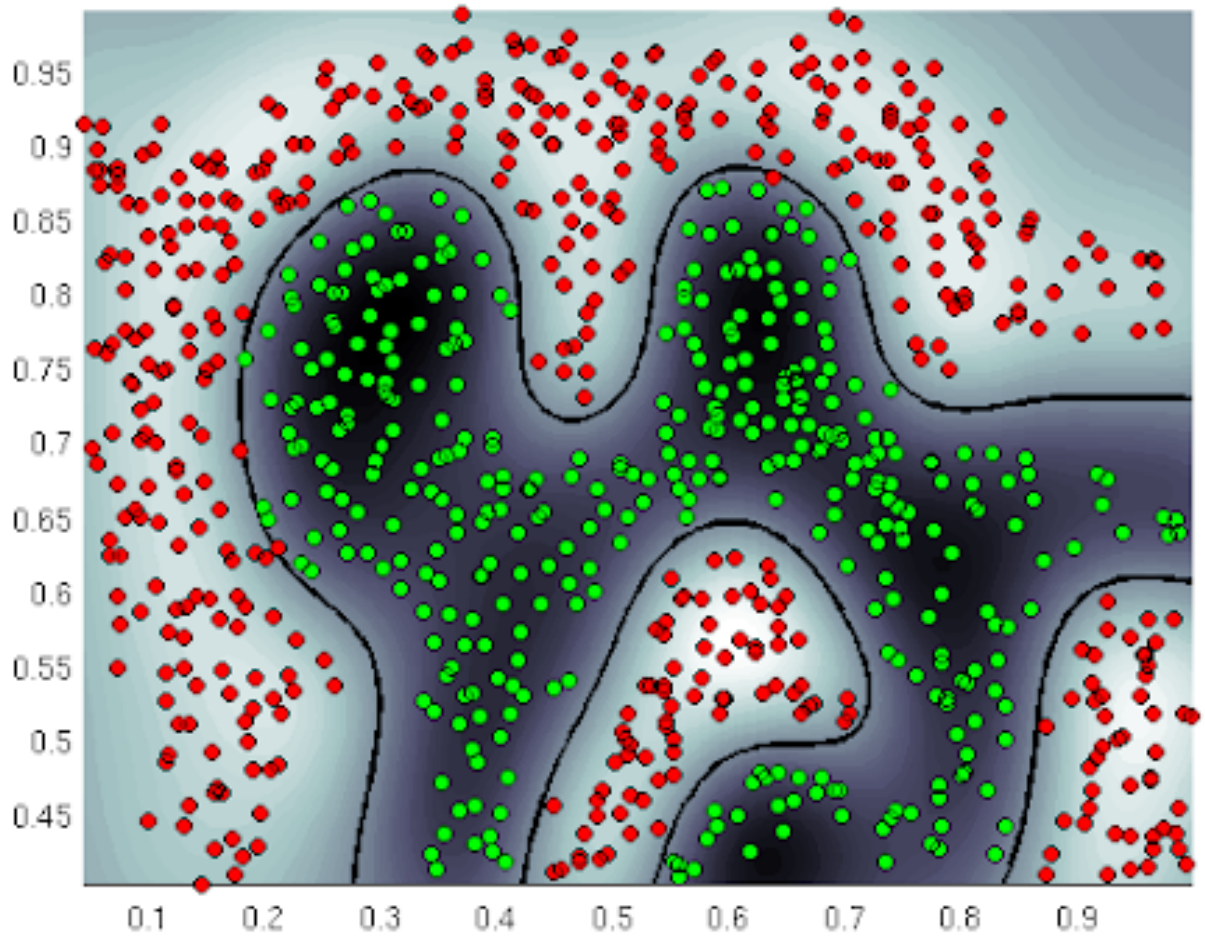
# Linear classification

So far we've looked at method for classification where the data can be (reasonably) separated by linear classifiers...



# Non-linear classification

... but the world is full of non-linear data!



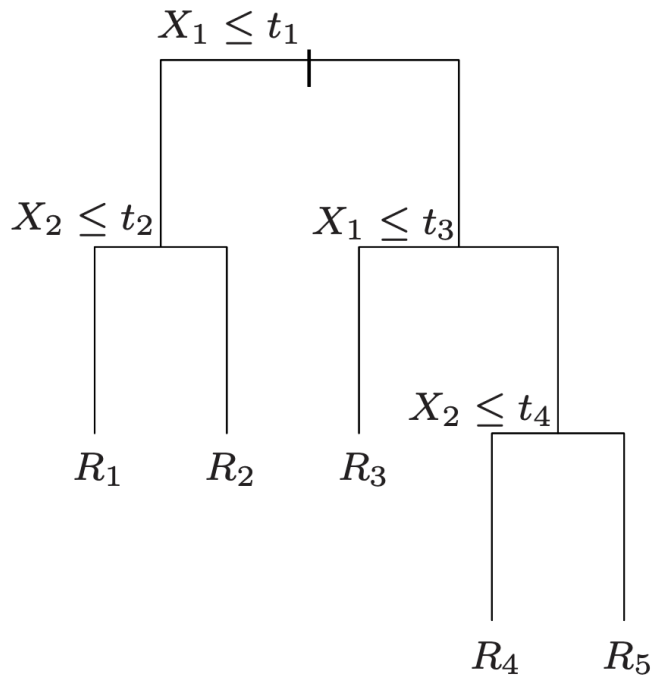

source:
http://openclassroom.stanford.edu/MainFolder/DocumentPage.php?

# Outline

# Decision trees

Decision trees are a non-linear machine learning model that

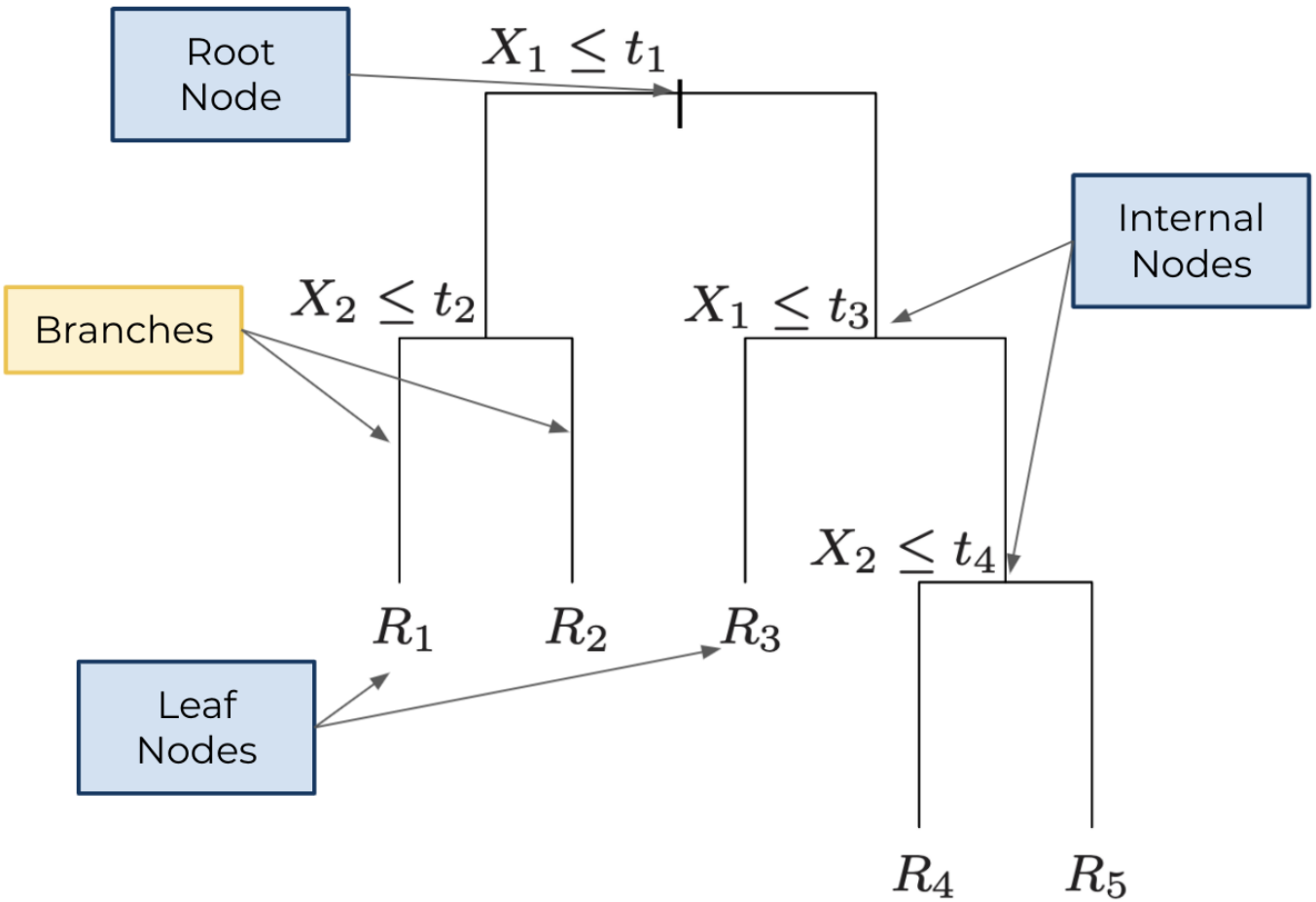
- assign each point to a **leaf** (region of feature space)
- according to a sequence of decisions based on features.




