Intro ML
ML. 4. Tree based algos for classification and regression

DataLab CSIC

Objectives and schedule

Improvements over CART providing good benchmarks requiring ‘little’ relatively thought.

- Bagging
- **Random forest**
- Boosting

Contents

- Review of CART
- Advanced versions

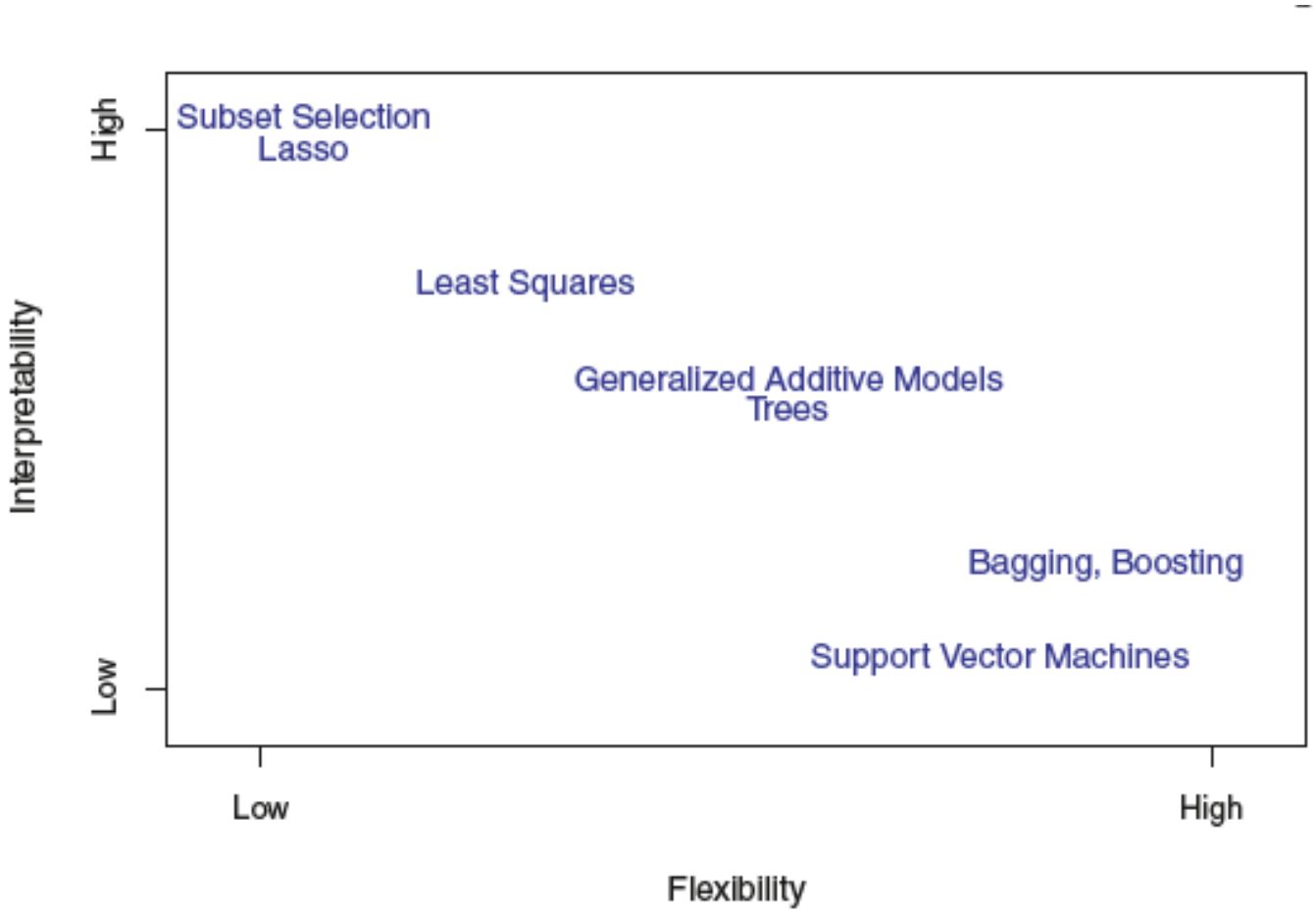
ISLR 8, CASI 17, ESL 9-15-16, Bishop 14

Labs

- Classification tree
- Regression tree
- Random forest classifier
- Gradient boosting

