P9120 - Statistical Learning and Data Mining

Lecture 5 - CART and Ensemble Methods

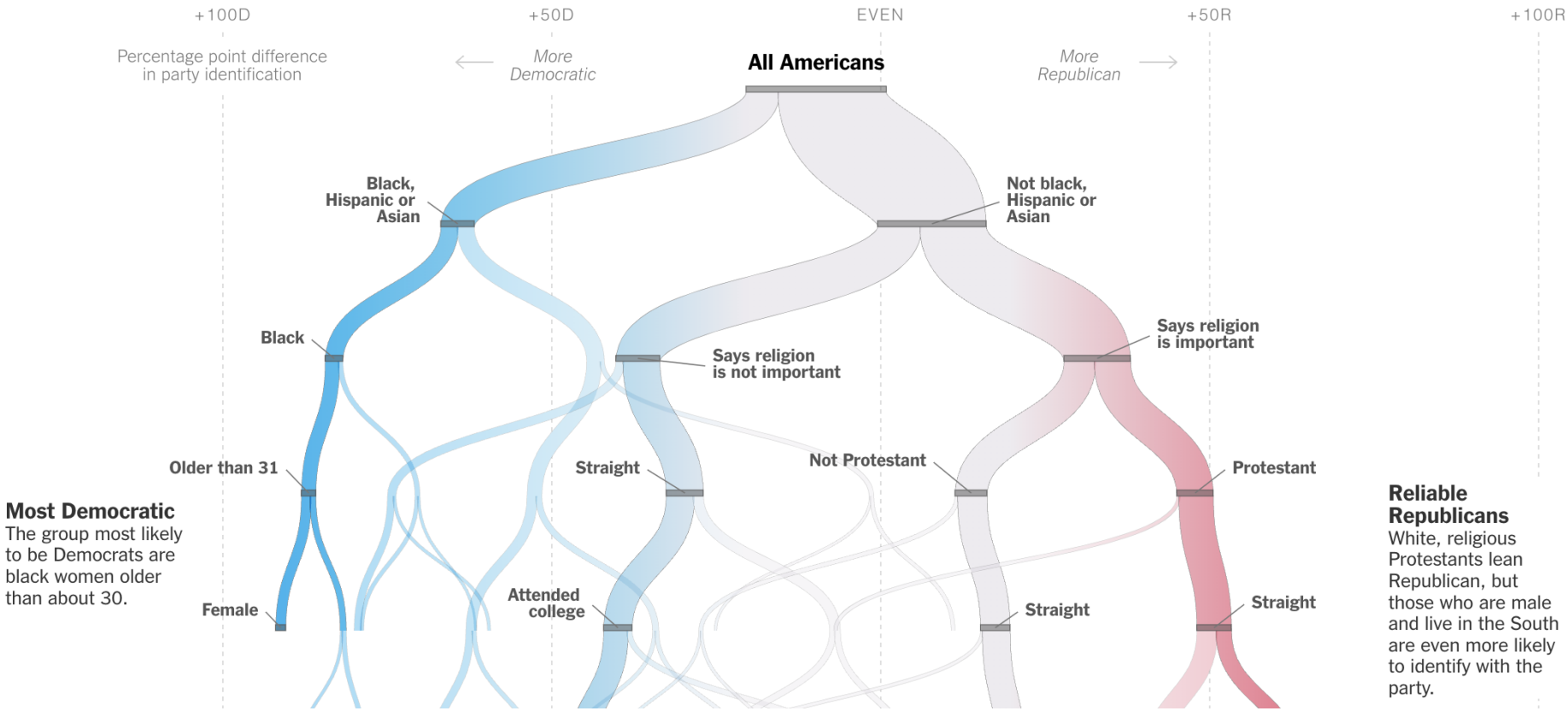
Min Qian

Department of Biostatistics, Columbia University

October 3rd, 2024

Outline

Classification Tree: Democrat or Republican? (NYT, 2019)



The quiz and tree of demographic groups use data from the 2016, 2017 and 2018 Cooperative Congressional Election Study. We applied a statistical technique called a regression tree to the self-reported party identification of respondents, including those who said they leaned toward a party but excluding independents. The demographic variables we considered were race, religion, education, age, gender, sexuality, marital status, whether the person considers religion "very" or "somewhat" important to their life, whether they have children under 18, whether they live in the South, whether they have ever been a member of a labor union, whether they immigrated to the United States and whether they or anyone in

Regression Tree: California Real Estate

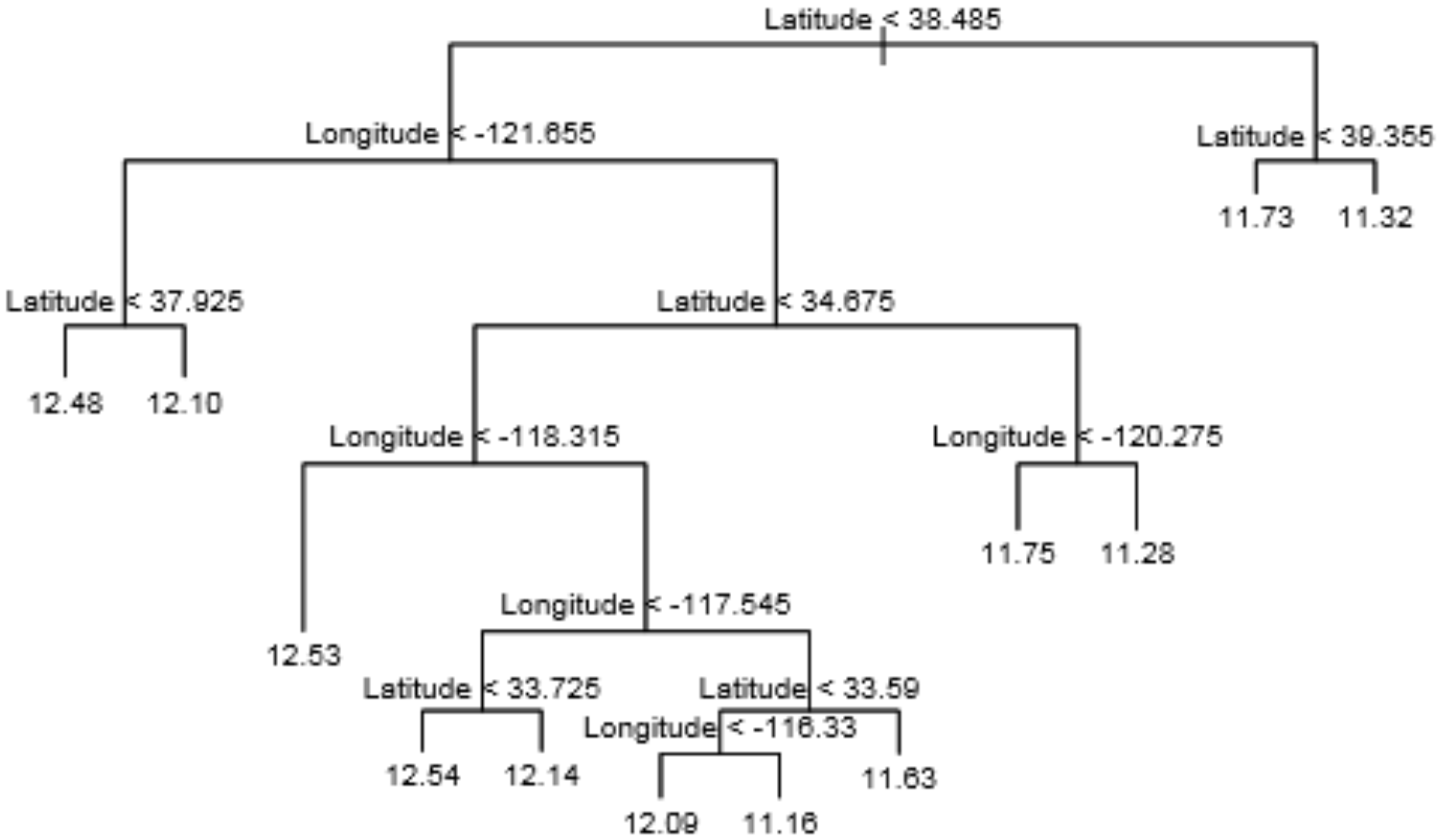


Figure 2: Regression tree for predicting California housing prices from geographic coordinates. At each internal node, we ask the associated question, and go to the left child if the answer is "yes", to the right child if the answer is "no". Note that leaves are labeled with *log* prices; the plotting function isn't flexible enough, unfortunately, to apply transformations to the labels.