We can assign each region a number (regression) or category (classification) to make a prediction.

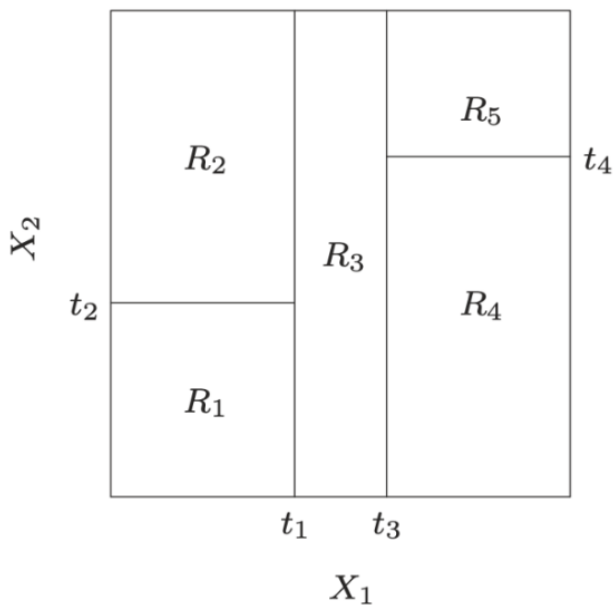
source: https://web.stanford.edu/~hastie/ElemStatLearn/

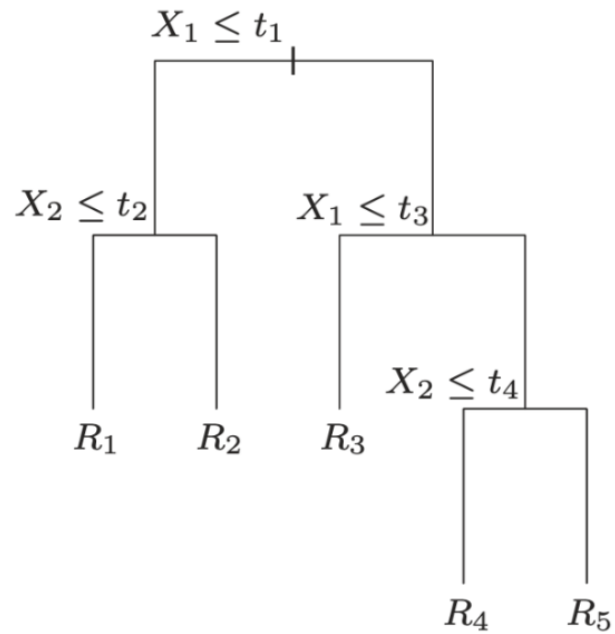
# Terminology



# Feature space partitioning

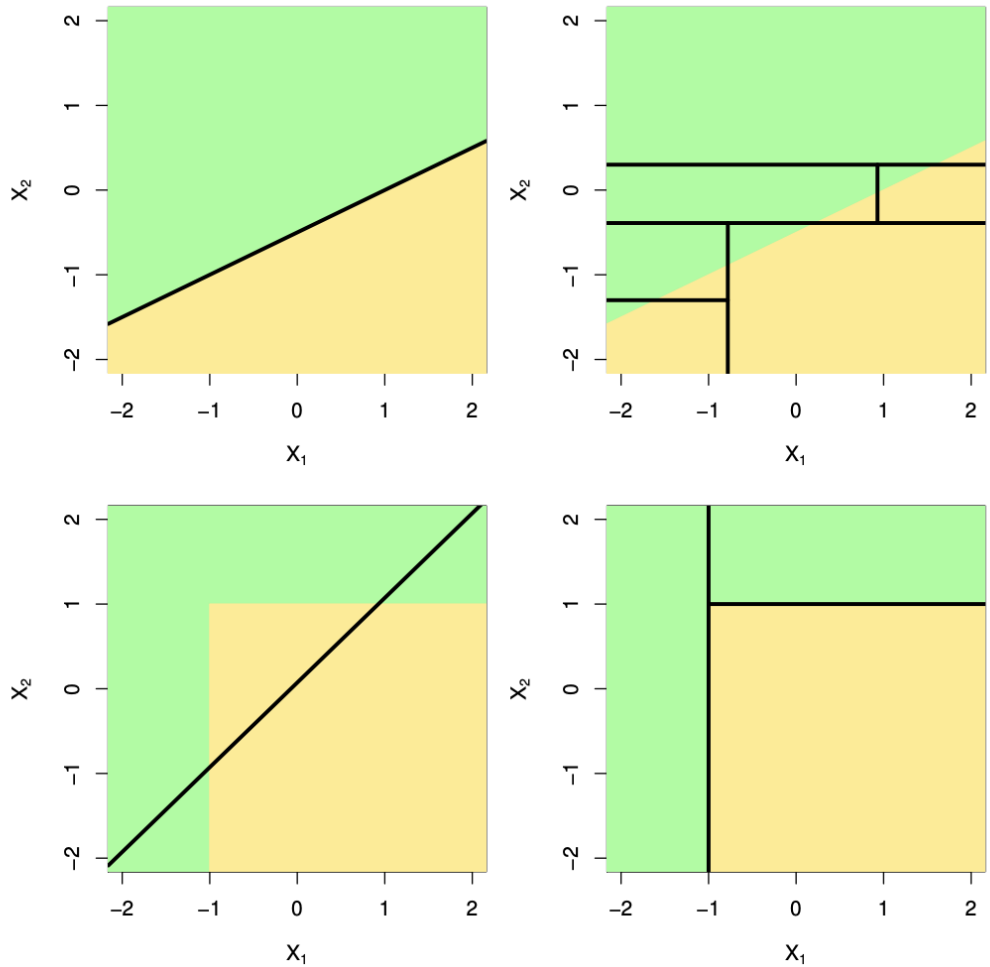
Each split in the decision trees considers a single feature, so every leaf is a rectangle. The leaves partition the feature space.



# Tress vs. linear classifiers

The best model depends on the data!




source: https://people.orie.cornell.edu/mru8/orie4741/lectures/

# Formalizing decision trees

- Partition the feature space into M regions $R_1, R_2, \ldots, R_M$.
- For each region $R_m$, assign a response $c_m \in \mathcal{Y}$.
- Our prediction is

$$\hat{f}(X) = \sum_{m=1}^{M} c_m \mathbb{1}(X \in R_m).$$

# Formalizing decision trees

- Partition the feature space into M regions $R_1, R_2, \ldots, R_M$.
- For each region $R_m$, assign a response $c_m \in \mathcal{Y}$.
- Our prediction is

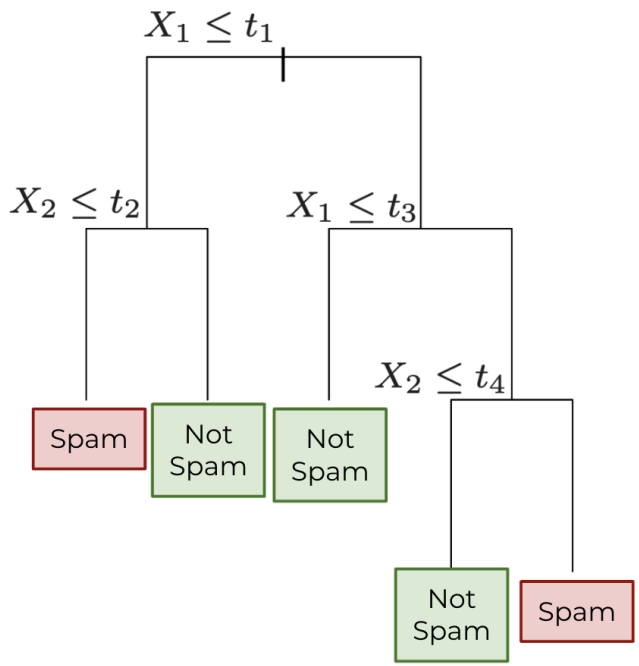
$$\hat{f}(X) = \sum_{m=1}^{M} c_m \mathbb{1}(X \in R_m).$$

What are good choices of $c_m$ for regression? Classification?

# Outline

# Classification trees

In classification, each $c_m$ corresponds to a label.




We seek the partition that minimizes classification error.