Trees in drug Discovery sesión by Nuria Campillo

An old good friend



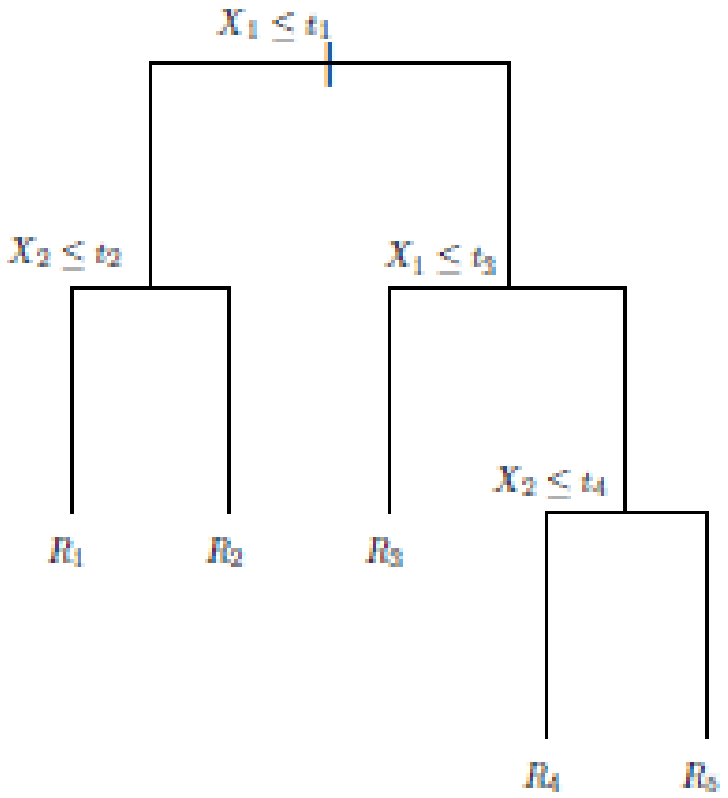
CART. Recall

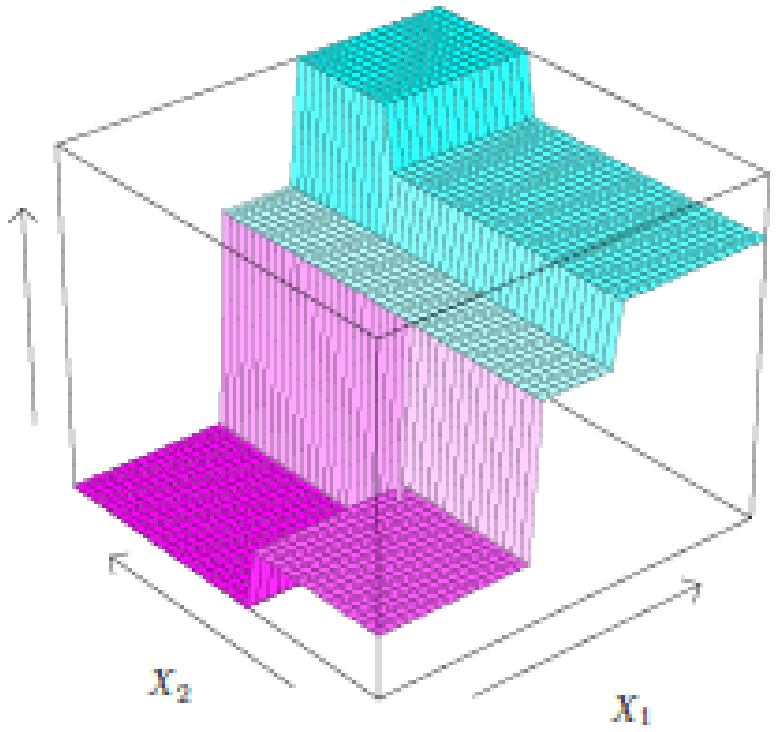
Advantages and disadvantages

- Very easy to explain
- Mirror human decision making in some domains (e.g. medicine)
- Graphical display making them interpretable (specially if small)
- Handle qualitative variables easily (no need for dummies)

- Not as good predictively as others… but improved through bagging, random forests, boosting (this chapter)

Keyword: Recursive partitioning





Regression Trees. Recall

Decision Trees. Basics

- Terminal nodes or leaves of tree
- Internal nodes (predictor split space)
- Branches