Regression Tree: California Real Estate

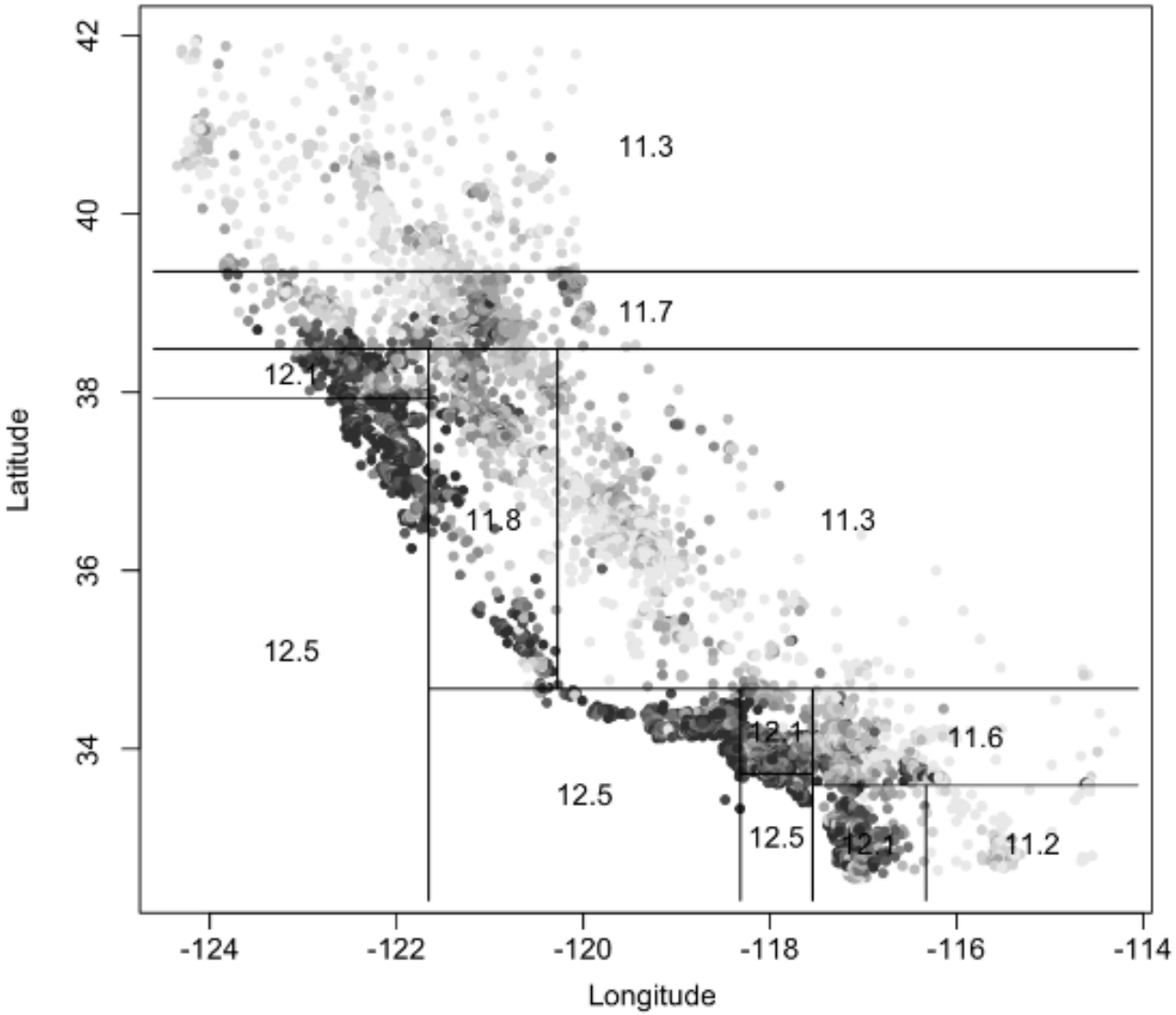
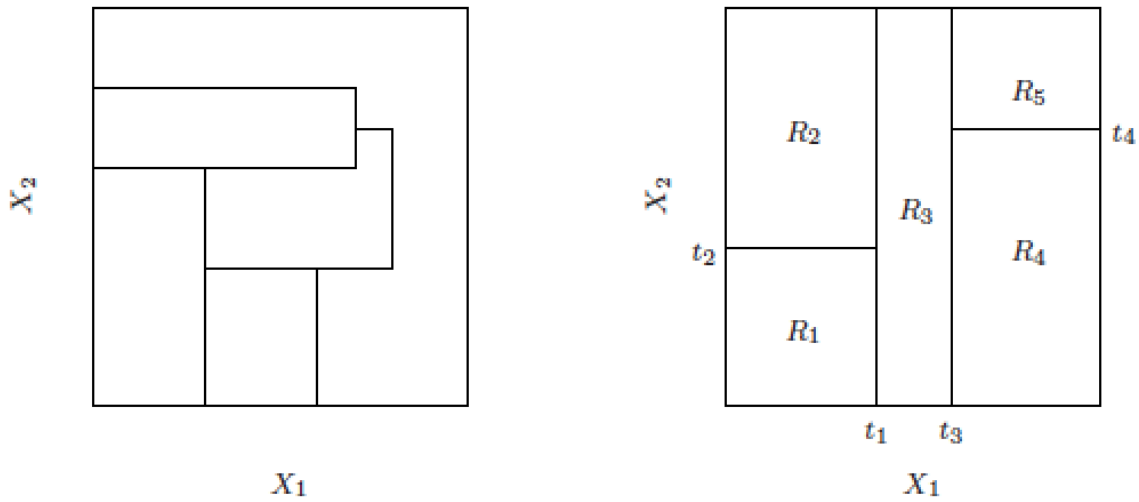


Figure 3: Map of actual median house prices (color-coded by decile, darker being more expensive), and the partition of the `treefit` tree.

Main Idea of CART

- A hierarchical yes/no questions are asked and the final decision depends on the answers to all the previous questions.
- The questions partition the space of input variables into regions with fixed prediction value.
- Example partition:

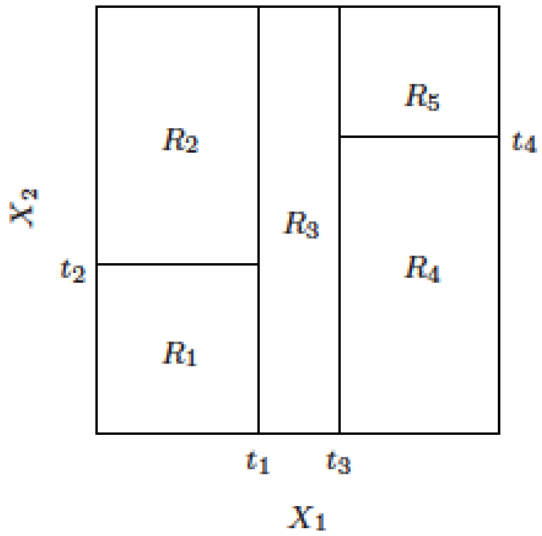


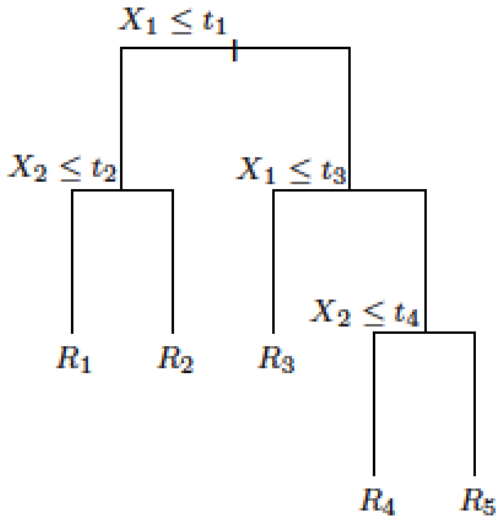
The regions on the right are easy to describe, e.g.,

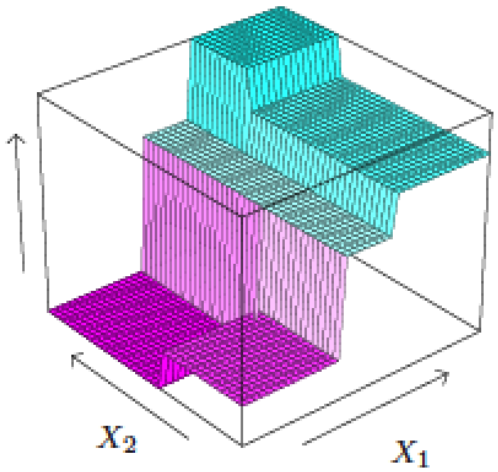
$$R_2 = \{\mathbf{x} : x_2 < t_1 \text{ and } x_2 > t_2\}.$$

Classification and Regression Trees

- E.g. regression







- Regions correspond to leaves.
- Associate each leaf with a fixed prediction value.
  - For regression tree,
  - For classification tree,
- Key questions: How do we build (learn) the tree?

Basic Ideas

Grow a big tree:

1. Start with the whole space.
2. Choose a splitting variable and a cutoff point to partition the data into two resulting regions.
3. Repeat the splitting process in step 2 on each region in a forward stepwise fashion.

Prune the tree: Sequentially collapse some branches together in a backward stepwise fashion to produce a sequence of sub-trees with different sizes.

Select the subtree with balanced training error and size.

Grow A Big Regression Tree T_0 (Greedy Algorithm)

- Consider a split at s along variable X_j,

$$R_l(j,s) = \{\mathbf{x}|x_j \leq s\}, \quad R_r(j,s) = \{\mathbf{x}|x_j > s\}$$

- Associate R_l and R_r with predictive value

$$\hat{c}_l = Ave(y_i|\mathbf{x}_i \in R_l), \hat{c}_r = Ave(y_i|\mathbf{x}_i \in R_r).$$

- Find (j,s) to minimize

$$\sum_{i \in R_l} \left(y_i - \hat{c}_l\right)^2 + \sum_{i \in R_r} \left(y_i - \hat{c}_r\right)^2.$$

- Proceed recursively, partitioning R_l and R_r by the same procedure... to obtain a big regression tree T_0.

Prune the Regression Tree

Prune the big tree T_0 to T by merging multiple leaves.

For each subtree T of T_0, its cost-complexity is measured by

$$C_\lambda(T) = \sum_{m=1}^{|T|} \text{sum of squared error}_m + \lambda|T|,$$

where λ is a tuning parameter.

Breiman (1984) and Ripley (1996) have shown that

- For a given $\lambda \geq 0$, there exists a unique smallest subtree T_λ that minimizes $C_\lambda(T)$.
- The sequence of subtrees obtained by weakest link pruning must contain T_λ, the tree that minimizes $C_\lambda(T)$!

Weakest link pruning

$$C_\lambda(T) = \sum_{m=1}^{|T|} n_m Q_m(T) + \lambda |T|.$$

- Successively collapse the internal node that produces the smallest per-node increase in $\sum_{m=1}^{|T|} n_m Q_m(T)$, where
  - $|T|$: number of leaves in T
  - $n_m = \#\{i : \mathbf{x}_i \in R_m\}$, the size of a leaf
  - $\hat{c}_m = \sum_{i:\mathbf{x}_i \in R_m} y_i / n_m$
  - $Q_m(T) = \sum_{i:\mathbf{x}_i \in R_m} (y_i - \hat{c}_m)^2 / n_m$, "node impurity" measure.
- Continue until we obtain a single-node (root) tree. This gives a (finite) sequence of subtrees.

How do we choose λ?

Classification Trees: Node Impurity

- In regression trees, squared error is a measure of "node impurity".
- In classification, we have a few choices of $Q_m(T)$.
  - Misclassification error:

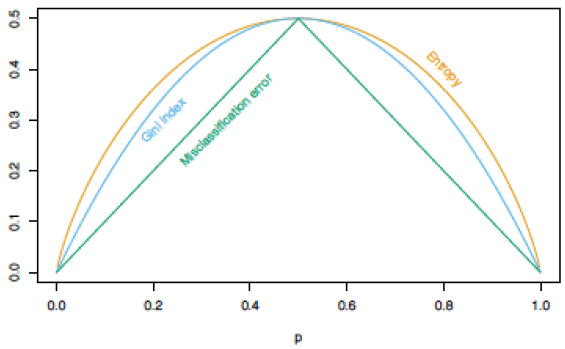
$$\frac{1}{n_m} \sum_{\mathbf{x}_i \in R_m} 1_{y_i \neq \hat{c}_m} = 1 - \max_k \hat{p}_{mk}, .$$

  - Cross-entropy or Deviance:

$$-\sum_{k=1}^{K} \hat{p}_{mk} \log \hat{p}_{mk}$$

  - Gini index:

$$\sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}).$$



Classification Tree

Grow a tree: Find (j, s) to minimize

$$n_l Q_l(T) + n_r Q_r(T).$$

Proceed recursively to obtain a large enough tree.

Prune a tree: choose T to minimize

$$C_\lambda(T) = \sum_{m=1}^{|T|} n_m Q_m(T) + \lambda |T|.$$

Common practice: use Gini or Deviance to grow the tree, misclassification error to prune.