# Aside: optimal decision trees

We can formulate the problem of finding the best decision tree as a massive (and confusing!) integer programming (IP) problem.

$$
\begin{aligned}
&\text{minimize} && \ell_{\mathrm{c/r}}(\mathcal{T}, \hat{y}) + \lambda \ell^{\mathrm{d}}_{\mathrm{c/r}}(\mathcal{T}, \hat{y}) && && \text{(7a)}\\
&\text{subject to} && p \in \mathcal{P},\ \hat{y} \in \hat{\mathcal{Y}}(z) && && \text{(7b)}\\
& && q_\nu - \textstyle\sum_{j \in \mathcal{F}_{\mathrm{q}}} p_{\nu j} \mathbf{x}_{i,j} = g^+_{i\nu} - g^-_{i\nu} && \forall \nu,\ i && \text{(7c)}\\
& && g^+_{i\nu} \le M w^{\mathrm{q}}_{i\nu} && \forall \nu,\ i && \text{(7d)}\\
& && g^-_{i\nu} \le M(1 - w^{\mathrm{q}}_{i\nu}) && \forall \nu,\ i && \text{(7e)}\\
& && g^+_{i\nu} + g^-_{i\nu} \ge \epsilon(1 - w^{\mathrm{q}}_{i\nu}) && \forall \nu,\ i && \text{(7f)}\\
& && z_{il} \le 1 - w^{\mathrm{q}}_{i\nu} + (1 - \textstyle\sum_{j \in \mathcal{F}_{\mathrm{q}}} p_{\nu j}) && \forall \nu, i, l \in \mathcal{L}^{\mathrm{r}}(\nu) && \text{(7g)}\\
& && z_{il} \le w^{\mathrm{q}}_{i\nu} + (1 - \textstyle\sum_{j \in \mathcal{F}_{\mathrm{q}}} p_{\nu j}) && \forall \nu,\ i,\ l \in \mathcal{L}^{\mathrm{l}}(\nu) && \text{(7h)}\\
& && s_{\nu jk} \le p_{\nu j} && \forall \nu,\ j \in \mathcal{F}_{\mathrm{c}},\ k \in \mathcal{X}_j && \text{(7i)}\\
& && w^{\mathrm{c}}_{i\nu} = \textstyle\sum_{j \in \mathcal{F}_{\mathrm{c}}} \sum_{k \in \mathcal{X}_j} s_{\nu jk} \mathcal{I}\left(\mathbf{x}_{i,j} = k\right) && \forall \nu,\ i && \text{(7j)}\\
& && z_{il} \le w^{\mathrm{c}}_{i\nu} + (1 - \textstyle\sum_{j \in \mathcal{F}_{\mathrm{c}}} p_{\nu j}) && \forall \nu,\ i,\ l \in \mathcal{L}^{\mathrm{l}}(\nu) && \text{(7k)}\\
& && z_{il} \le 1 - w^{\mathrm{c}}_{i\nu} + (1 - \textstyle\sum_{j \in \mathcal{F}_{\mathrm{c}}} p_{\nu j}) && \forall \nu, i, l \in \mathcal{L}^{\mathrm{r}}(\nu) && \text{(7l)}\\
& && \textstyle\sum_{l \in \mathcal{L}} z_{il} = 1 && \forall i && \text{(7m)}
\end{aligned}
$$

But in practice, we can't solve the problem for even medium datasets. (Though researchers are making progress!)

# The best split

Instead, we approximate the best partition using a greedy algorithm. For each internal node,

- consider all possible **features** to split on
- and **thresholds** to split at
- and choose the best split.

# The best split

Instead, we approximate the best partition using a greedy algorithm. For each internal node,

- consider all possible **features** to split on
- and **thresholds** to split at
- and choose the best split.

What should **best** mean?

A. make each leaf node as homogeneous as possible

B. make each leaf node as similar to other nodes as possible

# The best split

Instead, we approximate the best partition using a greedy algorithm. For each internal node,

- consider all possible **features** to split on
- and **thresholds** to split at
- and choose the best split.

What should **best** mean?

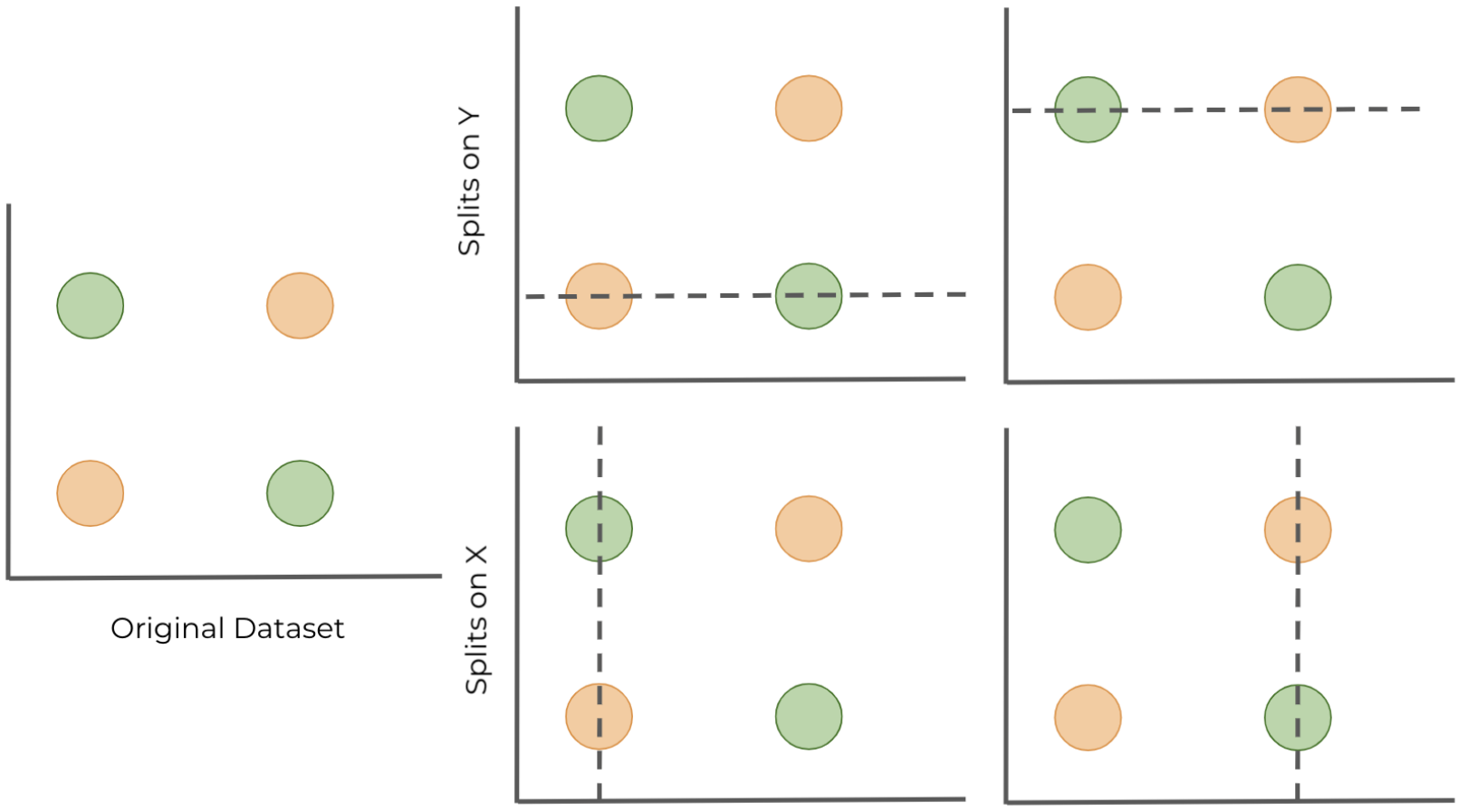
A. make each leaf node as homogeneous as possible

B. make each leaf node as similar to other nodes as possible

**Definition**

A node is **pure** if every point in the node has the same label.

# Choosing the best split



# How many possible splits?

Given a feature, how many possible thresholds must we consider to find the best?

A. infinitely many

B. $n$

C. $d$

D. $n - 1$

E. $\log n$

Checking all dimensions + all thresholds takes $O(dn \log(n))$ splits

# How to measure purity

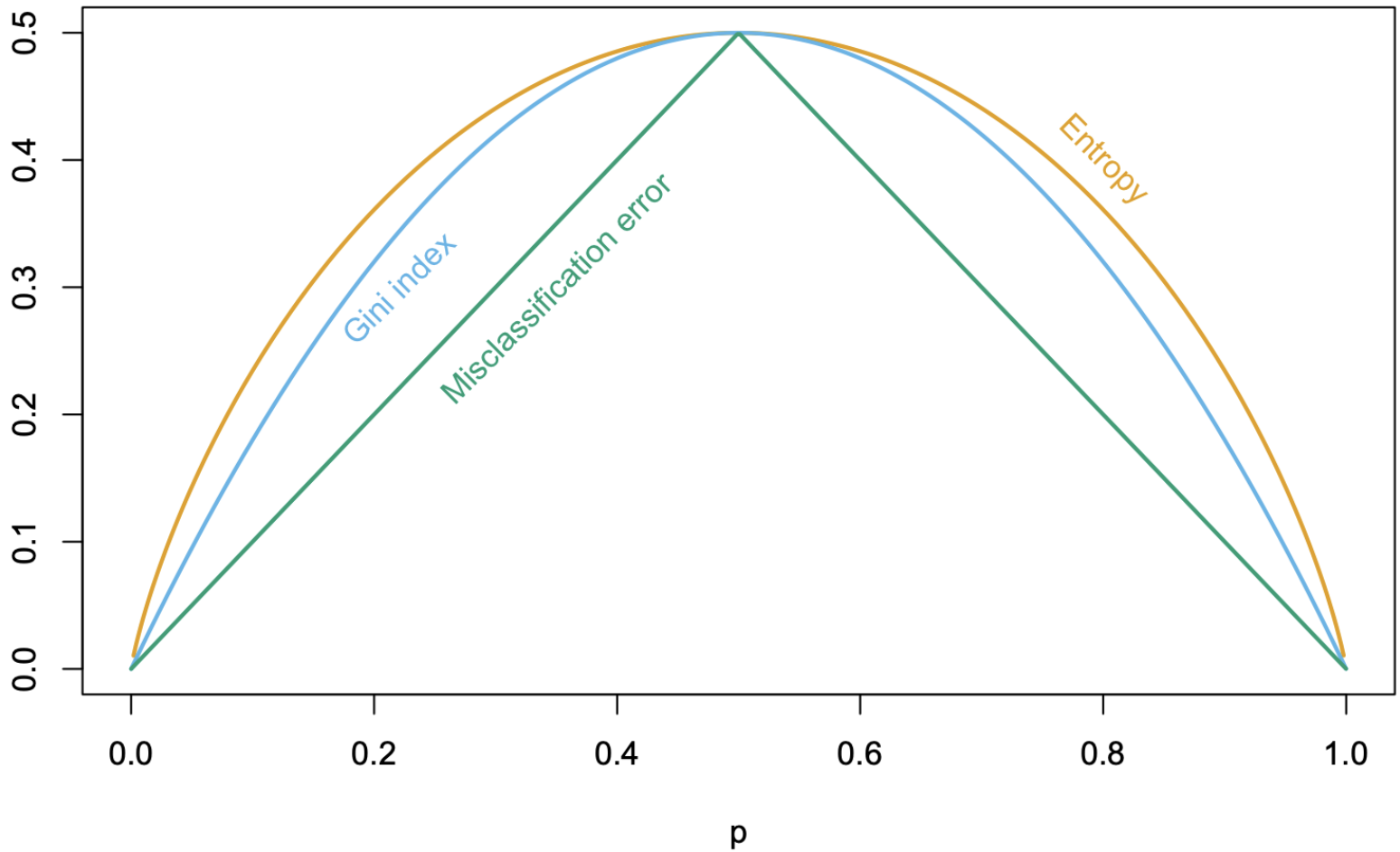
Consider a classification task with dataset $\{(x_1, y_1), \ldots, (x_n, y_n)\}$. Define