1. Partition predictor space into J regions
2. For each observation in region R_i prediction: mean response of training observations in such region

Decision trees. Basics. Step 1

- Computationally and interpretationally easier if regions are boxes
- Find boxes minimimizing RSS

$$\sum_{j=1}^{J} \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

- Computationally feasible: recursive binary splitting (top down, greedy)
- Select predictor X_j and cutpoint s leading to greatest reduction in RSS

$$R_1(j,s) = \{X : X_j < s\} \qquad R_2(j,s) = \{X : X_j \geq s\}$$

$$\sum_{i : x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i : x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

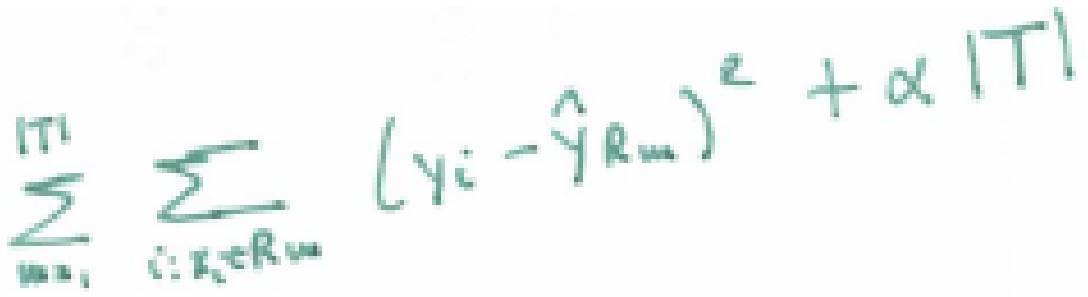
- Repeat until stopping criterion

Decision trees. Pruning

- Typically, good predictions on training set, but likely overfitting: too complex tree
- Smaller tree may lead to lower variance, better interpretation with just a bit more bias
- Solution 1. Build tree as long as RSS decrease exceeds threshold
- Solution 2. Grow large tree, then prune it back to obtain a subtree with lowest test error rate Cost complexity pruning

For each alpha find subtree T

Choose alpha by CV or VS



Regression tree algo

1. Use recursive binary splitting to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations.

2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

3. Use K-fold cross-validation to choose α. That is, divide the training observations into K folds. For each $k = 1, \ldots, K$:

   (a) Repeat Steps 1 and 2 on all but the kth fold of the training data.

   (b) Evaluate the mean squared prediction error on the data in the left-out kth fold, as a function of α.

   Average the results for each value of α, and pick α to minimize the average error.

4. Return the subtree from Step 2 that corresponds to the chosen value of α.

Classification Trees. Recall

Classification Trees. Basics

- Prediction: Most commonly occurring training observations in region.

  Class proportions

- RSS to be replaced:
  - Classification error rate. Fraction of training observations not belonging to most common class

    $1 - \max_k (\hat{p}_{mk})$

  - Gini index.