CART Summary

Strengths:

- non-model based, very flexible, very intuitive
- natural graphical display, easy to interpret
- naturally multi-class and nonlinear

Important limitations:

- hard splits (non-smooth regression)
- limited to axis-aligned splits
- often high variance (i.e. unstable classifiers) despite regularization

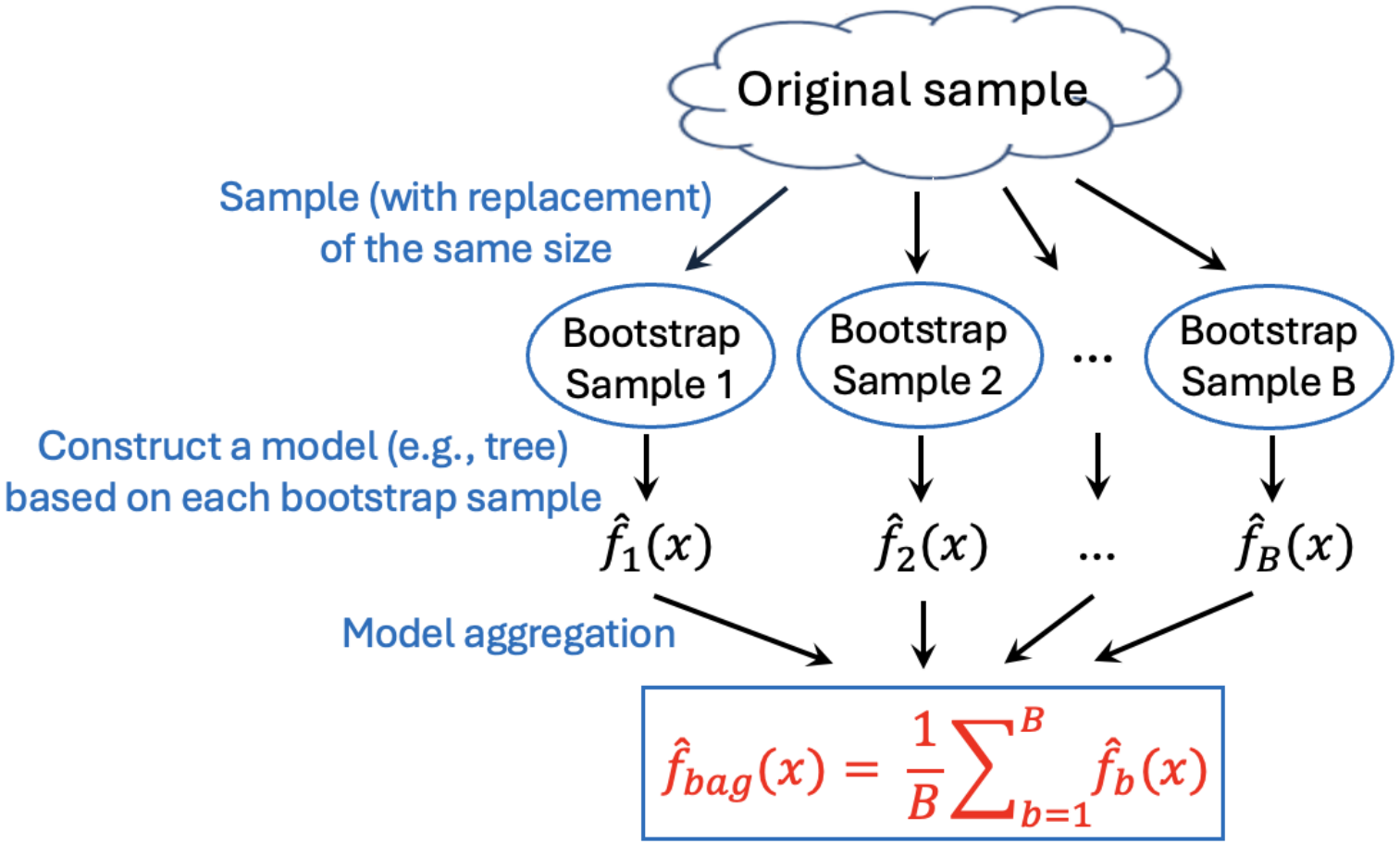
Ensemble Methods

Combining predictions from many trees (or, other predictive models) may improve performance

- Bagging (Breiman, 1996): Fit many trees using bootstrap samples, and average the results.
- Random Forest (Breiman, 1999): Improvements over bagging with randomized trees.
- Boosting (Freund and Schapire, 1997; Friedman, 2001): Fit an additive model based on a set of shallow trees.

Bagging

Bagging or Bootstrap Aggregating averages the prediction over a collection of bootstrap samples from the training data.



Bagging estimate has much lower variance than the original estimate.

Bagging for Classification ($Y \in \{1, \ldots, K\}$)

For each bootstrap sample $b = 1, \ldots, B$,

- $\hat{\mathbf{f}}_b(x) = (0, \ldots, 0, 1, 0, \ldots, 0)$ is the estimated classification rule, where the location of 1 indicates the predicted class for $X = x$.
- $\hat{\mathbf{p}}_b(x) = (\hat{p}_{m,1}(x), \ldots, \hat{p}_{m,K}(x))$, where $\hat{p}_{b,k}(x)$ is the predicted probability of observation x belonging to class k.

- Consensus Vote

$$\hat{\mathbf{f}}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{\mathbf{f}}_b(x),$$

The bagged classifier is $\widehat{G}_{bag}(x) = \arg\max_k \hat{f}_{bag,k}(x)$.

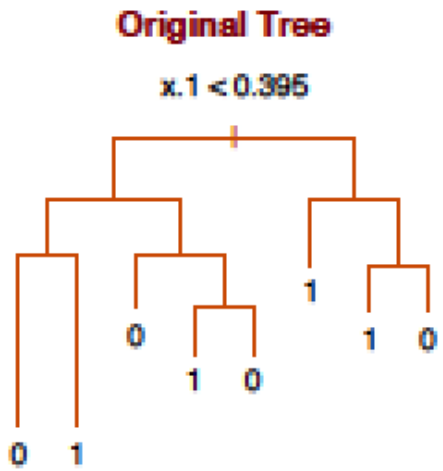
- Probability Average

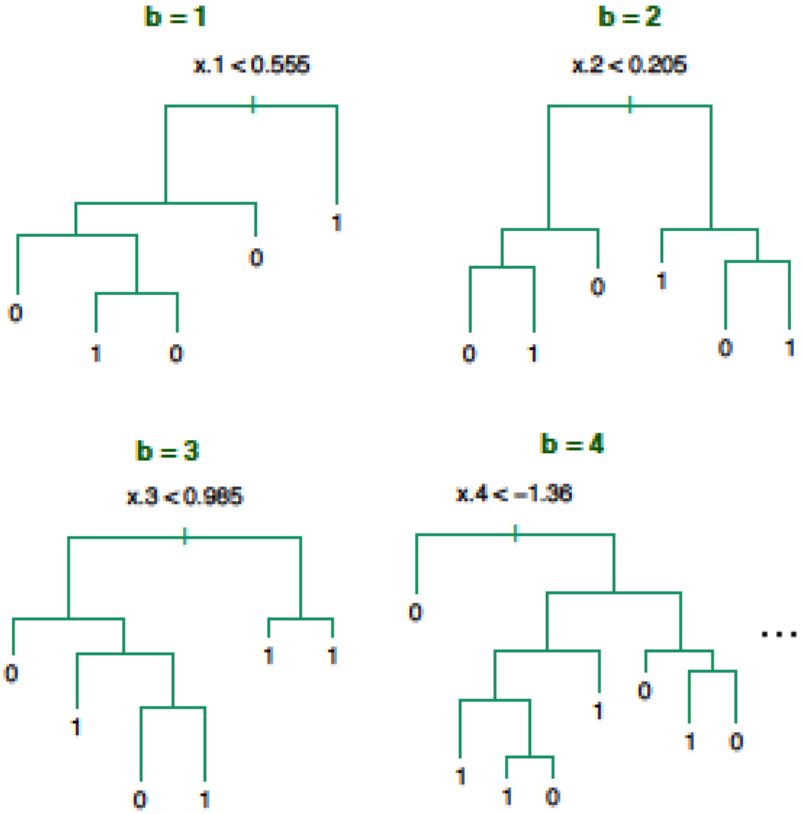
$$\hat{\mathbf{p}}_{bag}(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{\mathbf{p}}_m(x).$$

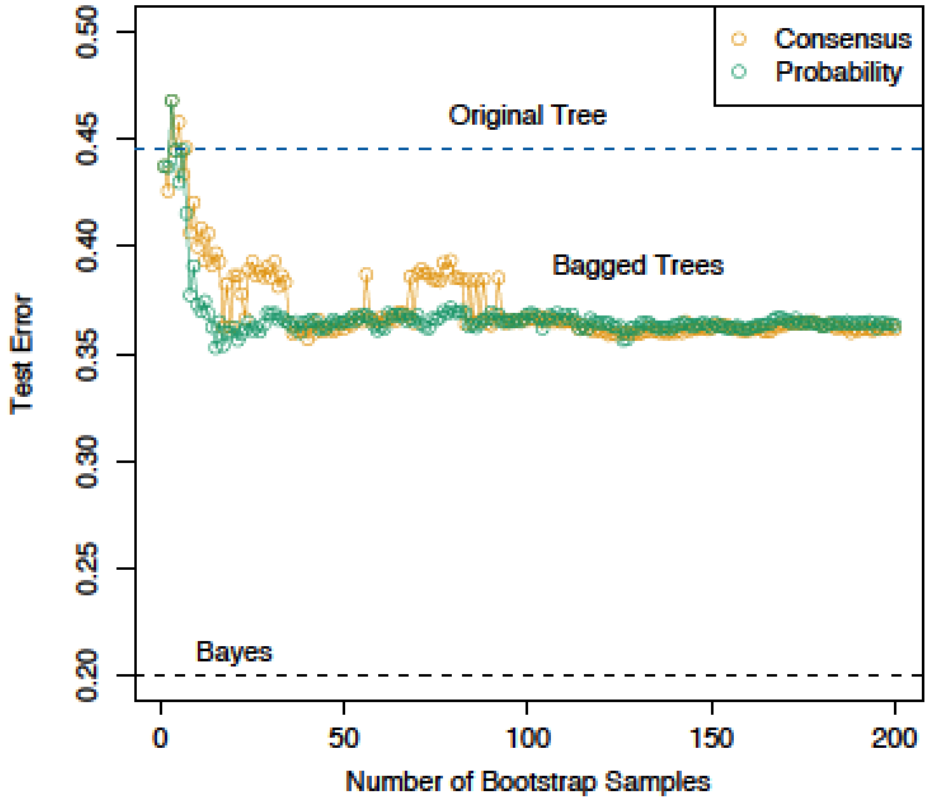
The bagged classifier is $\widehat{G}_{bag}(x) = \arg\max_k \hat{p}_{bag,k}(x)$.

Bagging Example: Trees with Simulated Data

- 2 class-problem with 5 features, each standard normal with pairwise correlation= 0.95.
- The response Y is generated according to $P(Y = 1|x_1 \leq 0.5) = 0.2$ and $P(Y = 1|x_1 > 0.5) = 0.8$.
- Training sample of size 30.







- Bagging based on 200 bootstrap samples.
- Performance evaluated on a test sample of size 2000.
- Trees have high variance due to correlation among predictors.
- Bagging succeeds in smoothing out this variance and hence reducing the test error.