- the number of points in region $m$: $N_m = |\{i : x_i \in R_m\}|$
- fraction of points in region $m$ with label 1: $\hat{p}_m = \frac{1}{N_m} \sum_{x_i \in R_m} \mathbb{1}(y_i = 1)$.

We can measure impurity via

- **Misclassification Error:** $\frac{1}{N_m} \sum_{i \in R_m} \mathbb{1}(y_i \neq c_m)$
- **Gini Index:** $\hat{p}_m(1 - \hat{p}_m)$
- **Cross-entropy:** $-\hat{p}_m \log(\hat{p}_m) - (1 - \hat{p}_m) \log(1 - \hat{p}_m)$

# Purity functions

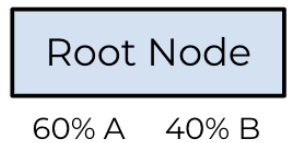



source: https://web.stanford.edu/~hastie/ElemStatLearn/

# Recursive partitioning

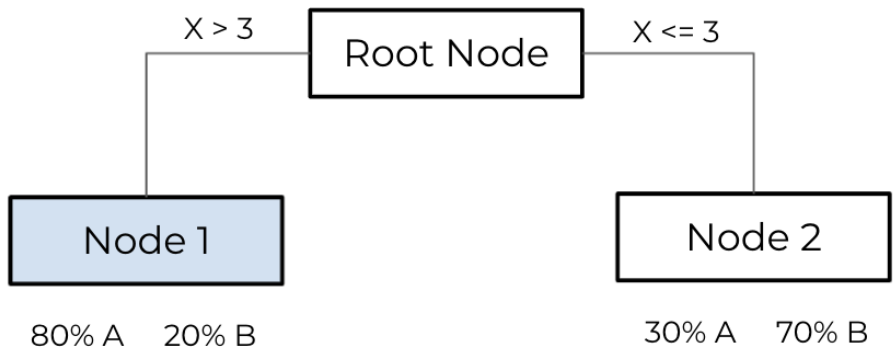
Step 1: Start at root node. Choose split to maximize purity function.

---



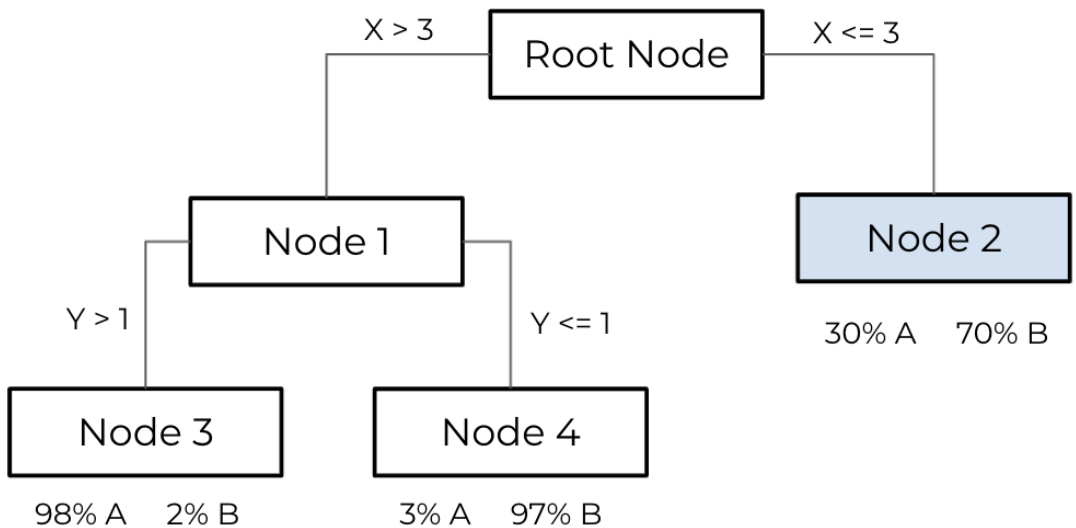
# Recursive partitioning

Step 2: Pick new node. Choose split to maximize purity function.



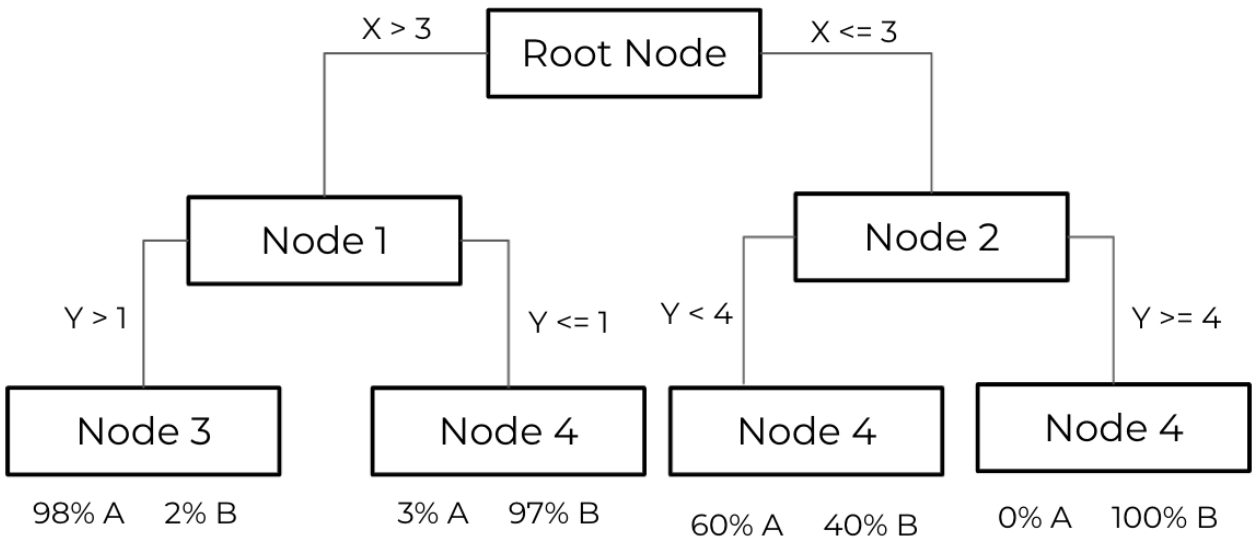
# Recursive partitioning

Step 3: Process next node...



# Recursive partitioning

Step 4: Stop?

