CEE 616: Probabilistic Machine Learning

M4 Nonparametric Methods: L4D: Trees and Ensemble Methods

Jimi Oke

UMass Amherst

College of Engineering

Tue, Nov 18, 2025

Outline

1 Introduction

2 Regression trees

3 Classification trees

4 Bagging

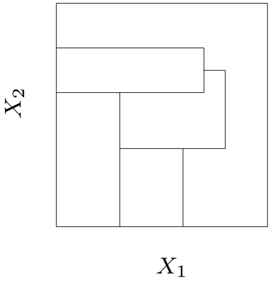
5 Random Forests

6 Boosting

7 Summary

Decision trees

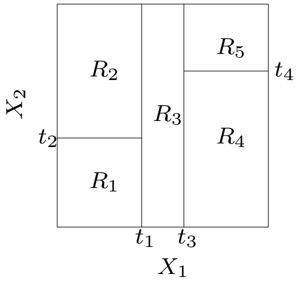
- Consider a regression problem with continuous response y and inputs X_1 and X_2.
- Now, imagine we want to model response y as a constant across different regions of the input space.
- One option can be:



- Can you describe the region using inequalities?
- How would you estimate response y in each region?

Recursive binary partitioning

In order to ensure simplicity and interpretability, we consider splitting a dataset using the **recursive binary partitioning** (RBP) algorithm

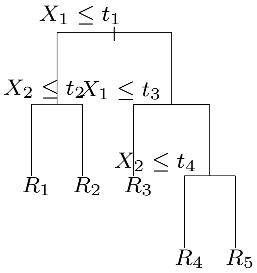


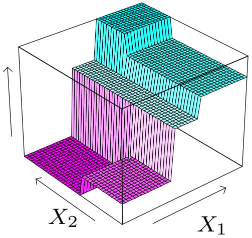
Steps:

1. Split the input space at $X_1 = t_1$; the response is modeled as the mean of y in each region
2. Split the region $X_1 \leq t_1$ at $X_2 = t_2$ and the region $X_1 > t_1$ at $X_1 = t_3$
3. Split the region $X_1 > t_3$ at $X_2 = t_4$

Thus we obtain 5 regions. In each case, the *decision* is to choose the **variable** and the **split-point** that provide the best fit.

Decision trees (cont.)

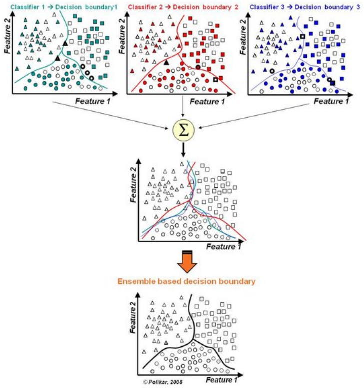




- The partitioning steps can be displayed as a **decision tree**
- The split points are the **internal nodes**
- Lines connecting the nodes are called *branches*
- The final regions are **terminal nodes** or **leaves** (R_1, R_2, R_3, R_4 and R_5 in the figure)

Ensemble learning

Ensemble methods employ various techniques to combine predictions from multiple models. This approach can be termed "learning by committee"


Classifier 1 → Decision boundary1
Classifier 2 → Decision boundary 2
Classifier 3 → Decision boundary 3
Feature 2
Feature 1
Σ
Ensemble based decision boundary
©Polikar, 2008

Source: http://www.scholarpedia.org/article/File:Combining_classifiers2.jpg

We will consider three approaches with a focus on decision trees:

1. **Bagging:** average/majority vote over bootstrap sample
2. **Random Forests:** average/majority vote over collection of de-correlated trees
3. **Boosting:** weighted aggregation of weak learners

Regression tree

Given D features and N observations, how do we predict the response for an observation $\boldsymbol{x}_0$ using a regression tree?

The goal is to partition the input space into M regions using the **recursive binary partitioning** (RBP) approach.

