Trees

Trees

· adaptive

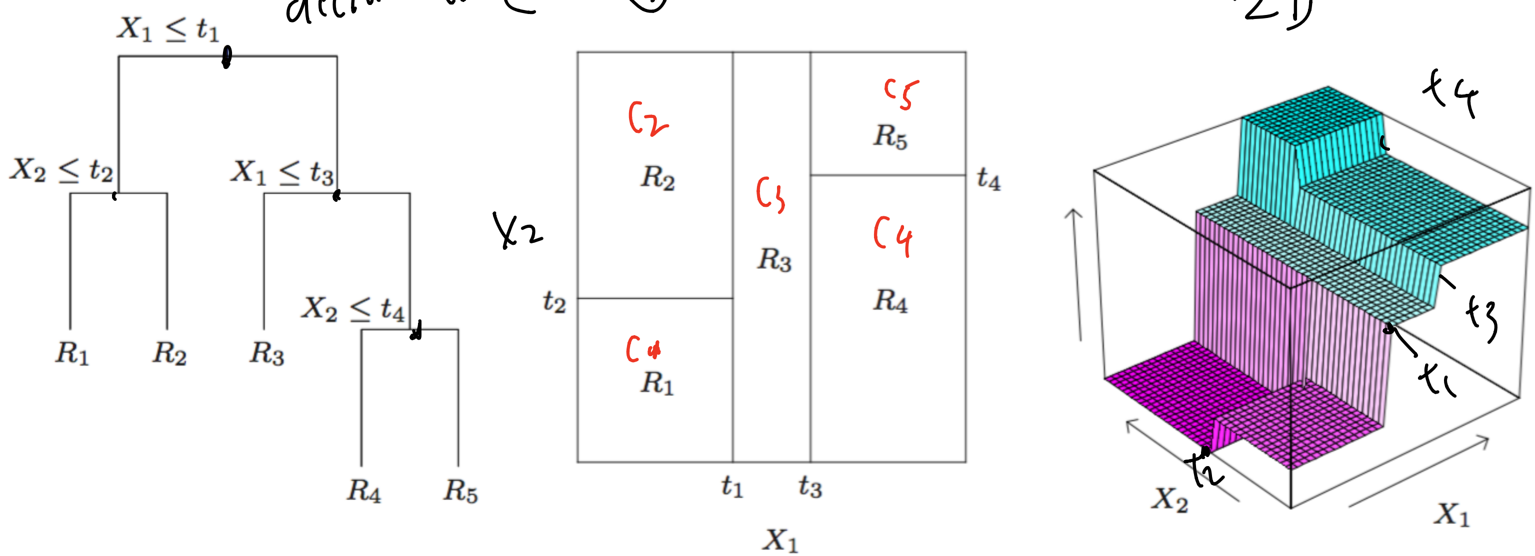
indicator

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

Build a binary tree, splitting along axes

decision tree

2D



Learning decision trees

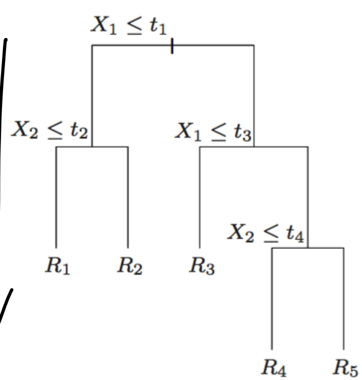
$\{(X_i, y_i)\}_{i=1}^N$, $y_i \in \{1,-1\}$

$\{(X_i, y_i)\}_{i=1}^N$, $y_i \in \{+,-\}$

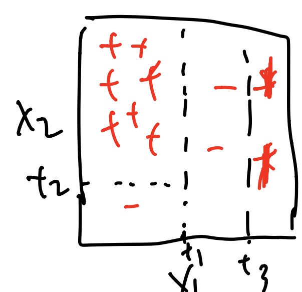
X_i: (1) real-valued vector (2) categorical