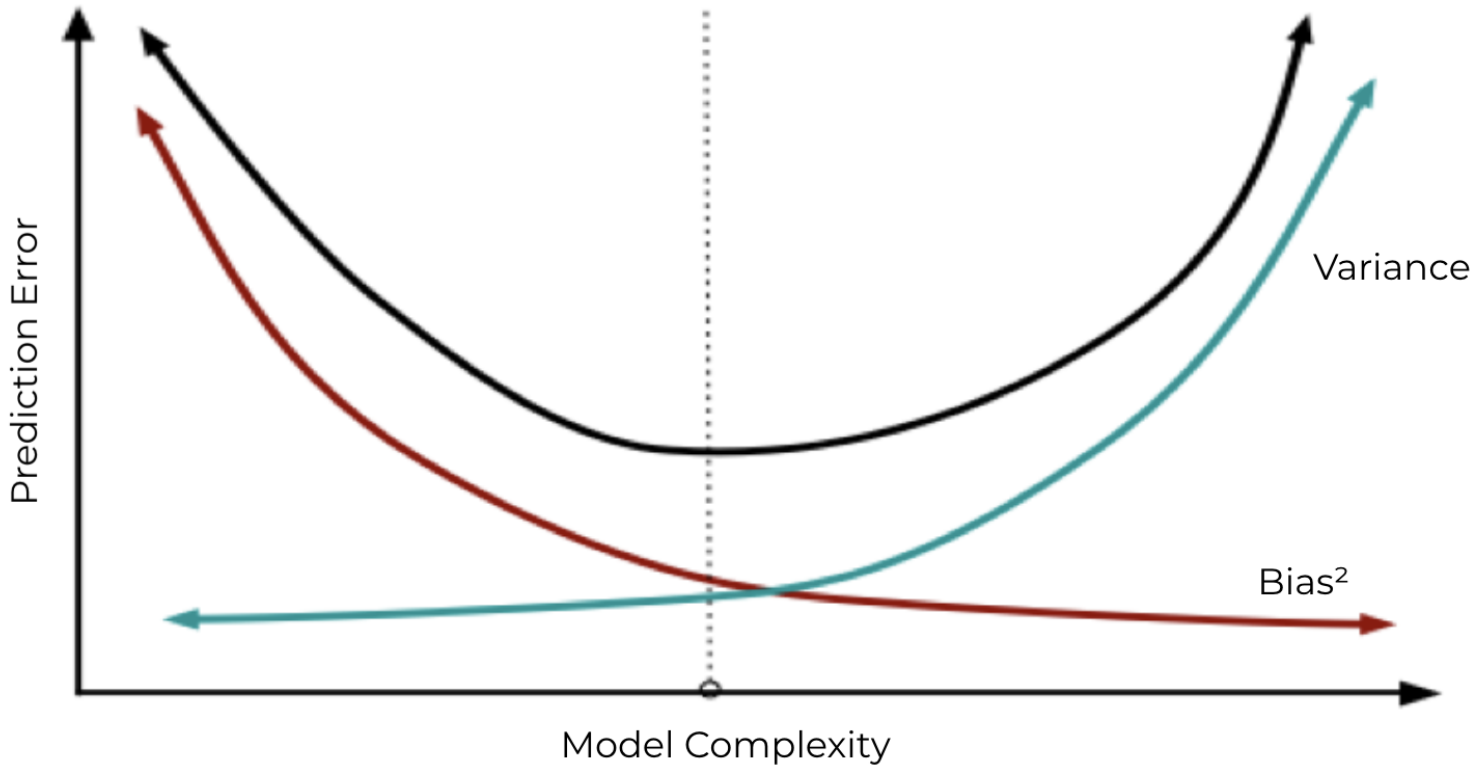


# Poll

When should we stop the recursive partitioning?

A. After 1 split
B. When each node has 1 data point
C. When each node is pure
D. Something else...

# Recall: bias-variance trade-off




source: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html

# Poll

Where would a deep tree (with lots of leaves) fall on the bias-variance trade-off?

A. High bias, low variance

B. Just right

C. Low bias, high variance

# Poll

Where would a shallow tree (with few leaves) fall on the bias-variance trade-off?

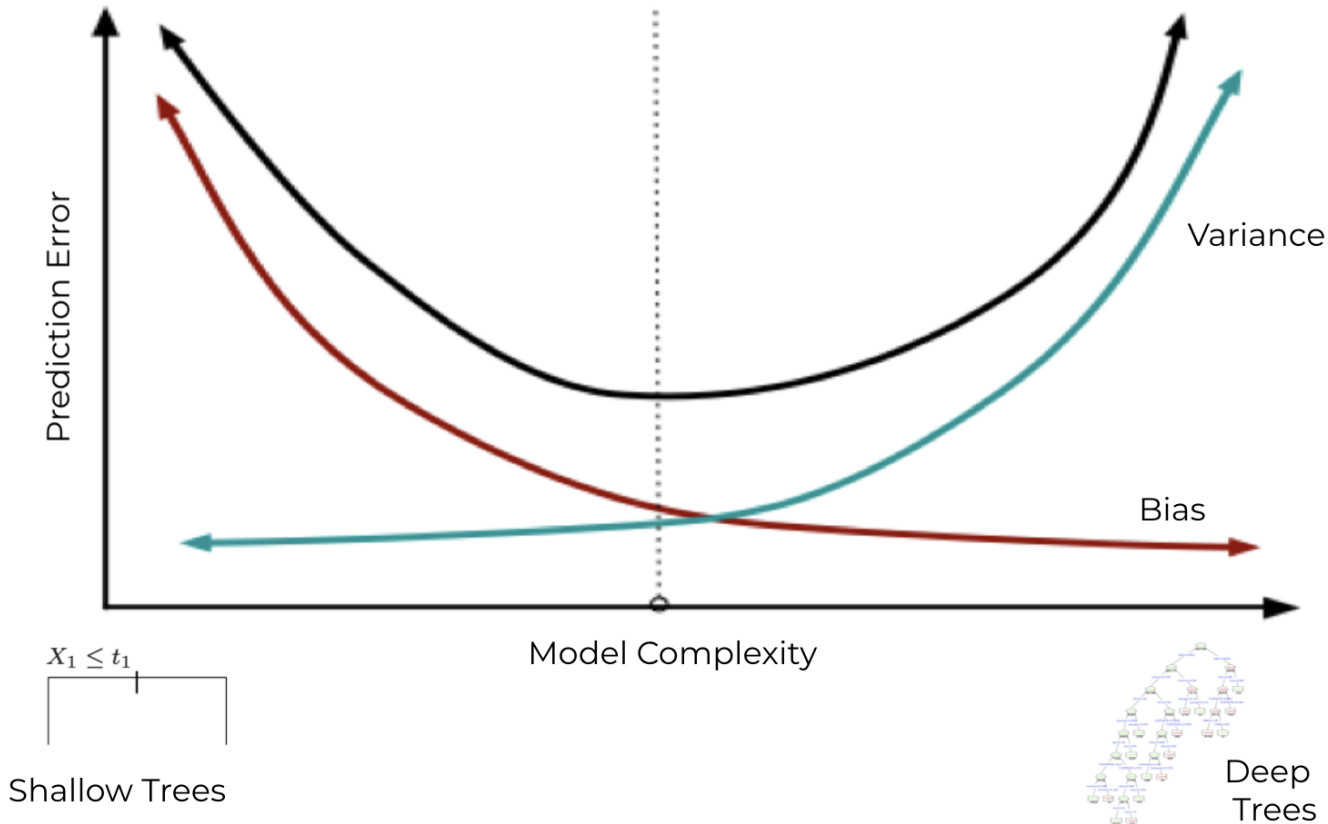
A. High bias, low variance

B. Just right

C. Low bias, high variance

# Trees and bias-variance trade-off

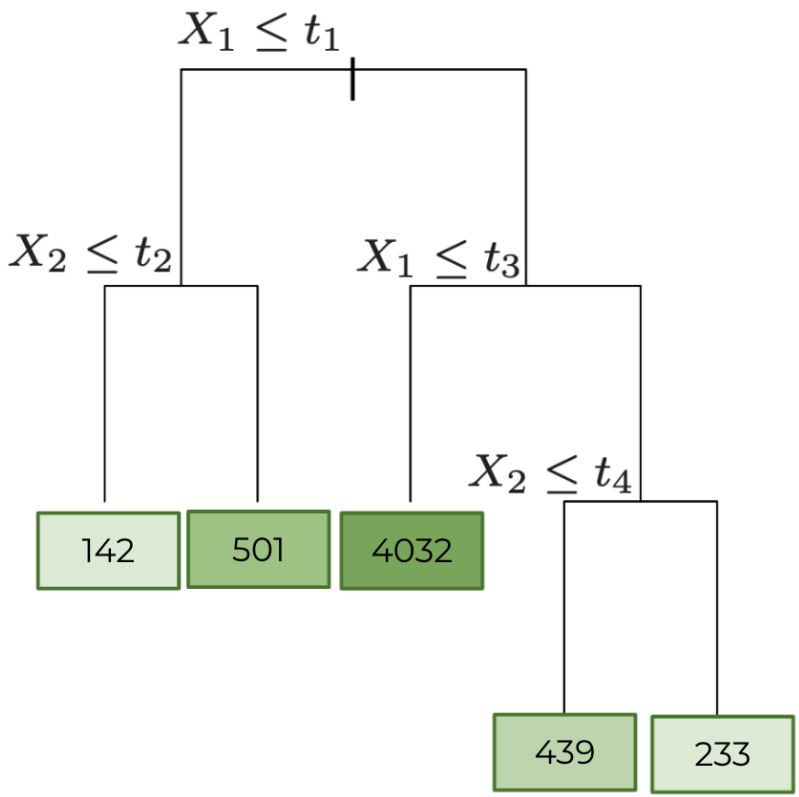
Deep trees are more complex.