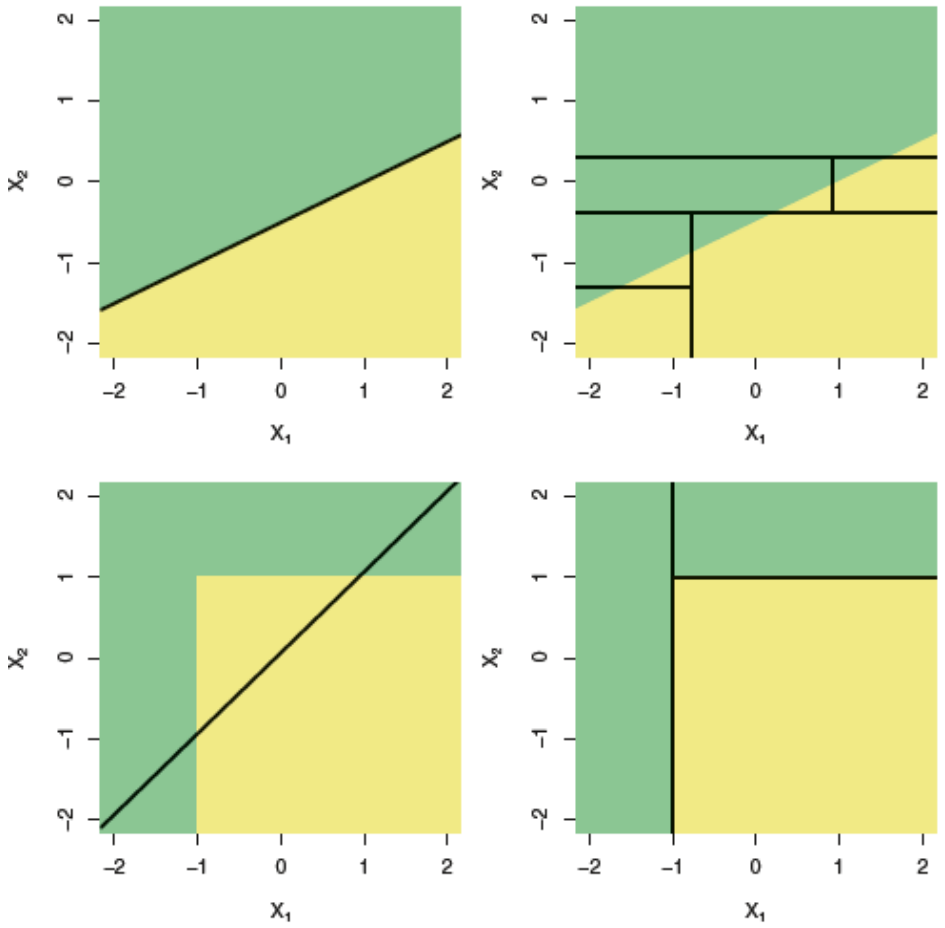
    $\sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk})$

  - Cross entropy.

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

  CER for pruning

CART vs linear model. No free lunch



Bagging

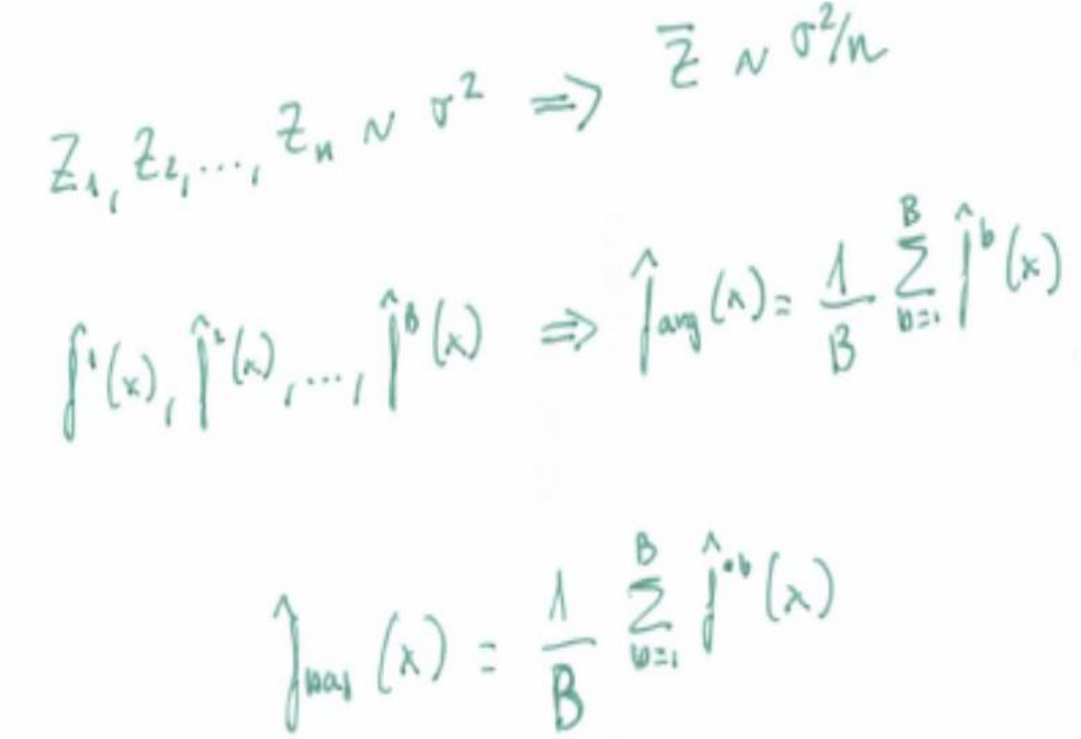
Motivation

- CART suffer from high variance: split training at random may yield two very different trees
- Bootstrap aggregation aka Bagging to reduce variance of a statistical machine learning procedure

Bootstrap

Recall

Pick many training sets
from population, build separate
model for each set and average
predictions



Way forward: Bootstrap. Sampling with
replacement from training set.