> Start from empty decision tree
> Split on next best attribute (feature) $\boxed{\text{purity}}$
  - Use, for example, information gain to select attribute
  - Split on $\arg\max_{i \in \{1,\dots,p\}} IG(X_i) = \arg\max_i \underbrace{H(Y)}_{\text{entropy}} - \underbrace{H(Y \mid X_i)}_{\text{conditional entropy}}$
> Recurse
> Prune

hyper-parameter depth

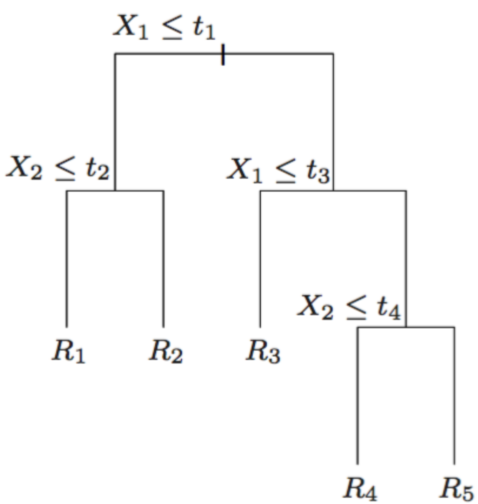


$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m).$$



Trees

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



- Trees
  - **have low bias, high variance**
  - deal with categorial variables well

    [Seattle, SF]
  - intuitive, interpretable
  - good software exists
  - Some theoretical guarantees

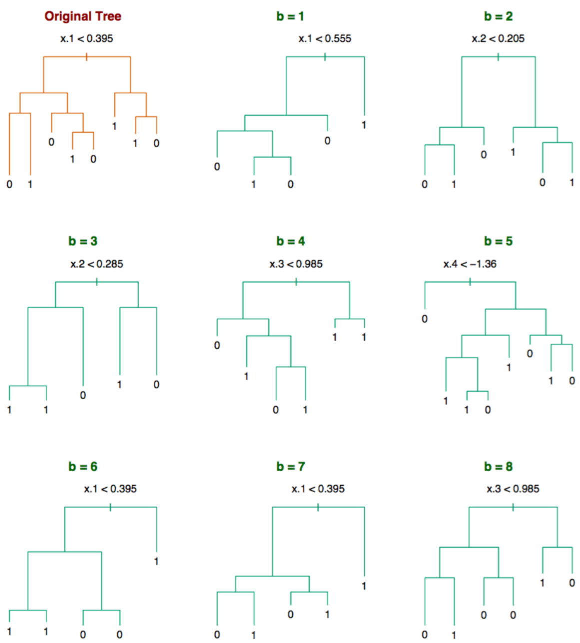
Random Forests

Random Forests

Tree methods have **low bias** but **high variance**.

One way to reduce variance is to construct a lot of “lightly correlated” trees and average them:

general idea



“Bagging:” Bootstrap aggregating

Random Forests

B: total # of trees

$\{(x_i, y_i)\}_{i=1}^{N}$

$x_i \in \mathbb{R}^p$

Algorithm 15.1 *Random Forest for Regression or Classification.*

1. For $b = 1$ to B:

   with replacement

   bagging

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   randomness

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

m~p/3

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

m~sqrt(p)

Random Forests

- Random Forests
  - **have low bias, low variance**
  - deal with categorial variables well
  - not that intuitive or interpretable
  - Notion of confidence estimates
  - good software exists
  - Some theoretical guarantees
  - **works well with default hyperparameters**

Boosting and Additive Models

Boosting

learning theory question

- 1988 Kearns and Valiant: "Can **weak learners** be combined to create a **strong learner?**"

Weak learner definition (informal):

An algorithm $\mathcal{A}$ is a *weak learner* for a hypothesis class $\mathcal{H}$ that maps $\mathcal{X}$ to $\{-1, 1\}$ if for all input distributions over $\mathcal{X}$ and $h \in \mathcal{H}$, we have that $\mathcal{A}$ correctly classifies h with error at most $1/2 - \gamma$ strong ε error , $\varepsilon \to 0$

- 1990 Robert Schapire: "Yup!"
- 1995 Schapire and Freund: "Practical for 0/1 loss" AdaBoost
- 2001 Friedman: "Practical for arbitrary losses"
- 2014 Tianqi Chen: "Scale it up!" XGBoost

Additive models

- Given: $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$
- Generate **random** functions: $\phi_t : \mathbb{R}^d \to \mathbb{R} \quad t = 1, \ldots, p$
- Learn some weights: $\widehat{w} = \arg\min_w \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$
- Classify new data: $f(x) = \text{sign}\left(\sum_{t=1}^p \widehat{w}_t \phi_t(x)\right)$

Additive models

- Given: $\{(x_i, y_i)\}_{i=1}^n \quad x_i \in \mathbb{R}^d, y_i \in \{-1, 1\}$
- Generate **~~random~~** functions: $\phi_t : \mathbb{R}^d \to \mathbb{R} \quad t = 1, \dots, p$
- Learn some weights: $\widehat{w} = \arg\min_{w} \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$
- Classify new data: $f(x) = \text{sign}\left(\sum_{t=1}^p \widehat{w}_t \phi_t(x)\right)$