# Outline

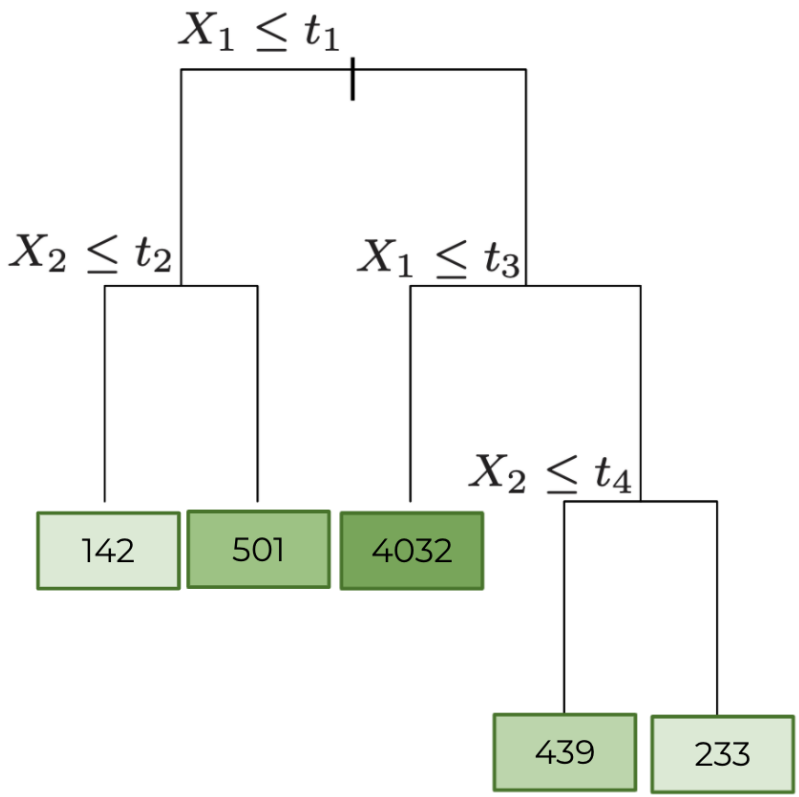
# Regression trees

To solve a regression problem (predicting e.g. house price, income, ...), use numeric $c_m$!



# Regression trees

To solve a regression problem (predicting e.g. house price, income, ...), use numeric $c_m$!




How to choose $c_m$?

# Constructing a regression tree

Given a partition $R_1, \ldots, R_M$ and $c_m$, on sample $x \in \mathbf{R}^d$, our tree predicts

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}(x \in R_m).$$

Given $R_m$ and output $y_i$ for $x_i \in R_m$, what $c_m$ minimizes

$$\sum_{i:x_i \in R_m} (y_i - f(x_i))^2?$$

# Constructing a regression tree

Given a partition $R_1, \ldots, R_M$ and $c_m$, on sample $x \in \mathbf{R}^d$, our tree predicts

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}(x \in R_m).$$

Given $R_m$ and output $y_i$ for $x_i \in R_m$, what $c_m$ minimizes

$$\sum_{i:x_i \in R_m} (y_i - f(x_i))^2?$$

A. mean of $y_i$ for $x_i \in R_m$
B. median of $y_i$ for $x_i \in R_m$
C. mode of $y_i$ for $x_i \in R_m$
D. minimum of $y_i$ for $x_i \in R_m$

# Constructing a regression tree

Given a partition $R_1, \ldots, R_M$ and $c_m$, on sample $x \in \mathbf{R}^d$, our tree predicts

$$f(x) = \sum_{m=1}^{M} c_m \mathbb{1}(x \in R_m).$$

Given $R_m$ and output $y_i$ for $x_i \in R_m$, what $c_m$ minimizes

$$\sum_{i:x_i \in R_m} (y_i - f(x_i))^2?$$

A. mean of $y_i$ for $x_i \in R_m$
B. median of $y_i$ for $x_i \in R_m$
C. mode of $y_i$ for $x_i \in R_m$
D. minimum of $y_i$ for $x_i \in R_m$

$$\hat{c}_m = \text{mean}(y_i \mid x_i \in R_m)$$

# Constructing a regression tree

search over possible splits on variable $j$ with threshold $s$:

$$R_1(j,s) = \{X | X_j \leq s\}, \qquad R_2(j,s) = \{X | X_j > s.\}$$

Our goal is find the coordinate $j$, threshold $s$, and values $c_1$ and $c_2$ to solve

$$\min_{j,s} \left[ \min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_1(j,s)} (y_i - c_2)^2 \right].$$

# Constructing a regression tree

search over possible splits on variable $j$ with threshold $s$:

$$R_1(j,s) = \{X | X_j \leq s\}, \qquad R_2(j,s) = \{X | X_j > s.\}$$

Our goal is find the coordinate $j$, threshold $s$, and values $c_1$ and $c_2$ to solve

$$\min_{j,s} \left[ \min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_1(j,s)} (y_i - c_2)^2 \right].$$

- Inner minimization can be solved by computing the average value for each region
- Outer minimization can be solved by checking every coordinate $j$ and threshold $s$ ($O(dn)$ checks)

# Demo

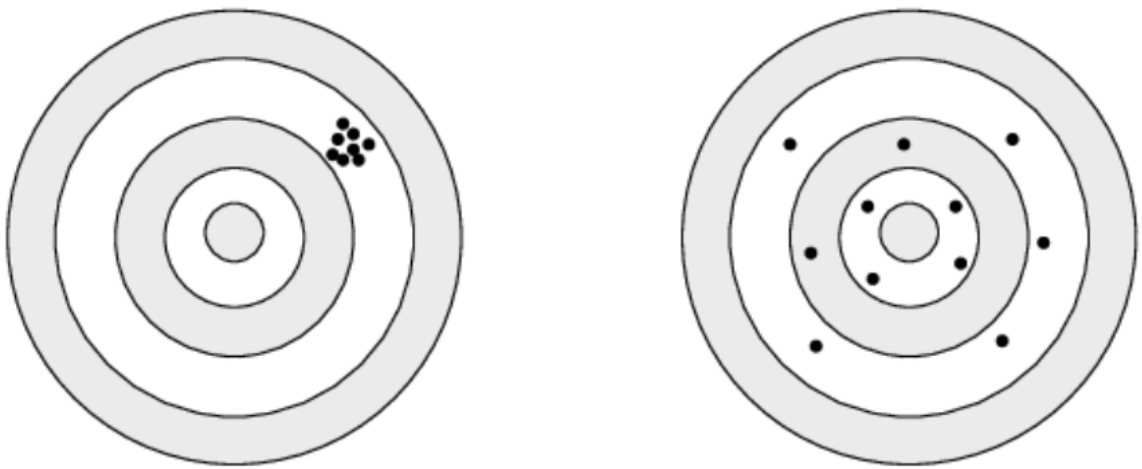
https://github.com/ORIE4741/demos/trees.ipynb

# Outline

# Ensemble methods

So far we've seen how to use trees to get

- low bias, high variance models or
- high bias, low variance models



can we do better?

source: `https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote12.html`

# Properties of an effective ensemble

**Definition**

An **ensemble** model aggregates many simple models (called **weak learners**) together to create a more powerful model with lower variance.

The ensemble strategy succeeds when

# Properties of an effective ensemble

**Definition**

An **ensemble** model aggregates many simple models (called **weak learners**) together to create a more powerful model with lower variance.

The ensemble strategy succeeds when

- each weak learner is better than random
- each weak learner makes different kinds of errors

Different ensemble methods use different weak learners (and aggretate in different ways). Examples: bagging, random forest, boosting, ...

# Outline

# Bagging

Bagging (**Bootstrap AGgretation**) generates weak learners by training on different bootstrap samples of our training data.

- Draw $m$ bootstrap samples from training data
- Train a model on each sample
- Aggregate the resulting predictions together:
  - regression: average the outputs
  - classification: majority vote

# Variance of bagging

Consider an average of $B$ models. What's the variance?

- Each model is identically distributed with variance $\sigma^2$.
- The models are not necessarily independent: they have pairwise correlation $\rho$.
- The variance of the average of the models is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

# Variance of bagging

Consider an average of $B$ models. What's the variance?

- Each model is identically distributed with variance $\sigma^2$.
- The models are not necessarily independent: they have pairwise correlation $\rho$.
- The variance of the average of the models is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

How can we reduce the variance of our averaged model predictions?

A. Increase $B$
B. Decrease $B$
C. Increase $\rho$
D. Decrease $\rho$

# Variance of bagging

Consider an average of $B$ models. What's the variance?

- Each model is identically distributed with variance $\sigma^2$.
- The models are not necessarily independent: they have pairwise correlation $\rho$.
- The variance of the average of the models is

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

How can we reduce the variance of our averaged model predictions?