Bagging for trees

Construct B regression trees using B bootstrapped training sets. Average resulting predictions

Deep trees, not pruned (each tree has high variance, but low bias)

Averaging B trees reduces variance

Construct B classification trees using B bootstrapped training sets.

Aggregate results by majority vote

B in the 100s, 1000s

OOB error estimation

Estimate test error of bagged model (without CV or validation set)

On average, each tree uses 2/3 of data (0.632 bootstrap). The remaining 1/3 out-of-bag (OOB) observations

Approx B/3 predictions for i-th observation.

Average or majority vote-→OOB prediction

Overall OOB MSE or classification error, valid test error estimate

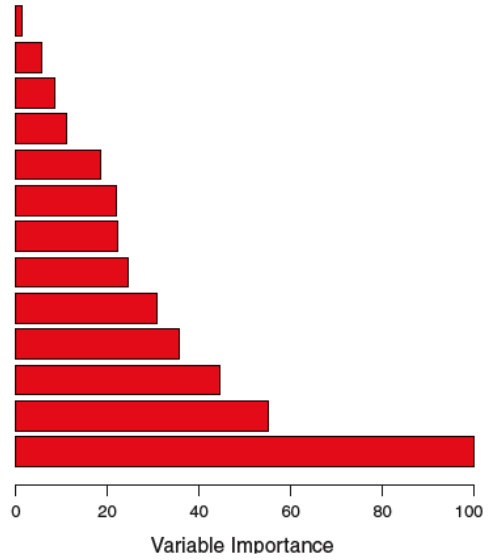
Variable importance

Improved accuracy, but difficult in interpreting resulting model

Overall summary of predictor importance using RSS or Gini index

Record reduction in RSS due to splits in predictor, averaged over all B tres. Large value-→ Important predictor

It in Gini index



Random forests

Rationale

Decorrelate trees

Several trees based on bootstrapped training samples

When splitting, a random sample of m predictors from the set of p is chosen (typically, m=sqrt (p))

Idea: If a very strong predictor, trees too correlated. Averaging highly correlated quantities does not lead to as large variance reduction

On average, (p-m)/p splits do not consider the strong predictor

If m=p, back to bagging

Specially useful with large number of correlated predictors

Random forest

1 Given training data set $\boldsymbol{d} = (\boldsymbol{X}, \boldsymbol{y})$. Fix $m \leq p$ and the number of trees B.

2 For $b = 1, 2, \ldots, B$, do the following.

(a) Create a bootstrap version of the training data $\boldsymbol{d}_b^*$, by randomly sampling the n rows with replacement n times. The sample can be represented by the bootstrap frequency vector $\boldsymbol{w}_b^*$.

(b) Grow a maximal-depth tree $\hat{r}_b(x)$ using the data in $\boldsymbol{d}_b^*$, sampling m of the p features at random prior to making each split.