Remarks on Bagging

- Bagging is a smoothing operation that reduces variance, while not affecting the bias much, often resulting in the reduction of MSE
- Bagging improves the MSE of unstable predictors considerably, while doesn't change the performance of stable estimators
- In effect, the bagging procedure smooths the "hard" decision function (e.g. an indicator) by a soft decision function.

Buhlman and Yu (2002). Analyzing Bagging. Annals of Statistics, 30(4): 927-961.

Random Forest: Motivation

Bagging:

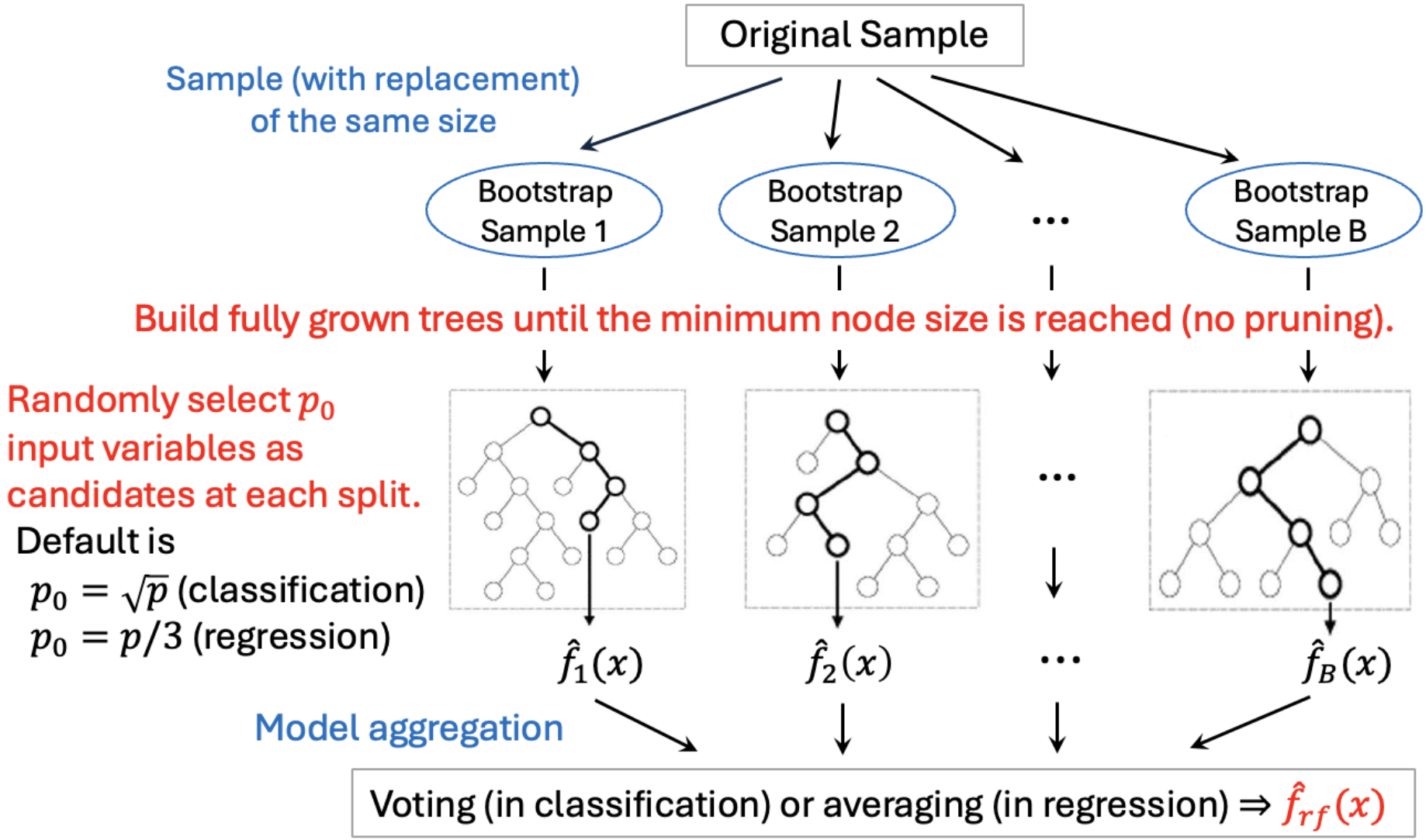
- large overlap between original sample and the bootstrap samples.
- All input variables are considered at each split
- Predictions from the bootstrap sample trees may be highly correlated.
  e.g., One very strong predictor $\rightarrow$ similar trees
- For B identically distributed variables with positive pairwise correlation ρ, the variance of the average is

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

- If correlation is large, averaging does not lead to a substantial reduction in variance!

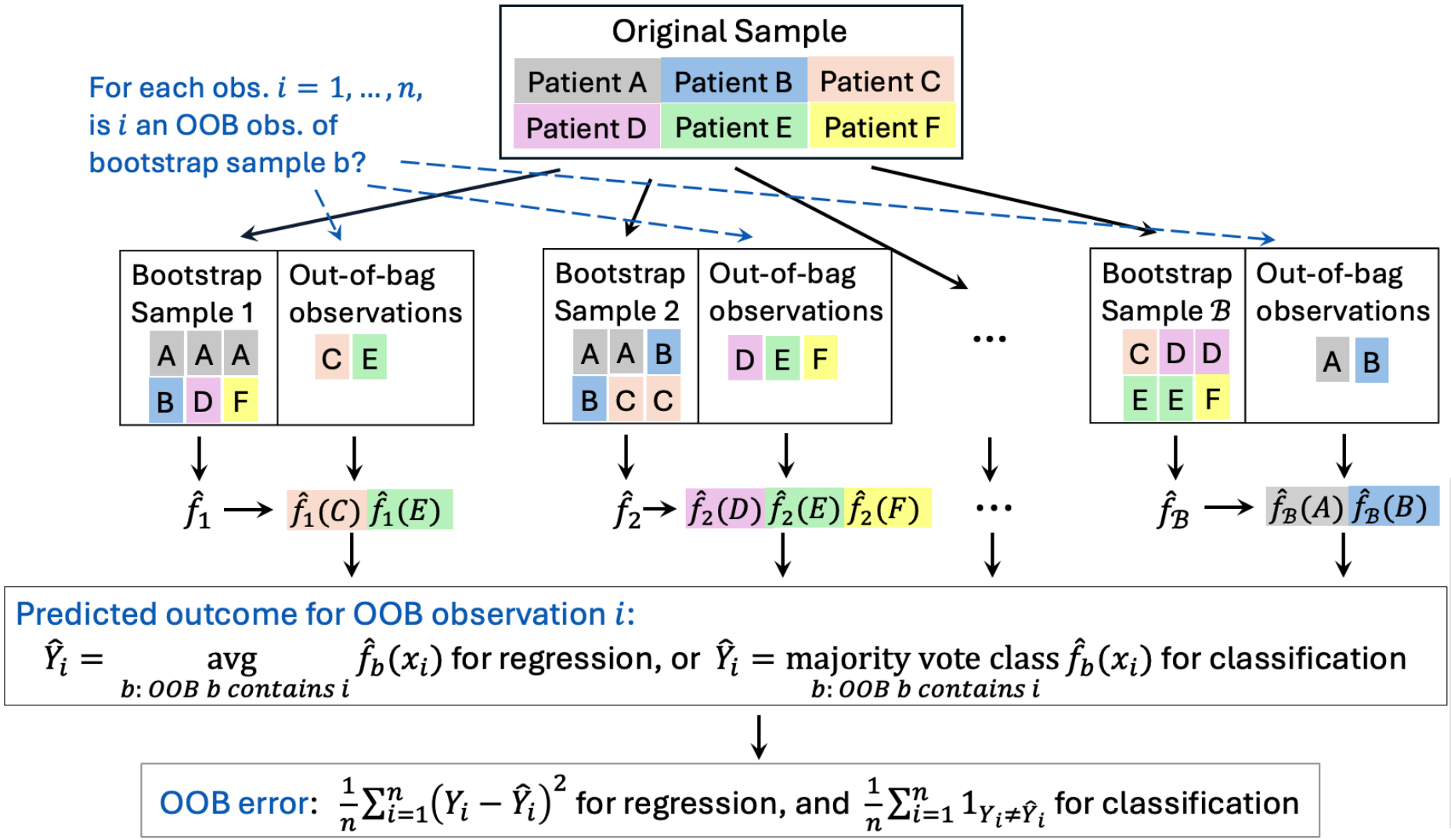
Random Forest

Idea: random subsampling the feature space (called "feature bagging") to reduce the correlation of trees based on bootstrap samples.

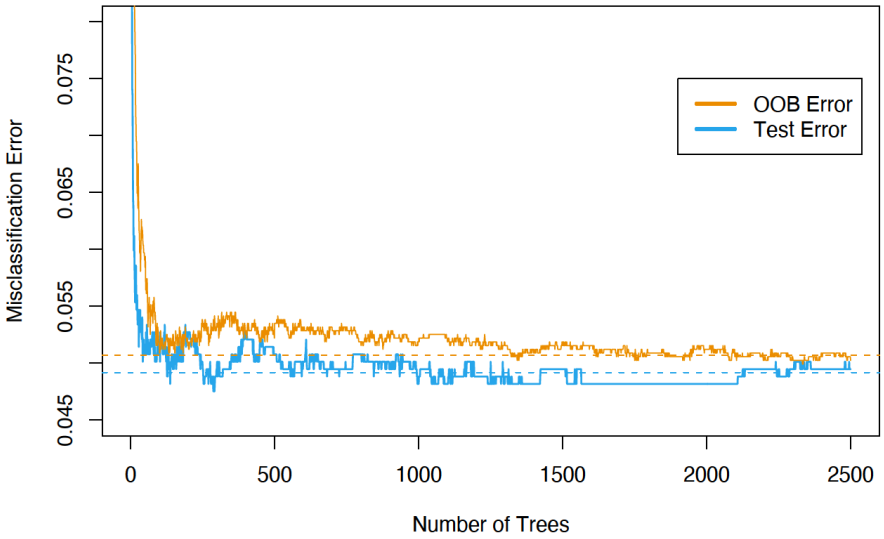


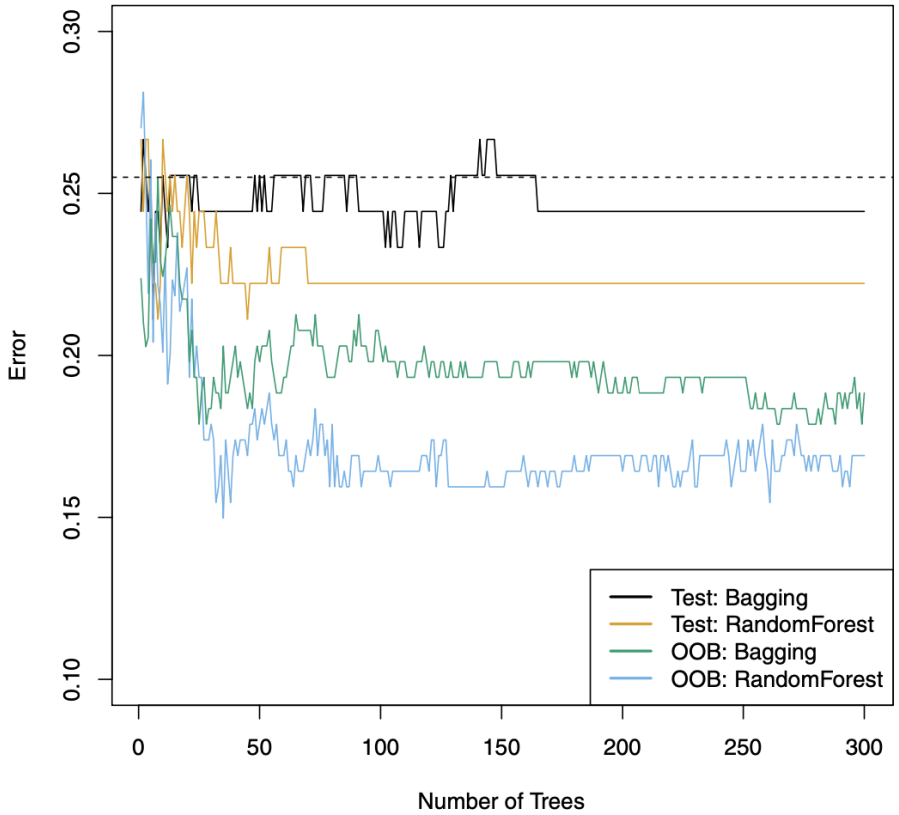
Out-of-Bag Error (OOB)

OOB error is an estimate of test error for models based on bootstrapped samples.