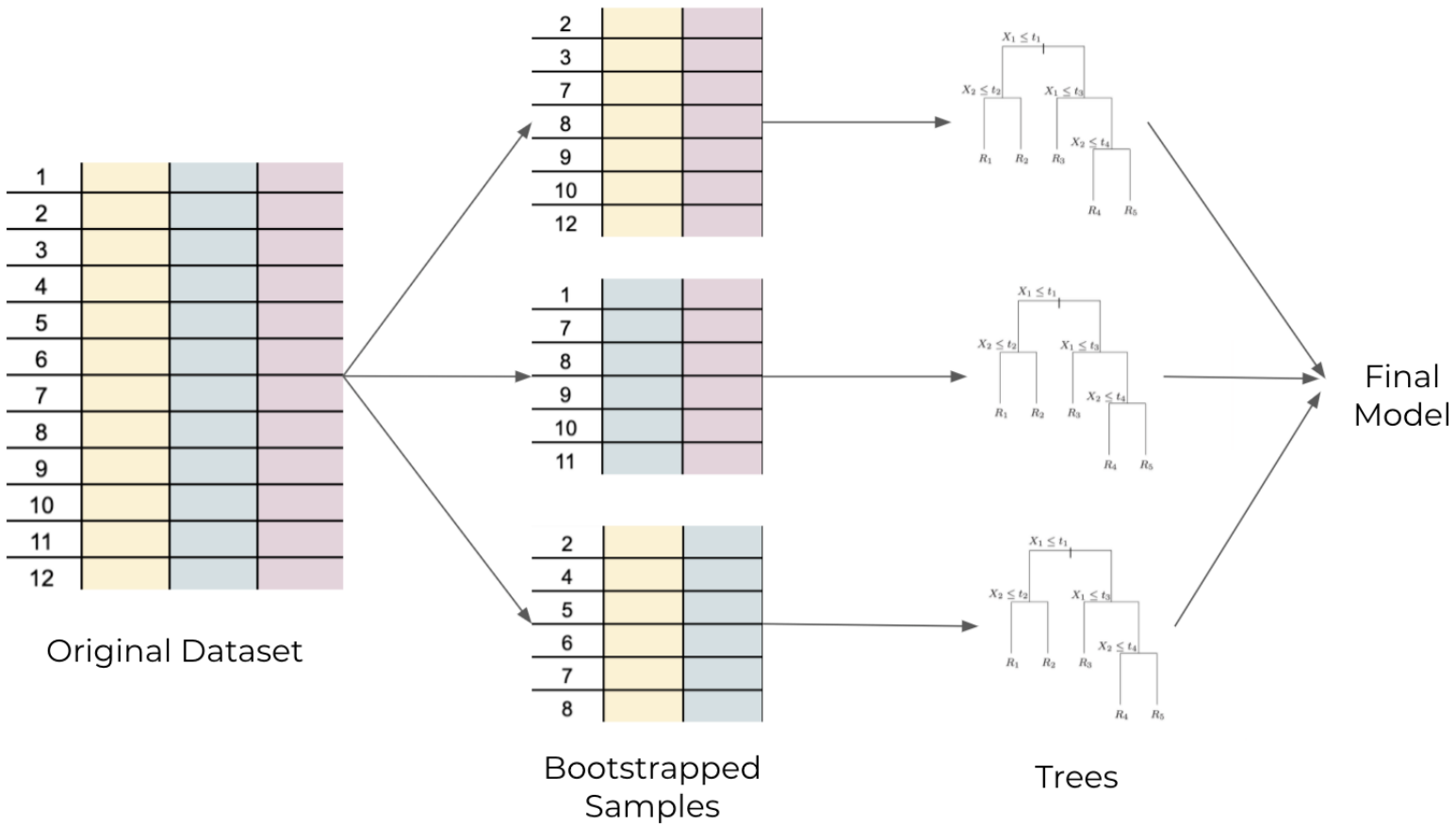
A. Increase $B$
B. Decrease $B$
C. Increase $\rho$
D. Decrease $\rho$

Increasing $B$ is easy! How could we change $\rho$?

# Random forests

**key idea:**

- bagged ensemble
- of trees, each limited to a random subset of features



# Random forests algorithm

**Algorithm 15.1** *Random Forest for Regression or Classification.*

1. For $b = 1$ to $B$:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size $N$ from the training data.
   (b) Grow a random-forest tree $T_b$ to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $n_{min}$ is reached.
      i. Select $m$ variables at random from the $p$ variables.
      ii. Pick the best variable/split-point among the $m$.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point $x$:

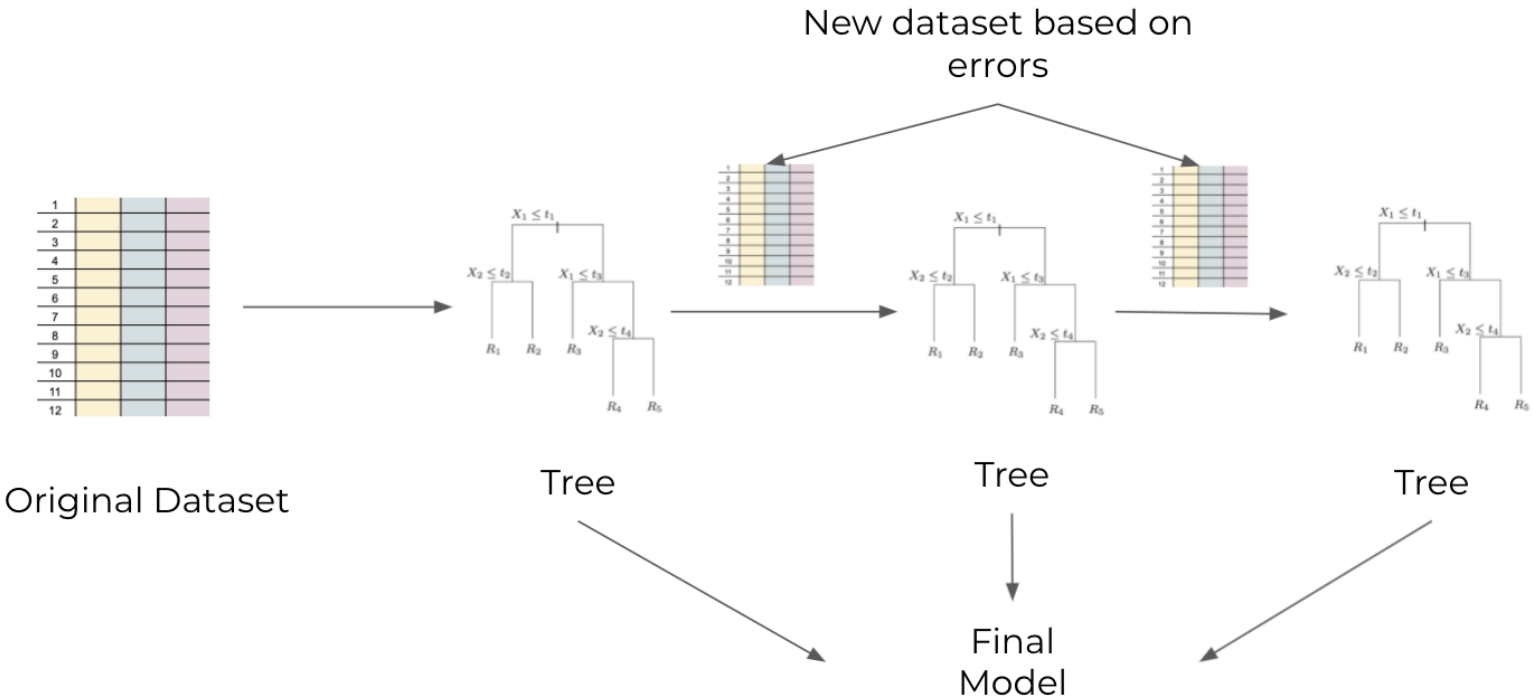
*Regression:* $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^B T_b(x)$.

*Classification:* Let $\hat{C}_b(x)$ be the class prediction of the $b$th random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

# Outline

# Boosting

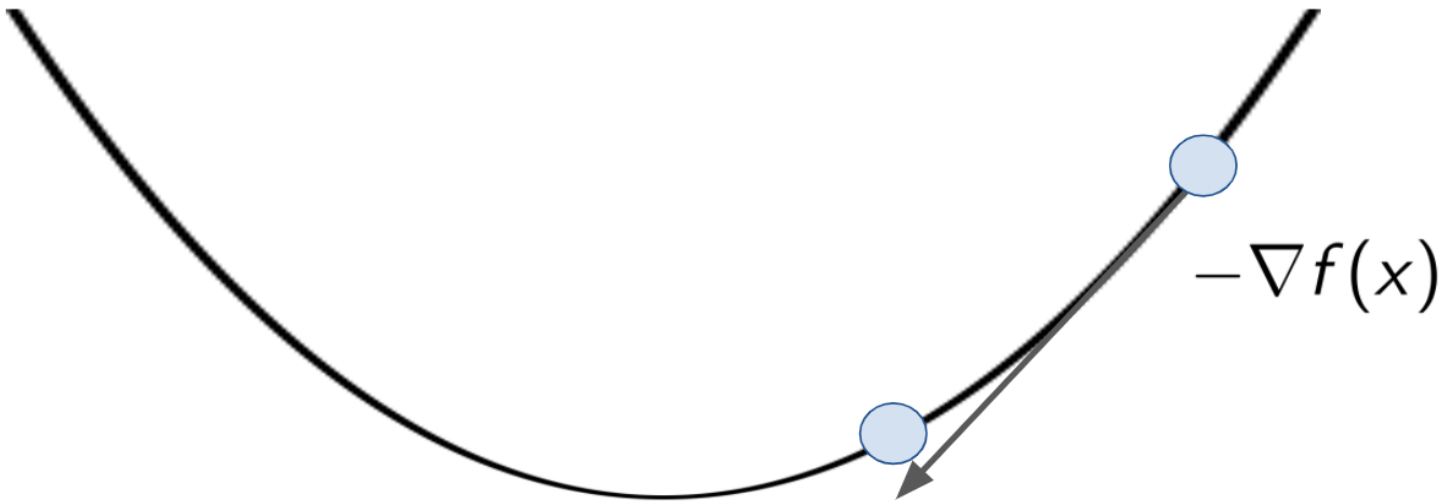
An alternative approach is to train weak learners sequentially to improve predictions.