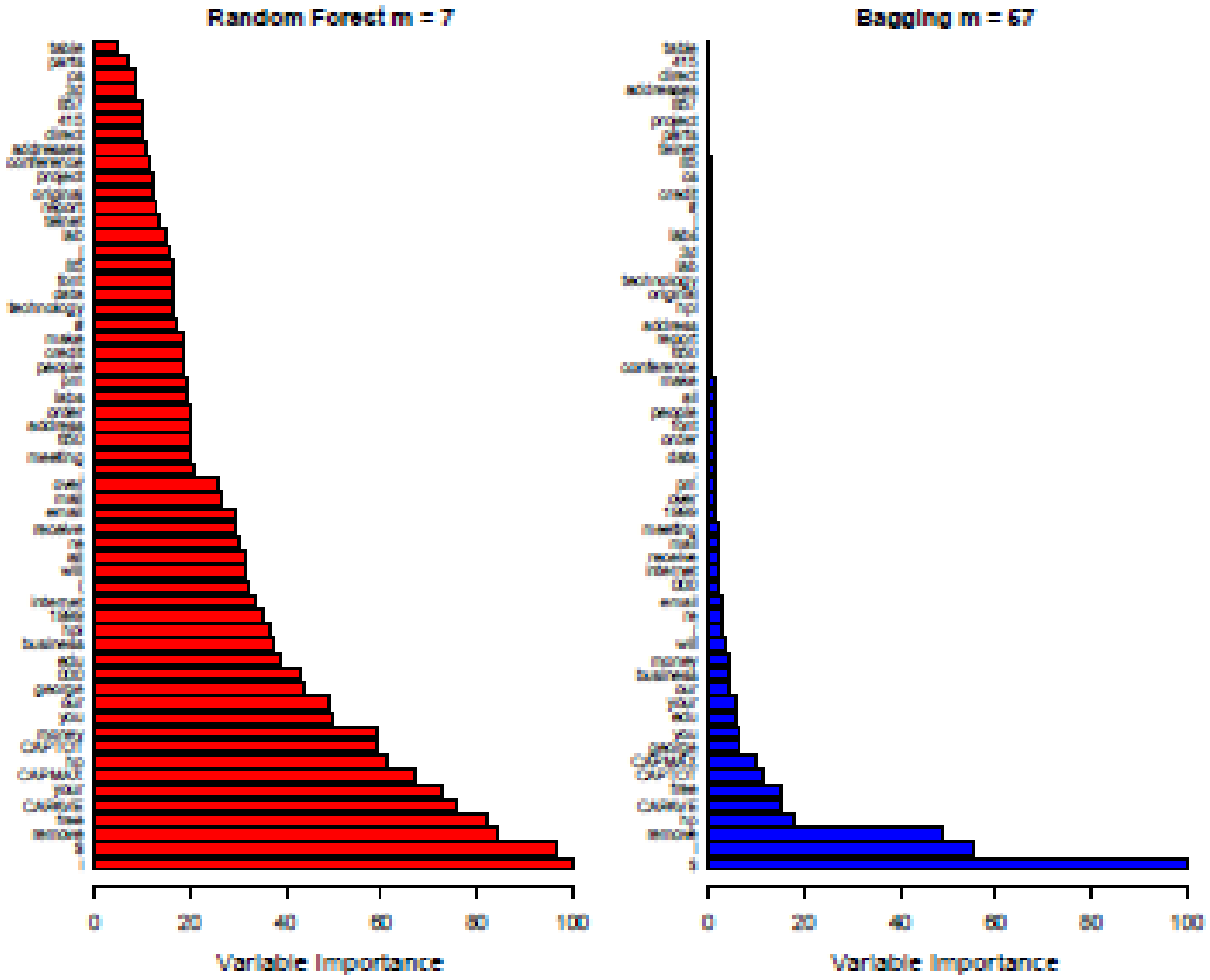
(c) Save the tree, as well as the bootstrap sampling frequencies for each of the training observations.

3 Compute the random-forest fit at any prediction point x_0 as the average

$$\hat{r}_{\mathrm{rf}}(x_0) = \frac{1}{B} \sum_{b=1}^{B} \hat{r}_b(x_0).$$

4 Compute the OOB_i error for each response observation y_i in the training data, by using the fit $\hat{r}_{\mathrm{rf}}^{(i)}$, obtained by averaging only those $\hat{r}_b(x_i)$ for which observation i was *not* in the bootstrap sample. The overall OOB error is the average of these OOB_i.

RF vs bagging



Boosting

Motivation

Like bagging, a generic approach to improve CaR methods, beyond CARTs

Boosting for trees:

A large number of trees is built

Each tree sequentially grown using info from previously grown trees

No bootstrap sampling!! Each tree fit on modifed version of original data set

Rationale

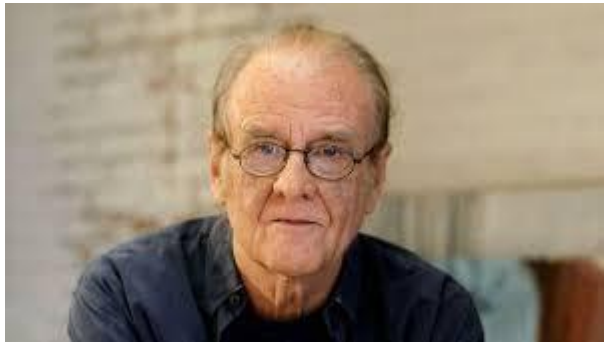
Fitting a single large tree→ Fitting data ‘hard’ potentially overfitting