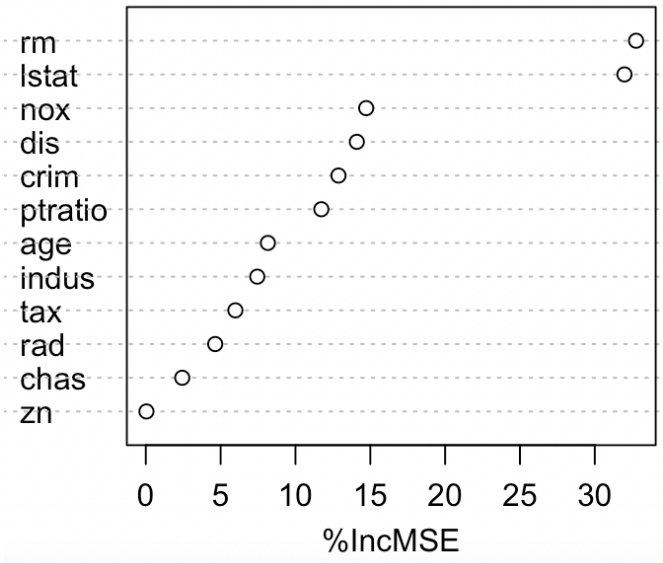
OOB

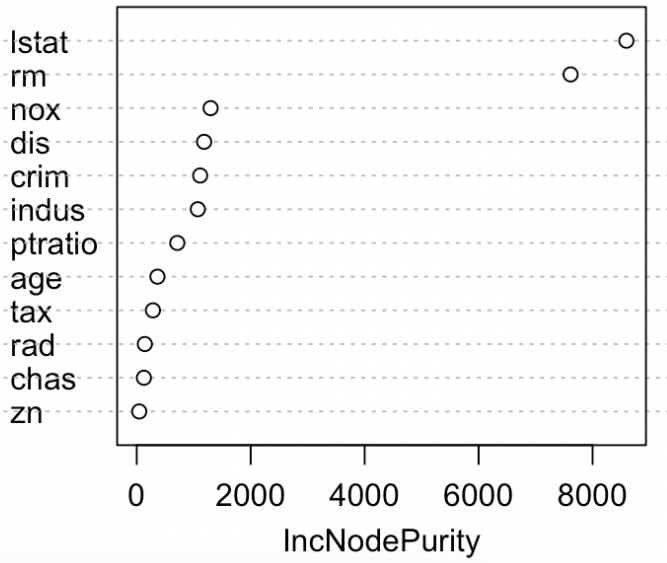


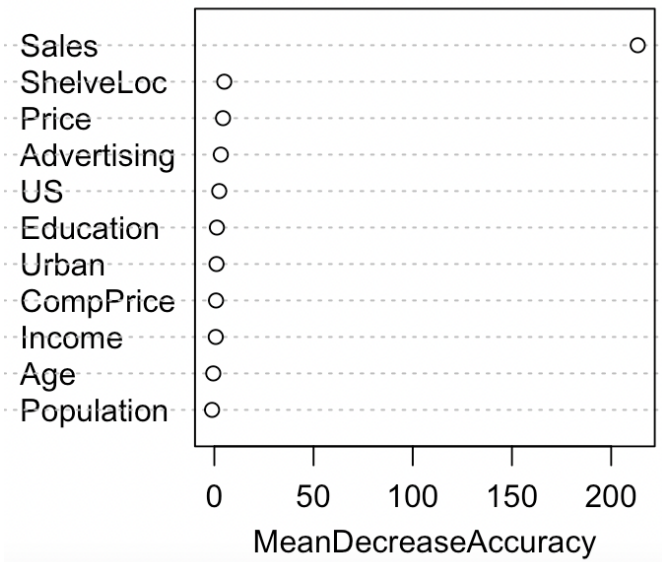


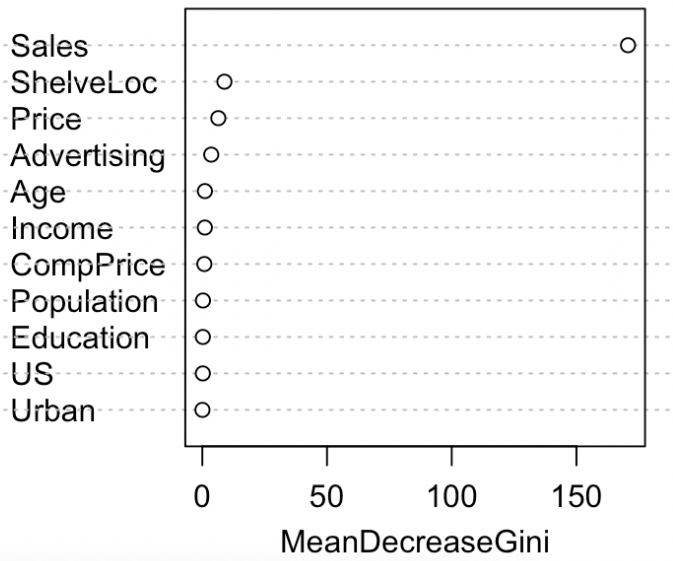
- Choose B so that OOB is stabilized. No need to perform cross-validation.

Variable Importance









Variable Importance: Improvement in Node Impurity

For a single decision tree b with L internal nodes, the importance of variable X_j is defined as

$$\mathcal{I}_j(b) = \sum_{l=1}^{L} \hat{\Delta}_l I\{X_j \text{ is used to partition the region at node } l\},$$

where $\hat{\Delta}_l$ is the improvement in risk deduction of splitting at node l.

- Square error loss for regression (IncNodePurity)
- Gini index for classification (MeanDecreaseGini)

For RF, the importance of X_j is $\sum_{b=1}^{B} \mathcal{I}_j(b)/B$.

Variable Importance: OOB Accuracy by Permutation

For a single decision tree b,

- Calculate prediction error in the OOB sample
- Randomly permute the values of X_j in the OOB sample, and then calculate the prediction error in the new OOB sample.
- Calculate the increase in prediction error as a result of the permutation

The importance of X_j is the average increase in prediction error over B trees.

- MSE for regression (%IncMSE)
- Classification error for classification (MeanDecreaseAccuracy)

(use scale=F to see the raw value of the importance measure in randomForest)

Random Forest Summary

Advantages

- Good prediction performance in most cases.
- Fast (especially when handling a large number of predictors)
- Produce variable importance measures.

Caution

- Random forests may not well handle large numbers of irrelevant features (i.e. features not predictive of the outcome).

R-packages

- randomForest; randomForestSRC (for survival data); ranger (fast implementation for high dimensional data), grf (generalized random forest for causal effect estimation).

Boosting

- Boosting: general method of converting "weak classifiers" into highly accurate prediction rule in a forward stage-wise manner.

$$G = \text{sign}\Big[\sum_m \alpha_m G_m(\mathbf{x})\Big]$$

- In each stage, introduce a weak learner to compensate the shortcomings of existing weak learners.

- A weak classifier is one whose error rate is only slightly better than random guessing:

$$\frac{1}{2} - \epsilon < \text{err} \triangleq \frac{1}{n}\sum_{i=1}^{n} 1_{y_i \neq G_m(\mathbf{x}_i)} < \frac{1}{2}.$$

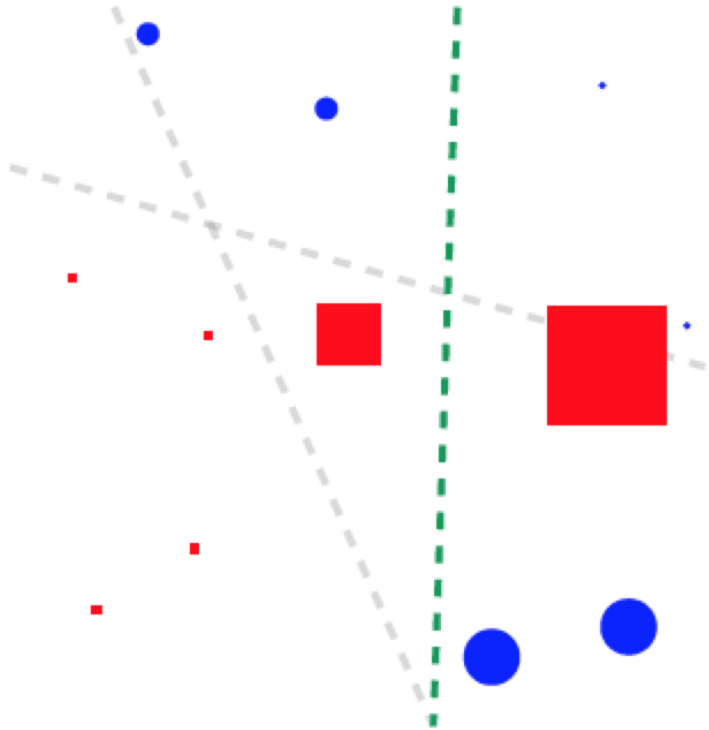
Example: A stump (single-node tree)

AdaBoost: Intuition ($Y \in \{-1, 1\}$)

Greedy algorithm:

for $m = 1, \ldots, M$,

1. Construct a weak classifier $\hat{G}_m$
2. Adjust weights: misclassified data points get "heavier" weights
3. Set vote α_m according to classification performance of $\hat{G}_m$.