How should we train each subsequent tree?

# Recall: gradient descent

Gradient descent moves the current iterate in the direction of fastest decrease. It determines the step size using line search.



# Gradient boosting

**key idea:** train weak learners to approximate the gradient.

- Let $L(y_i, \hat{y}_i)$ be our loss function of choice.
- Let $F_m(x)$ be our predictor after m iterations.
- Our goal is to replicate the gradient update

$$F_m(x) = F_{m-1}(x) + \gamma \sum_{i=1}^{n} \nabla_{F_{m-1}} L(y_i, F_{m-1}(x_i)),$$

  where $\gamma$ is a step size determined through line search.

How could we approximate the gradient with a weak learner?

# Gradient boosting

Construct a training set to predict the gradient:

- Compute gradient term for each data point:

$$r_{im} = -\Big[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\Big]_{F(x)=F_{m-1}(x)}.$$

- Fit a weak learner $h_m$ on the dataset $\{x_i, r_{im}\}_{i=1}^n$.
- Compute the step direction

$$\gamma_m = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)).$$

- Update model: $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x_i)$.

# Gradient boosting trees

Boosting for trees is even simpler (and faster)!

- Recall: Output of tree is a set of partitions $R_1, \ldots, R_M$ each with a response $c_m$
- Our boosting function now looks like:

$$F_m(x) = F_{m-1}(x) + \gamma_m \sum_{i=1}^{M} c_i \mathbb{1}(x \in R_i)$$

- Instead of learning $\gamma$ for the tree as a whole, we can combine $\gamma$ and $c$ and learn them together for each partition:

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_m} L(y_i, F_{m-1}(x_i) + \gamma)$$

# Gradient boosting trees algorithm

**Algorithm 10.3** *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to $M$:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets $r_{im}$ giving terminal regions $R_{jm}$, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

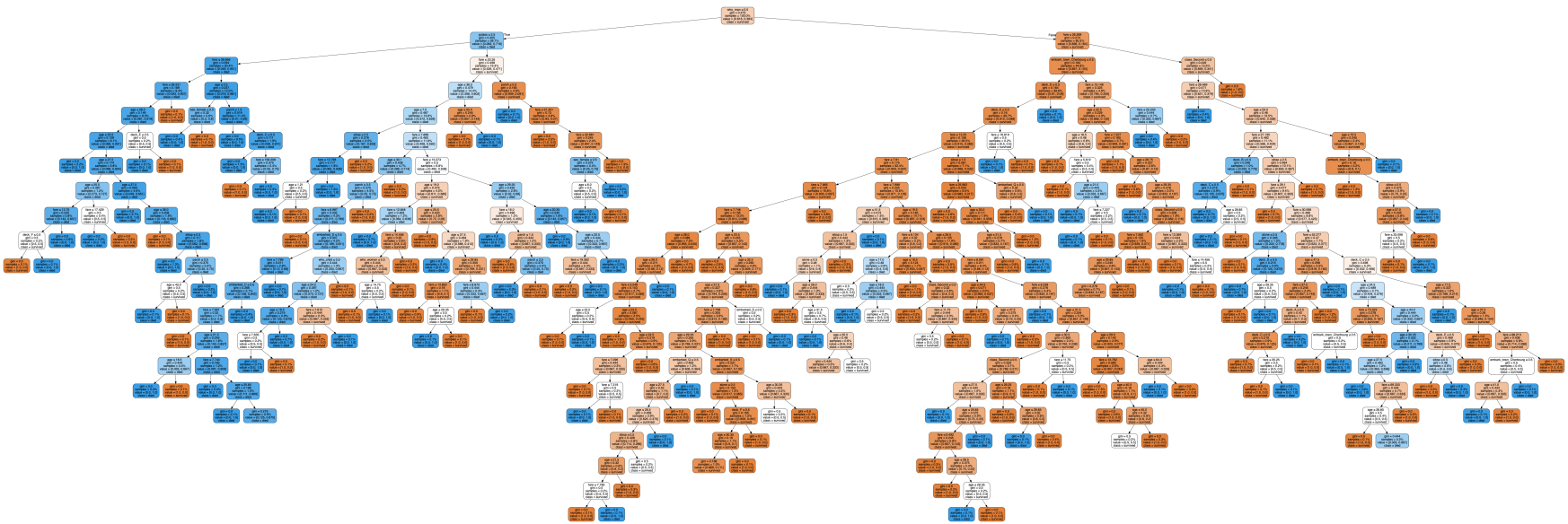
# Demo

https://github.com/ORIE4741/demos/ensembles.ipynb

# Outline

# Interpreting features in a decision tree

Decision trees are a popular tool in interpretable ML because we can follow a decision through from root to leaf node. . .



But in a deep tree, or an ensemble, how can we tell which features are important?

# Mean decrease impurity

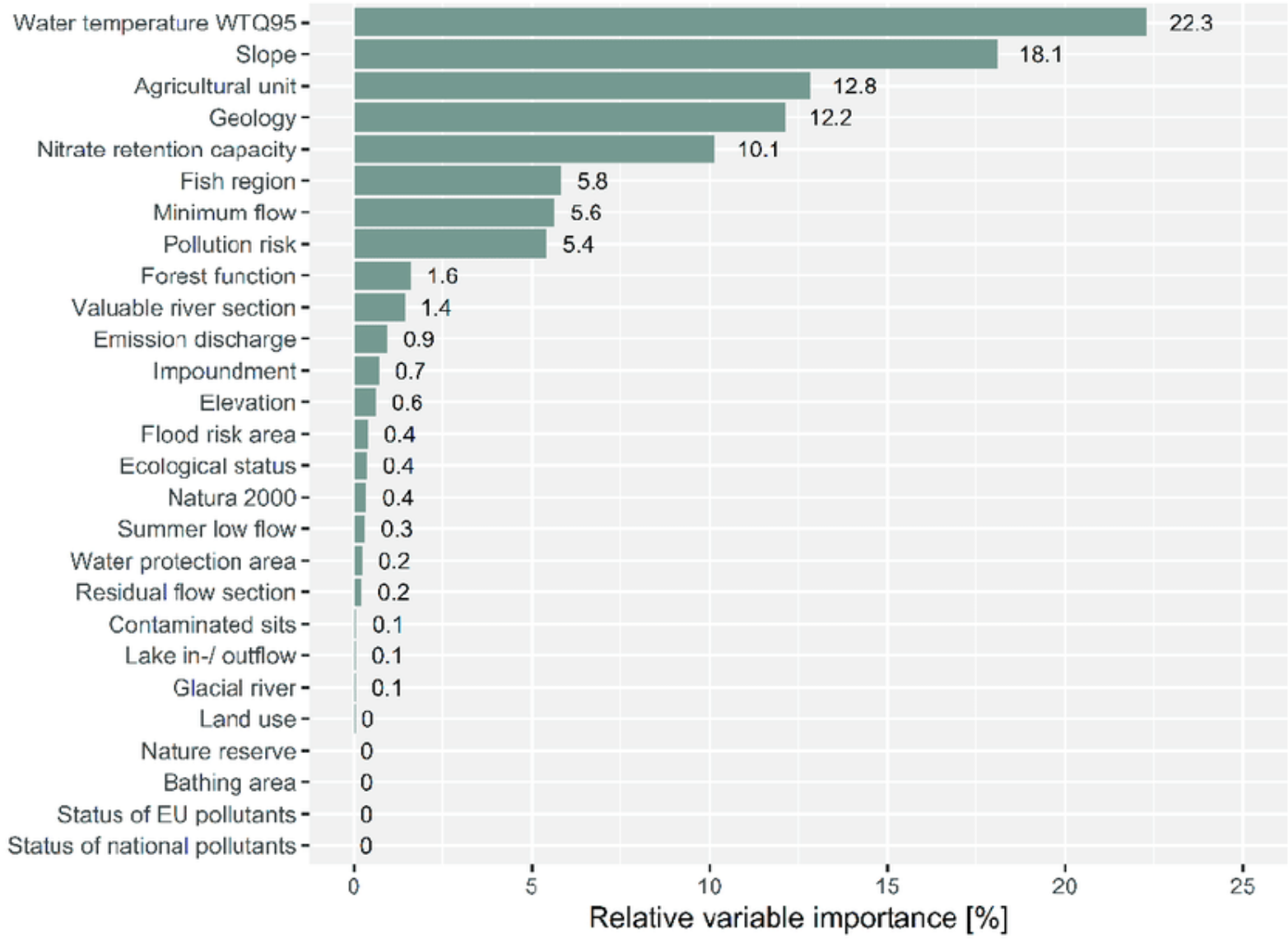
MDI says a variable is important if splits using it lead to a big decrease in node purity (i.e. Gini).

- Let $I_P, I_L, I_R$ be the impurity at the parent, left child, and right child nodes of a split respectively.
- Let $p_l, p_r$ be the proportion of data that goes to the left and right child node respectively.
- The decrease in purity for a split is:

$$I_P - p_l I_L - p_r I_r$$

- To get the mean decrease impurity, we can average the decrease over every split in the tree or forest using that variable.

# Visualizing feature importance




source: https://www.researchgate.net/figure/