Boosting learns slowly

Rationale

Fitting a single large tree→ Fitting data ‘hard’ potentially overfitting

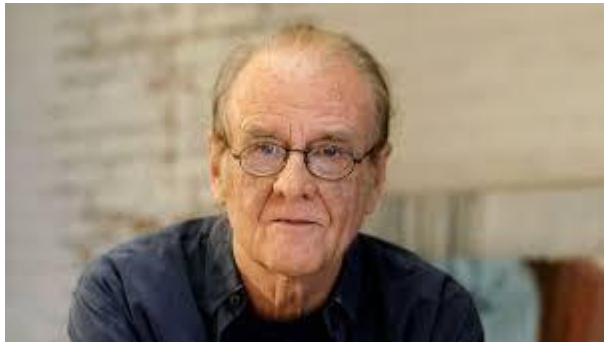
Boosting learns slowly



Rationale

Fitting a single large tree→ Fitting data ‘hard’ potentially overfitting

Boosting learns slowly



https://www.youtube.com/watch?v=SZHUTu5yxYk

Rationale

Fitting a single large tree→ Fitting data ‘hard’ potentially overfitting

Boosting learns slowly:

- Given current model, fit a tree with residuals
- Add new tree to fitted function, update residuals

Trees are small (d terminal nodes). Slowly improve model, where it does not perform well

Shrinkage parameter to further slow down (more and different sahped trees to attack residuals)

Algo: Boosting for regression tree

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d+1$ terminal nodes) to the training data (X, r).

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Boosting. Tuning the parameters

1. B. If too large, overfitting risk. Cross-validation
2. Lambda. Small positive. Learning rate. Between 0.01 and 0.001. Very small entails very large B to achieve good performance
3. d. Controls complexity of boosted ensemble.

   Often d=1 works!!! A stump. Additive model

   d interaction depth, controls interaction order of boosted model

Algo: Gradient boosting

1 Start with $\hat{G}_0(x) = 0$, and set B and the shrinkage parameter $\epsilon > 0$.

2 For $b = 1, \ldots, B$ repeat the following steps.

(a) Compute the pointwise negative gradient of the loss function at the current fit:

$$r_i = -\frac{\partial L(y_i, \lambda_i)}{\partial \lambda_i}\bigg|_{\lambda_i = \hat{G}_{b-1}(x_i)}, \; i = 1, \ldots, n.$$

(b) Approximate the negative gradient by a depth-d tree by solving

$$\underset{\gamma}{\text{minimize}} \sum_{i=1}^{n} (r_i - g(x_i; \gamma))^2 .$$

(c) Update $\hat{G}_b(x) = \hat{G}_{b-1}(x) + \hat{g}_b(x)$, with $\hat{g}_b(x) = \epsilon \cdot g(x; \hat{\gamma}_b)$.

3 Return the sequence $\hat{G}_b(x), \; b = 1, \ldots, B$.

Algo: Adaboost (adaptive boosting)

1 Initialize the observation weights $w_i = 1/n,\ i = 1, \ldots, n$.

2 For $b = 1, \ldots, B$ repeat the following steps.

(a) Fit a classifier $\hat{c}_b(x)$ to the training data, using observation weights w_i.

(b) Compute the weighted misclassification error for $\hat{c}_b$:

$$\text{err}_b = \frac{\sum_{i=1}^{n} w_i I\left[y_i \neq \hat{c}_b(x_i)\right]}{\sum_{i=1}^{n} w_i}.$$

(c) Compute $\alpha_b = \log[(1 - \text{err}_b)/\text{err}_b]$.

(d) Update the weights $w_i \leftarrow w_i \cdot \exp\left(\alpha_b \cdot I\left[y_i \neq c_b(x_i)\right]\right),\ i = 1, \ldots, n$.

3 Output the sequence of functions $\widehat{G}_b(x) = \sum_{\ell=1}^{b} \alpha_m \hat{c}_\ell(x)$ and corresponding classifiers $\widehat{C}_b(x) = \text{sign}\left[\widehat{G}_b(x)\right], b = 1, \ldots, B$.

Final comments

Random forest, frequently good benchmark (with ‘little’ thought)
No free lunches
Boosting and Bagging as general strategies for machine learning

Ensemble methods, Model averaging
Bayesian CART, bayesian model averaging

Graphical models, Neural networks

(Other) decision trees