Combine weak classifiers to obtain the final prediction

$$\hat{G}(\mathbf{x}) = \text{sign}\Big[\sum_{m=1}^{M} \hat{\alpha}_m \hat{G}_m(\mathbf{x})\Big].$$

AdaBoost Algorithm

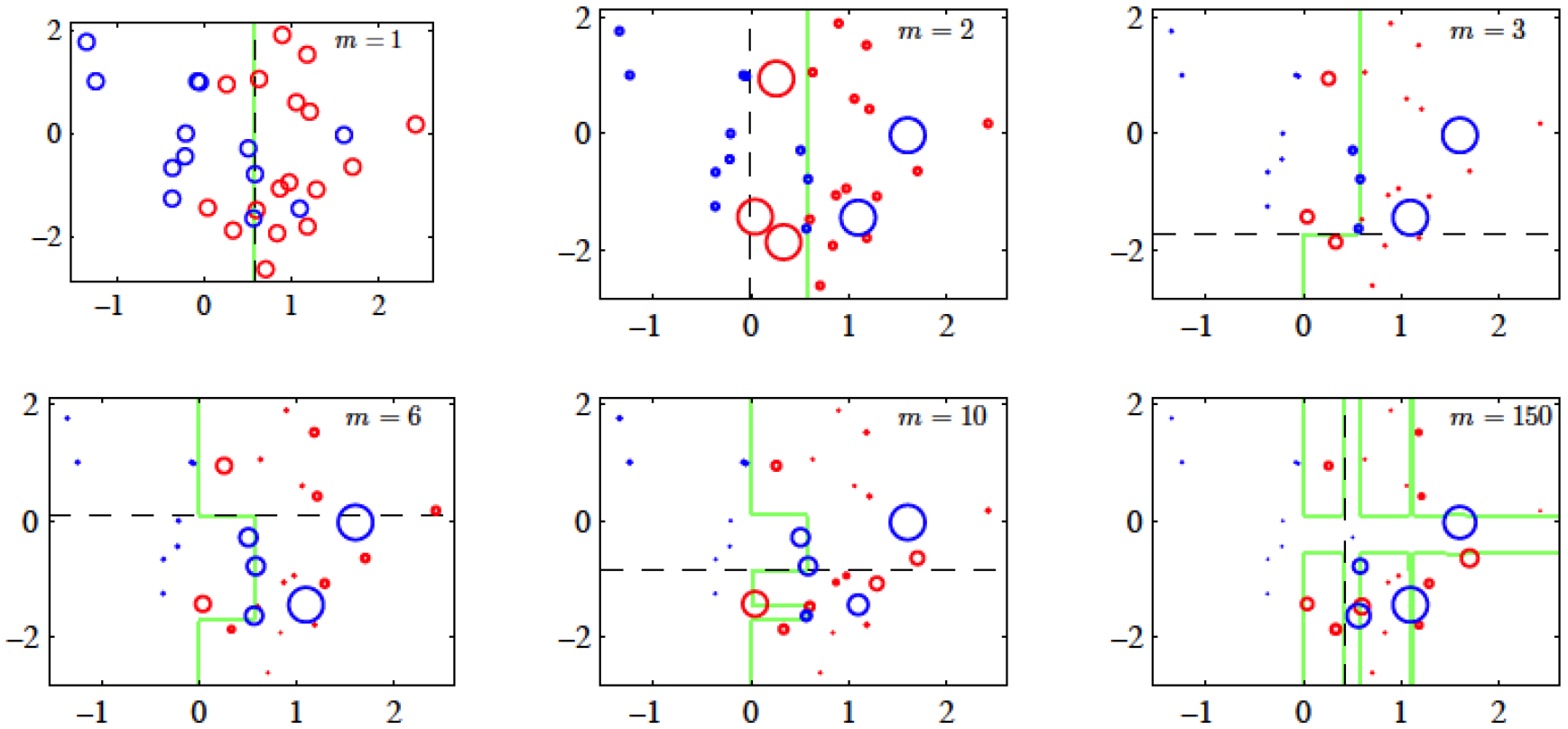
- Initialize the observation weights $w_i^{(1)} = 1, i = 1, \ldots, n$.
- For $m = 1, \ldots, M$ (pre-chosen number of iterations):
  - Fit a weak classifier $\hat{G}_m(\mathbf{x})$ to the training data with weights $w_i^{(m)}$'s;
  - Compute weighted error

$$\text{err}_m = \frac{\sum_{i=1}^n w_i^{(m)} 1_{y_i \neq \hat{G}_m(\mathbf{x}_i)}}{\sum_{i=1}^n w_i^{(m)}}.$$

  - Compute vote $\hat{\alpha}_m = \frac{1}{2} \log \frac{1-\text{err}_m}{\text{err}_m}$ (small err_m large vote).
  - Update weights $w_i^{(m+1)} = w_i^{(m)} \cdot \exp\left[2\hat{\alpha}_m 1_{y_i \neq \hat{G}_m(\mathbf{x}_i)}\right], i = 1, \ldots, n.$
    - ★ If $y_i = \hat{G}_m(\mathbf{x}_i)$, then weights remain unchanged.
    - ★ If $y_i \neq \hat{G}_m(\mathbf{x}_i)$, then $w_i^{(m+1)} = w_i^{(m)} \underbrace{(1 - \text{err}_m)/\text{err}_m}_{>1}$.
- Final prediction: $\hat{G}(\mathbf{x}) = \text{sign}\left[\sum_{m=1}^M \hat{\alpha}_m \hat{G}_m(\mathbf{x})\right]$.

Boosting Decision Stumps

Example of a boosting run



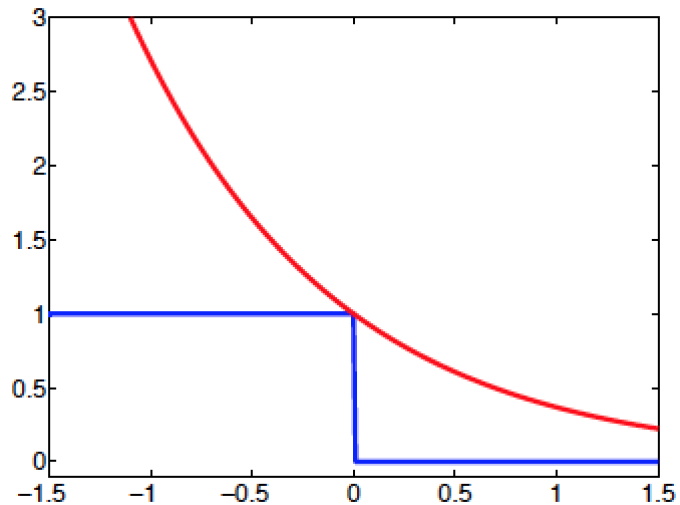
Adaboost, Additive Models, and Exponential Loss

$$\hat{G}(\mathbf{x}) = \text{sign}\Big[\sum_{m=1}^{M} \hat{\alpha}_m \hat{G}_m(\mathbf{x})\Big].$$

Adaboost is equivalent to forward stagewise fitting an additive model using the exponential loss (a discovery by Friedman et al. 2000).

- $1_{z<0} \leq \exp(-z)$.
- For any classifier $G(X) = \text{sign}(f(X))$,

  $1_{Y \neq G(X)} = 1_{Yf(X)<0} \leq \exp(-Yf(X))$.
- The population risk minimizer



$$f^*(X) = \arg\min_f E(\exp(-Yf(X)) = \frac{1}{2}\log\frac{P(Y=1|X)}{P(Y=-1|X)}.$$

Forward Stagewise Fitting for Additive Models

Additive Model:

Basic models, e.g. trees

$b(\mathbf{x};\gamma_1)$ $b(\mathbf{x};\gamma_2)$ $\cdots$ $b(\mathbf{x};\gamma_M)$

α_1 α_2 ... α_M

$$f(\mathbf{x}) = \sum_{m=1}^{M} \alpha_m b(\mathbf{x};\gamma_m)$$

Forward stagewise fitting:

Initialize $\hat{f}_0(\mathbf{x}) = 0$

For $m = 1, \ldots, M$

$$(\hat{\alpha}_m, \hat{\gamma}_m) = \underset{(\alpha_m,\gamma_m)}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} L\left(y_i, \hat{f}_{m-1}(\mathbf{x}) + \alpha_m b(\mathbf{x}_i;\gamma_m)\right)$$

Set $\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \hat{\alpha}_m b(\mathbf{x};\hat{\gamma}_m)$

Final Estimated model:

$$\hat{f}(\mathbf{x}) = \hat{f}_M(\mathbf{x}) = \sum_{m=1}^{M} \hat{\alpha}_m b(\mathbf{x};\hat{\gamma}_m)$$

Forward Stagewise Additive Modeling Using Exponential Loss

- Initialize $\hat{f}_0(\mathbf{x}) = 0$.

- For $m = 1, \ldots, M$,
  - Choose $\alpha_m > 0$ and weak learner $G_m(\mathbf{x})$ to minimize the exponential loss

$$\sum_{i=1}^{n} \exp\Big\{-y_i\Big[\hat{f}_{m-1}(\mathbf{x}_i) + \alpha_m G_m(\mathbf{x}_i)\Big]\Big\} \triangleq \sum_{i=1}^{n} \hat{w}_i^{(m)} \exp\Big\{-y_i \alpha_m G_m(\mathbf{x}_i)\Big\}.$$

$\hat{w}_i^{(m)}$ captures the "history" of classification of $\mathbf{x}_i$ by $\hat{f}_{m-1}(\mathbf{x})$.

  - Set $\hat{f}_m(\mathbf{x}) = \hat{\alpha}_1 \hat{G}_1(\mathbf{x}) + \ldots + \hat{\alpha}_m \hat{G}_m(\mathbf{x})$.