- This is a greedy, top-down approach
- We find the RSS-minimizing split (variable-cutpoint pair) for each subsequent region
- Tree-growing is terminated using a reasonable stopping condition

Recursive binary partitioning decision

For a given jth variable and split point s, we have a pair of half-planes:

$$R_1(j,s) = \{\boldsymbol{x} : \boldsymbol{x}_j < t\} \tag{1}$$
$$R_2(j,s) = \{\boldsymbol{x} : \boldsymbol{x}_j \geq t\} \tag{2}$$

Find splitting variable j^* and point s^* that solve:

$$j^*, s^* = \underset{j,t}{\arg\min} \left[\sum_{x_i \in R_1(j,t)} (y_i - \hat{y}_{R_1})^2 + \sum_{x_i \in R_2(j,t)} (y_i - \hat{y}_{R_2})^2 \right] \tag{3}$$

where

$$\hat{y}_{R_1} = \text{ave}(y_i : \boldsymbol{x}_i \in R_1(j,t)) \tag{4}$$
$$\hat{y}_{R_2} = \text{ave}(y_i : \boldsymbol{x}_i \in R_2(j,t)) \tag{5}$$

Overfitting and pruning

- Earlier, we mentioned tree partitioning ends when the appropriate terminating condition is met
  - Traditional thresholding can be myopic, as a potentially better solutions could be discarded.
- Also, we want to avoid excessively large trees, as these are prone to overfitting
- How then can we guarantee the best result?

Strategy

- Grow a large tree T_0 to a minimum specified node size
- Use *cost-complexity pruning* to prune T_0 to find an acceptable subtree $T \subset T_0$

Cost-complexity pruning

Let m be the index of terminal nodes (recall that each node represents a region). Then let:

$$\begin{aligned} N_m &= \sum_{i=1}^{N} \mathbb{I}(\boldsymbol{x}_i \in R_m) && (6) \\ \hat{y}_m &= \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} y_i && (7) \\ Q_m(T) &= \frac{1}{N_m} \sum_{\boldsymbol{x}_i \in R_m} (y_i - \hat{y}_m)^2 \quad \text{(a measure of impurity)} && (8) \end{aligned}$$

The **cost-complexity criterion** is then given by:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha |T| \quad (9)$$

where $|T|$ is the number of terminal nodes in T and $\alpha \geq 0$ is a tuning parameter.

Cost-complexity pruning (cont.)

Using the cost-complexity criterion, we find subtree $T_\alpha \subseteq T_0$ that minimizes $C_\alpha(T)$.

- $\alpha|T|$ is a complexity penalty term
- If $\alpha = 0$, then $T_\alpha = T_0$
- Larger α values result in smaller trees T_α
- We find T_α using a procedure known as *weakest link pruning*[1]

Your final question should be, how do we find the optimal $\hat{\alpha}$?

Answer

Cross-validation (k-fold)

[1] Consult ESL p. 308 for further details on this algorithm.

Classification trees

- In regression trees, the predicted response for an observation in a certain region (terminal node) is given by the *mean response* of observations at that node
- In classfication trees, the *most commonly occurring class* is the predicted response
- To make the binary splits in classification trees, we use a suitable impurity measure in place of the RSS

The proportion of class c observations in node m is given by:

$$\hat{p}_{mc} = \frac{1}{N_m} \sum_{\mathbf{x}_i \in R_m} \mathbb{I}(y_i = c) \tag{10}$$

We then classify observations in node m to the class with maximum representation:

$$c(m) = \arg\max_k \hat{p}_{mc} \tag{11}$$

Node impurity measures

The recursive binary partitioning splits the observations into regions R_m to minimize **node impurity** at m, measured by:

- **Misclassification error**

$$E = \frac{1}{N_m} \sum_{i \in R_m} \mathbb{I}(y_i \neq c(m)) = 1 - \hat{p}_{mc(m)} \tag{12}$$

  where $ck(m) = \arg\max_k \hat{p}_{mc}$

- **Gini index**

$$G = \sum_{c \neq c'} \hat{p}_{mc}\hat{p}_{mc'} = \sum_{c=1}^{C} \hat{p}_{mc}(1 - \hat{p}_{mc}) \tag{13}$$

- **Cross-entropy (deviance)**

$$D = -\sum_{c=1}^{C} \hat{p}_{mc} \log \hat{p}_{mc} \tag{14}$$

Node impurity measures (cont.)