$\phi_t \in$ classifier class

An interpretation:
Each $\phi_t(x)$ is a classification rule that we are assigning some weight $\widehat{w}_t$

$$\widehat{w}, \widehat{\phi}_1, \dots, \widehat{\phi}_t = \arg \min_{w, \phi_1, \dots, \phi_p} \sum_{i=1}^n \text{Loss}\left(y_i, \sum_{t=1}^p w_t \phi_t(x_i)\right)$$

is in general computationally hard

Forward Stagewise Additive models

$f_t(x)$

$b(x, \gamma)$ is a function with parameters γ

Examples: $b(x, \gamma) = \dfrac{1}{1 + e^{-\gamma^T x}}$

$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:

   (a) Compute

   fixed

   $$(\beta_m, \gamma_m) = \arg\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, \underline{f_{m-1}(x_i)} + \underline{\beta} \underline{b(x_i; \gamma)}).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

$$f_M(x) = \sum_{m=1}^{M} \beta_m \cdot b(x, \gamma_m)$$

Idea: greedily add one function at a time

Forward Stagewise Additive models

$b(x, \gamma)$ is a function with parameters γ

Examples: $b(x, \gamma) = \dfrac{1}{1 + e^{-\gamma^T x}}$

$b(\cancel{x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2}\}$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:

   (a) Compute

   $$(\beta_m, \gamma_m) = \arg\min_{\beta, \gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$

   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

$(\beta_1, \gamma_1), (\beta_2, \gamma_2) \ldots$

Idea: greedily add one function at a time

AdaBoost: $b(x, \gamma)$: classifiers to $\{-1, 1\}$

$L(y, f(x)) = \exp(-yf(x))$ exponential loss

$f(x) = \sum \beta_m b(x; \gamma_m)$

Forward Stagewise Additive models

$b(x, \gamma)$ is a function with parameters γ

Examples: $b(x, \gamma) = \dfrac{1}{1 + e^{-\gamma^T x}}$

$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

Idea: greedily add one function at a time

Boosted Regression Trees:

$$L(y, f(x)) = (y - f(x))^2$$

$b(x, \gamma)$: regression trees

Forward Stagewise Additive models

$b(x, \gamma)$ is a function with parameters γ

Examples: $b(x, \gamma) = \dfrac{1}{1 + e^{-\gamma^T x}}$

$b(x, \gamma) = \gamma_1 \mathbf{1}\{x_3 \leq \gamma_2\}$

Algorithm 10.2 *Forward Stagewise Additive Modeling.*

1. Initialize $f_0(x) = 0$.
2. For $m = 1$ to M:
   (a) Compute
   $$(\beta_m, \gamma_m) = \arg\min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)).$$
   (b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

r_{im}: residual

Idea: greedily add one function at a time

Boosted Regression Trees: $L(y, f(x)) = (y - f(x))^2$

$$\begin{aligned} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma)) &= (y_i - f_{m-1}(x_i) - \beta b(x_i; \gamma))^2 \\ &= (\underline{r_{im}} - \beta b(x_i; \gamma))^2, \qquad r_{im} = y_i - f_{m-1}(x_i) \end{aligned}$$

Efficient: No harder than learning regression trees!

Additive models

- Boosting is popular at parties: Invented by theorists, heavily adopted by practitioners.
- Computationally efficient with "weak" learners. But can also use trees! Boosting can scale.
- Gradient boosting generalization with good software packages (e.g., *XGBoost)*. Effective on Kaggle

Bagging versus Boosting

- Bagging *averages* many **low-bias**, **lightly dependent** classifiers to reduce the variance
- Boosting *learns* linear combination of **high-bias**, **highly dependent** classifiers to reduce error

Which algorithm do I use?

TABLE 10.1. *Some characteristics of different learning methods. Key:* ▲ *= good,* ◆ *=fair, and* ▼ *=poor.*

multivariate adaptive regression spline

Characteristic	Neural Nets	SVM	Trees	MARS	k-NN, Kernels
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼
Handling of missing values	▼	▼	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼
Computational scalability (large N)	▼	▼	▲	▲	▼
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼
Ability to extract linear combinations of features	▲	▲	▼	▼	◆
Interpretability	▼	▼	◆	▲	▼
Predictive power	▲	▲	▼	◆	▲