- The final prediction rule

$$\hat{G}(\mathbf{X}) = \text{sign}\Big(\hat{f}_M(\mathbf{x})\Big).$$

Optimizing Weak Learner

$$\sum_{i=1}^{n} \hat{w}_i^{(m)} \exp\Big\{ - y_i \alpha_m G_m(\mathbf{x}_i) \Big\}$$

$$= \exp(-\alpha_m) \sum_{i: y_i = G_m(\mathbf{x}_i)} \hat{w}_i^{(m)} + \exp(\alpha_m) \sum_{i: y_i \neq G_m(\mathbf{x}_i)} \hat{w}_i^{(m)}$$

$$= \exp(-\alpha_m) \sum_{i=1}^{n} \hat{w}_i^{(m)} + (\exp(\alpha_m) - \exp(-\alpha_m)) \sum_{i} \hat{w}_i^{(m)} 1_{y_i \neq G_m(\mathbf{x}_i)}$$

- Choose G_m to minimizes $\sum_i \hat{w}_i^{(m)} 1_{y_i \neq G_m(\mathbf{x}_i)} \Rightarrow$ weak learner $\hat{G}_m$.
- Choose α_m to minimize $\sum_{i=1}^{n} \hat{w}_i^{(m)} \exp\Big\{ - y_i \alpha_m \hat{G}_m(\mathbf{x}_i) \Big\}$:

$$\hat{\alpha}_m = \frac{1}{2} \log \frac{\sum_i \hat{w}_i^{(m)} 1_{y_i = \hat{G}_m(\mathbf{x}_i)}}{\sum_i \hat{w}_i^{(m)} 1_{y_i \neq \hat{G}_m(\mathbf{x}_i)}} = \frac{1}{2} \log \frac{1 - \text{err}_m}{\text{err}_m}.$$

Weights Update

- Initial weights: $\hat{w}_i^{(1)} = 1, i = 1, \ldots, n$.
- ...
- At the $(m+1)$-th step,

$$\begin{aligned}\hat{w}_i^{(m+1)} =& \exp(-y_i \hat{f}_m(\mathbf{x}_i)) \\ =& \hat{w}_i^{(m)} \exp\left(-y_i \hat{\alpha}_m \hat{G}_m(\mathbf{x}_i)\right) \\ =& \hat{w}_i^{(m)} \exp\left(2\hat{\alpha}_m 1_{y_i \neq \hat{G}_m(\mathbf{x}_i)} - \hat{\alpha}_m\right),\end{aligned}$$

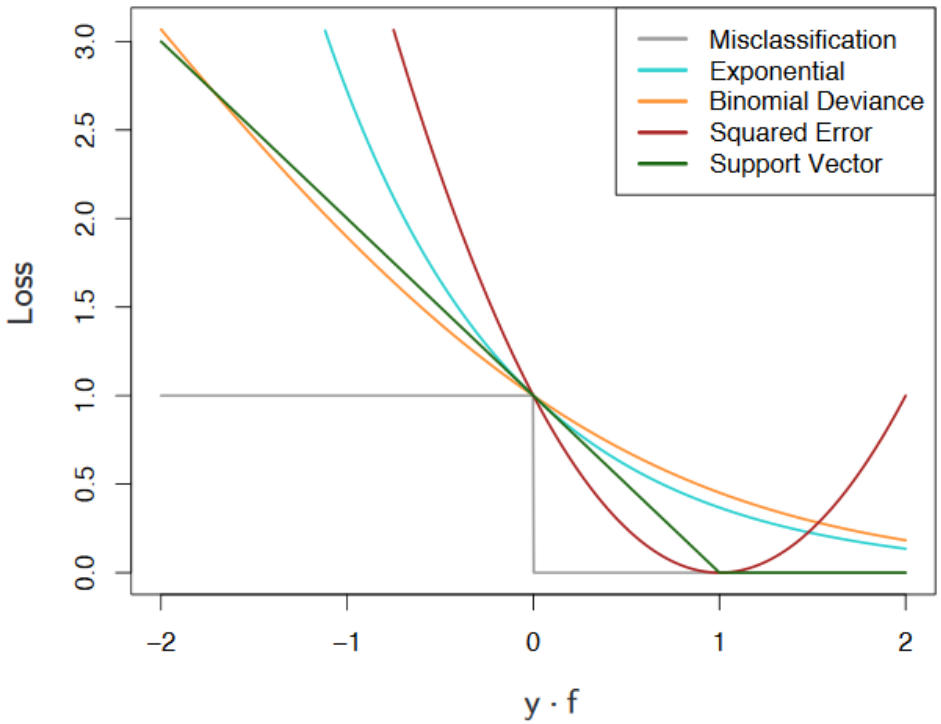
which is equivalent to the weights used in AdaBoost (normalized version).

Boosting and Bias-variance Tradeoff

$$\hat{G}(X) = \text{sign}\left(\sum_{m=1}^{M} \hat{\alpha}_m \hat{G}_m(X)\right).$$

- What determines the complexity of boosted classifier?
  Complexity of weak classifiers $\hat{G}_m(X)$ and the number M.
- Regularization of $\hat{G}(X)$: early stopping
- Typically use simple weak classifiers: stumps, shallow decision trees.

Classification as Risk Minimization



- $Y \in \{-1, 1\}$
- Classification rule: $G(x) = sign(f(x))$.

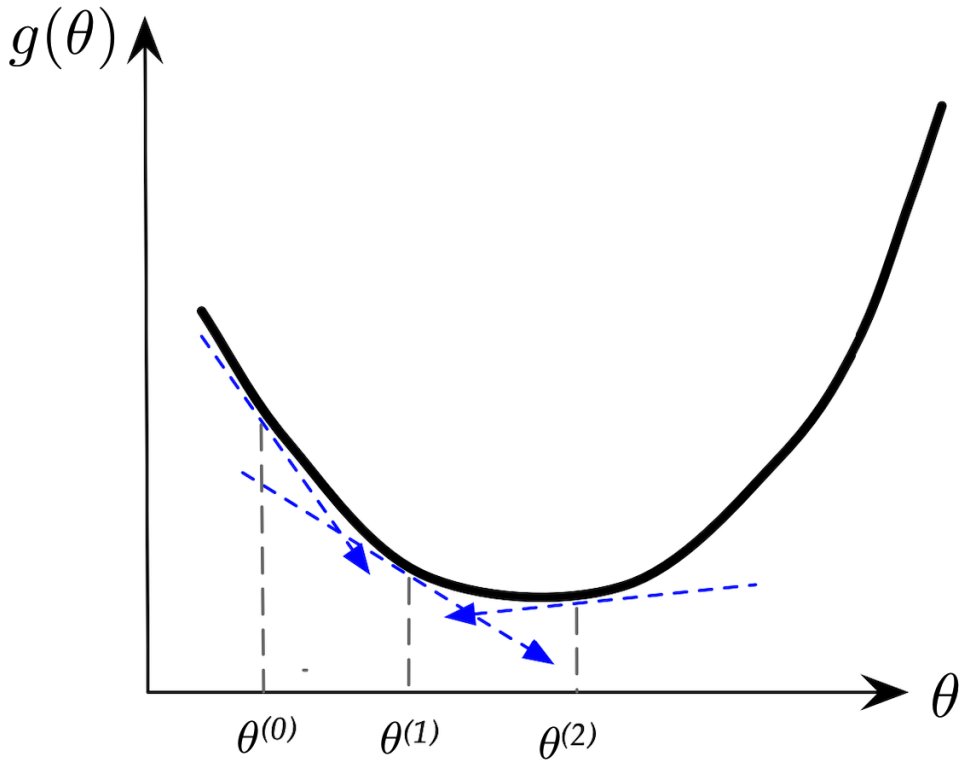
- Misclassification error (0-1 loss): $1_{yf(x)<0}$
- SVM (hinge loss): $(1 - yf(x))_+$
- Adaboost (exponential): $\exp(-yf(x))$
- Regression (squared error): $[y - f(x)]^2 = [1 - yf(x)]^2$
- Logistic regression (binomial deviance): $\log[1 + \exp(-yf(x))]$

Gradient Boosting: Gradient Descent + Boosting

Gradient Descent:
Minimize a function $g(\theta)$ by moving in the opposite direction of the gradient.

$$\theta_{new} = \theta_{old} + \rho\Big[- \frac{\partial g(\theta)}{\partial \theta}\Big]\Big|_{\theta=\theta_{old}}$$

Boosting: Add "weak learner" that predicts negative gradient.



Gradient Boosting: Start with regression

- Goal: Find f to minimize $EL(Y, f(X)) = E[Y - f(X)]^2$
- For $X = \mathbf{x}$, minimize $R(f(\mathbf{x})) = E[(Y - f(X))^2 | X = \mathbf{x}]$
- Gradient Descent:

$$\begin{aligned} f_{new}(\mathbf{x}) =& f_{old}(\mathbf{x}) + \rho \Big[- \frac{\partial R(f)}{\partial f} \Big] \Big|_{f=f_{old}} \\ =& f_{old}(\mathbf{x}) + \rho E[Y - f_{old}(X) | X = \mathbf{x}] \end{aligned}$$

- Build a weak learner $h_{new} : X \to \widetilde{Y}$, where $\widetilde{Y} = Y - f_{old}(X)$.
- Set step size $\rho_{new} = \arg\min_\rho \sum_{i=1}^n \left[y_i - (f_{old}(\mathbf{x}_i) + \rho h_{new}(\mathbf{x}_i)) \right]^2$.
- Update $f_{new}(\mathbf{x}) = f_{old}(\mathbf{x}) + \rho_{new} h_{new}(\mathbf{x})$.

Gradient Tree Boosting

- Fit a shallow regression tree (input $\mathbf{x}$, output $y - f_{old}(\mathbf{x})$):

$$h_{new}(\mathbf{x}) = \sum_{j=1}^{J} b_j 1_{(\mathbf{x}) \in R_j},$$

  where $\{R_j, j = 1, \ldots, J\}$ is a partition of space.

- $f_{new}(\mathbf{x}) = f_{old}(\mathbf{x}) + \sum_{j=1}^{J} \gamma_j 1_{(\mathbf{x}) \in R_j}$, where $\gamma_j = \rho_{new} b_j$.

- Choose $\{\gamma_j, j = 1, \ldots, J\}$ to minimize

$$\sum_{i=1}^{n} L\left(y_i, f_{old}(\mathbf{x}_i) + \sum_{j=1}^{J} \gamma_j 1_{(\mathbf{x}_i) \in R_j}\right) = \sum_{j=1}^{J} \sum_{i:\mathbf{x}_i \in R_j} L(y_i, f_{old}(\mathbf{x}_i) + \gamma_j)$$

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 $\hat{f}(x) = f_M(x)$.

Gradient Boosting: Binary Classification $Y \in \{0, 1\}$

- $\log \frac{p(\mathbf{x})}{1-p(\mathbf{x})} = f(\mathbf{x})$, where $p(\mathbf{x}) = P(Y = 1|X = \mathbf{x})$.
- Goal: Estimate $f(X)$ using gradient boosting.
- Use Binomial Deviance as loss function

$$\begin{aligned} L(Y, f(X)) &= -Y \log p(X) - (1 - Y) \log[1 - p(X)] \\ &= -Y f(X) + \log \left[1 + \exp(f(X))\right] \end{aligned}$$

- Negative gradient w.r.t. f is $Y - p(X)$.