- The Gini index and the cross-entropy are more sensitive to changes in node probabilities
- They are also continuously differentiable can be readily optimized

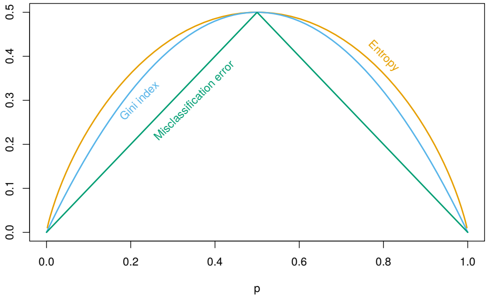


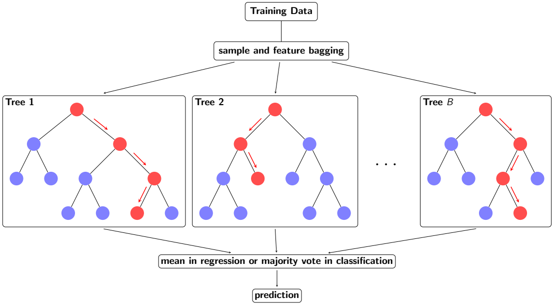
Figure: Various impurity measures plotted against p in the two-class case

Typically, the misclassification error is used in pruning, while the others are used in growing the tree

BAGGING: Bootstrap AGGregatING

Consider a training data set $\mathcal{D} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_n, y_n)\}$. Let a fitted decision tree be given by $\hat{f}(\boldsymbol{x})$.

- Predictions have low bias but high variance (trees are unstable)
- Obtaining predictions from trees fitted to B bootstrap samples reduces variance (**bagging**)


Training Data
sample and feature bagging
Tree 1
Tree 2
Tree B
. . .
mean in regression or majority vote in classification
prediction

Bagging algorithm

1. Sample with replacement from $\mathcal{D}$ to generate B bootstrap samples, $\{\mathcal{D}^{*b} : b = 11, 2, \ldots, B\}$
2. Fit a decision tree to each $\mathcal{D}^{*b}$ to obtain prediction $\hat{f}^{*b}(\boldsymbol{x})$
3. Obtain the **bagging estimate**:
   a. Regression: Average the predictions

$$\hat{f}_{\text{bag}}(\boldsymbol{x}) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(\boldsymbol{x}) \tag{15}$$

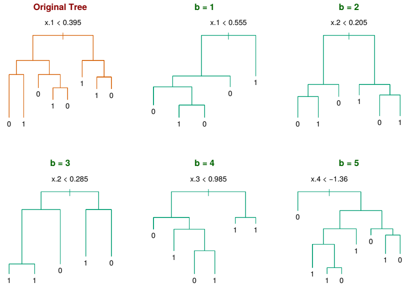
   b. Classification: majority vote

$$\hat{f}_{\text{bag}}(\boldsymbol{x}) = \arg\max_{c} \sum_{b=1}^{B} \mathbb{I}(\hat{f}^{*b}(\boldsymbol{x}) = c) \tag{16}$$

Example 1: Bagging trees on simulated dataset

Given a dataset of $n = 30$ observations with $p = 5$ predictors

- Pairwise correlation for each predictor is 0.95
- 200 bootstrap samples are created and unpruned decision trees are fitted
- High correlations leads to high variance in individual decision tree estimates



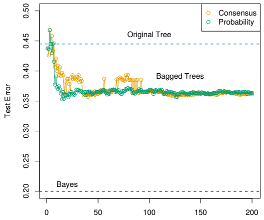


Figure: (L): Fitted trees on 6 bootstramp samples. (R): Test errors.

Out-of-bag error

Bootstrap samples only contain 63% of the unique observations on average. Why?