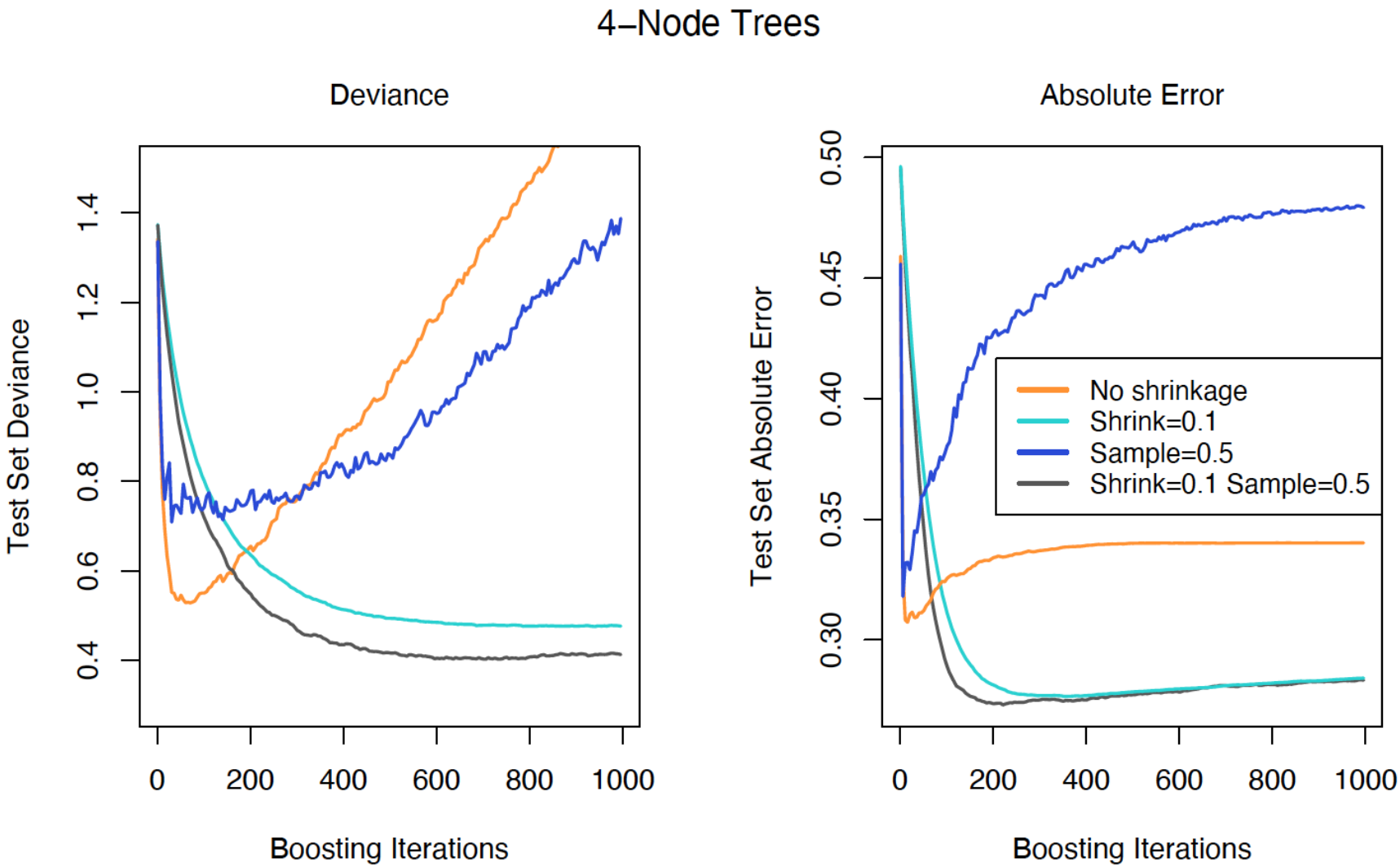
Tuning Parameters of Gradient Boosting

- Tree size $J_m = J$
  - A tree with size J could represent no more than $J-1$-order interactions.
  - Experience so far indicates that $4 \leq J \leq 8$ works well.

- Number of weak learners M: early stopping based on a small validation set.

- Other fine tuning.
  - Learning rate: $f_m(\mathbf{x}) = f_{m-1}(\mathbf{x}) + \upsilon \sum_{j=1}^{J_m} \gamma_{jm} I(\mathbf{x} \in R_{jm})$ for some $0 < \upsilon < 1$. Friedman suggested $\upsilon < 0.1$.
  - Subsampling: grow a tree to predict the negative gradient on a random subsample (e.g. 50% of data)

Tuning Parameters



XGBoost (eXtreme Gradient Boosting)

Why XGBoost?

XGBoost gained significant favor in the last few years as a result of helping individuals and teams win virtually every Kaggle structured data competition. In these competitions, companies and researchers post data after which statisticians and data miners compete to produce the best models for predicting and describing the data.

Initially both Python and R implementations of XGBoost were built. Owing to its popularity, today XGBoost has package implementations for Java, Scala, Julia, Perl, and other languages. These implementations have opened the XGBoost library to even more developers and improved its appeal throughout the Kaggle community.

XGBoost has been integrated with a wide variety of other tools and packages such as scikit-learn for Python enthusiasts and caret for R users. In addition, XGBoost is integrated with distributed processing frameworks like Apache Spark and Dask.

In 2019 XGBoost was named among InfoWorld's coveted Technology of the Year award winners.

- At each step m, find weak learner

$$h_m(\mathbf{x}) = \arg\min_{h} \sum_{i=1}^{n} \{L(y_i, f_{old}(\mathbf{x}_i) + h(\mathbf{x}_i)) + \Omega(h)\},$$

  where $\Omega(h)$ is a regularization term (complexity penalty).
- Set $f_{new}(\mathbf{x}) = f_{old}(\mathbf{x}) + h_m(\mathbf{x})$

XGBoost

- Taylor expansion:

$$g_i(h) \triangleq L(y_i, f_{old}(\mathbf{x}_i) + h(\mathbf{x}_i)) \approx g_i(0) + g_i'(0)h(\mathbf{x}) + \frac{1}{2}g_i''(0)h^2(\mathbf{x}_i)$$
$$= \frac{1}{2}g_i''(0)\left[h(\mathbf{x}) + \frac{g_i'(0)}{g_i''(0)}\right]^2 + \text{terms irrelevant of } h$$

- The weak learner satisfies

$$h_m = \arg\min_h \sum_{i=1}^{n} \left\{ \frac{1}{2}g_i''(0)\left[h(\mathbf{x}_i) + \frac{g_i'(0)}{g_i''(0)}\right]^2 + \Omega(h) \right\}.$$

- XGBoost reduces to the original Gradient boosting when there is no regularization and $g_i''(0)$ is replaced by a constant.

Super Learner (van der Laan, Polley and Hubbard, 2007)

Let $f_1, \ldots, f_M$ be a set of base learners (learning algorithms). Super learner aims to find a convex combination of those base learners

$$f_{SL}(\mathbf{x}) = \lambda_1 f_1(\mathbf{x}) + \ldots + \lambda_M f_M(\mathbf{x}),$$

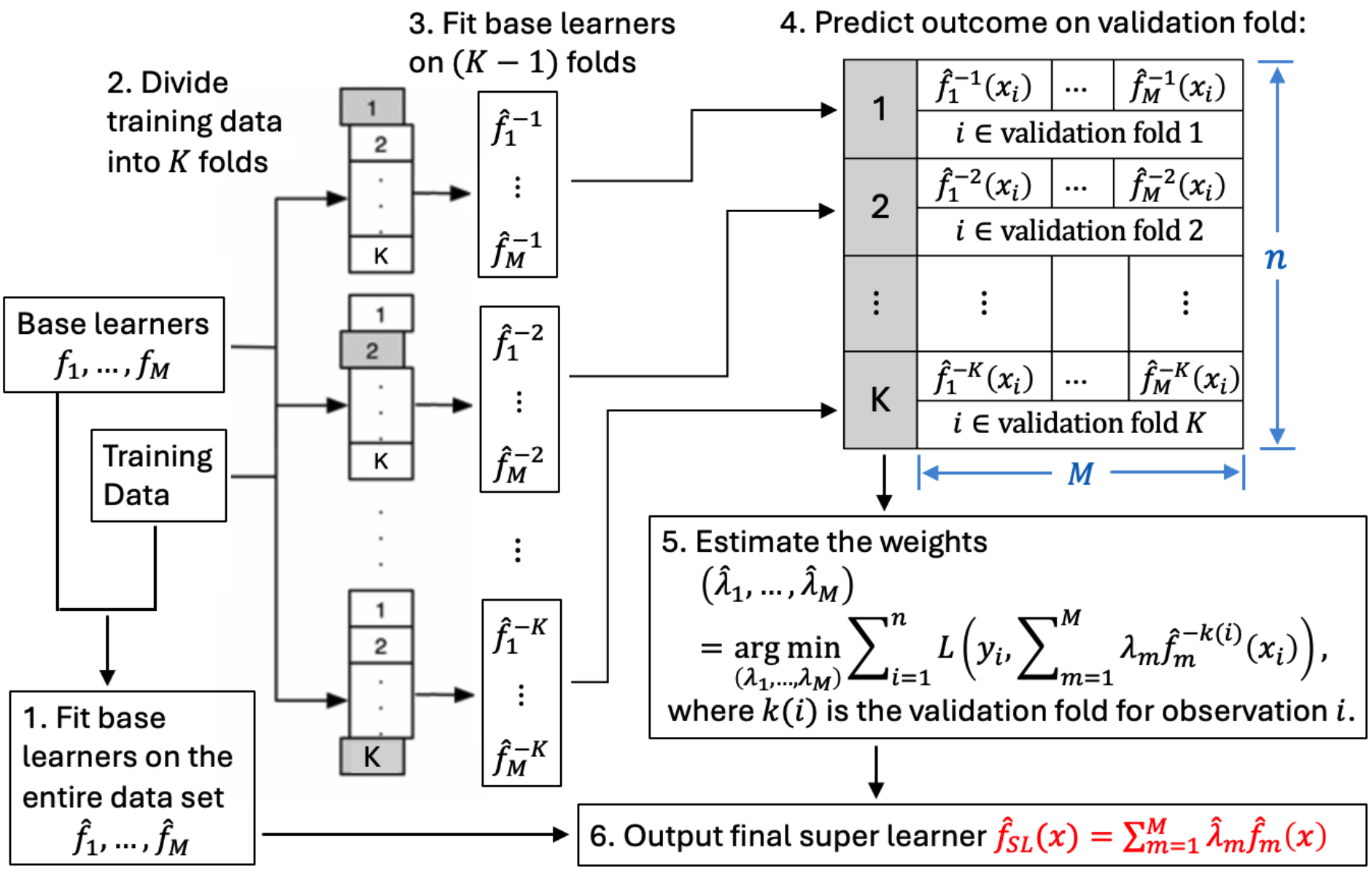
where $\lambda_j \geq 0$ and $\sum_{m=1}^{M} \lambda_j = 1$.

- Learn $\hat{f}_1, \ldots, \hat{f}_M$ from the whole training set.
- Learn $\hat{\lambda}_1, \ldots, \hat{\lambda}_M$ using cross-validation.
- Final super learner:

$$\hat{f}_{SL}(\mathbf{x}) = \hat{\lambda}_1 \hat{f}_1(\mathbf{x}) + \ldots + \hat{\lambda}_M \hat{f}_M(\mathbf{x}).$$

- R package: h2oEnsemble

Super Learner Flow Chart



Summary

- Bagging Averaging the same type of learners (from bootstrapped datasets) to reduce variance.
- Random Forest: Improvements over bagging with randomized trees to reduce correlation between trees.
- Boosting: ensemble weak learners sequentially to reduce bias.
  - Adaboost: weak learner based on weighted data points.
  - GB: weak learner to predict negative gradient.
- Super Learner: convex combination of any base learners to improve prediction performance.

Reminders

- Homework #1 is due at 9pm on October 5th.
- Quiz is due at 9pm EST on October 7th.
- Install python and Jupyter lab on your computer. In class Python lab session will start next week. Remember to bring your laptop!