- Observations left out of bootstrap sample are **out-of-bag** (OOB) instances
- Errors can be computed on each OOB instance per bootstrap sample
- These are then averaged to obtain a validation error estimate for use in hyperparameter selection

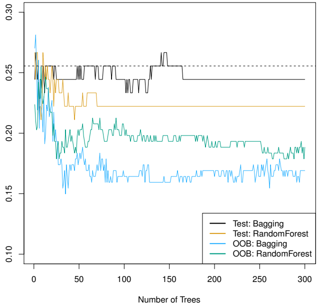


Figure: Comparing test (validation set approach) and OOB (similar to cross-validation) errors on bagging and random forest estimates on the `Heart` dataset (survival of heart transplant patients).

Feature importance

Bagging provides stability (reduced variance) at the expense of simplicity (interpretability).

- To aid inference, we compute **feature importances** of bagged tree predictors:

$$fi_d(T) = \sum_{m=1}^{M-1} G_m \mathbb{I}(t_m = d) \tag{17}$$

  where
  - $fi_d(T)$ is the importance of feature d in tree T
  - G_m is the gain in accuracy (reduction in loss) at node m
  - $j_m = d$ if node m uses feature d as its splitting variable t
- These are then averaged over all B trees (estimators)
- Regression trees: total amount of RSS decreases due to splits
- Classification trees: Gini index used

Random forests

Invented by Leo Breiman in 2001, **random forests** build on bagging by fitting *decorrelated* trees on the bootstrap samples.

- Implemented by splitting the observations by considering only a *random* subset $S \subset D$ of predictors.
- Helps to further reduce variance when predicted values are highly correlated across samples (due to correlated predictors).
- In bagging, $S = D$.
- Recommended values of S: $S = \left\lfloor \frac{D}{3} \right\rfloor$ (regression); $S = \sqrt{D}$ (classification)

Algorithm

```
1. For $b = 1$ to $B$:
    a. Draw a bootstrap sample $\mathcal{D}^*$ of size $N$ from training dataset
    b. Grow a random-forest tree $T_b$ to fit the sample by RBP as follows until
       minimum node size is reached:
        i. Randomly select subset $S_m$ of the $D$ predictors at node $m$
        ii. Pick the best predictor/split-point $(j, t)$ from $S_m$
        iii. Split node $m$ into two daughter nodes
2. Output the ensemble of trees $\{T_b\}_1^B$
```

Feature importance in random forests

- Since the feature subset in each step of partitioning is random in all the B trees, the standard variable importance approach is not effective in random forests
- Instead, OOB samples are used to determine the variable importance
- This is done by perturbing each variable j in the OOB samples and then averaging the decrease in accuracy across the trees
- Illustrated on the `spam` dataset:

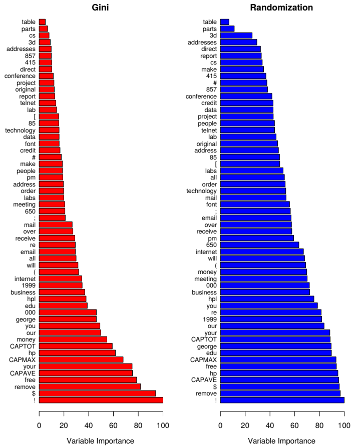


Figure: Variable importance computed on a scale of 0-100 from (Left) gradient boosted model (via Gini index) and (Right) random forest model (via OOB). Rankings are comparable, but relative values are not.

Boosting

Generally, boosting refers to the approach of sequentially weighting the predictions of weak learners to obtain a final prediction
It can be considered as a method of fitting an additive model:

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{m=1}^{M} \beta_m b_m(\boldsymbol{x};\boldsymbol{\theta}_m) \tag{18}$$

where:

- β_m are the expansion coefficients (weights)
- $b_m(\boldsymbol{x};\boldsymbol{\theta}_m)$ are elementary (simple) basis functions of $\boldsymbol{x}$ characterized by a set of parameters $\boldsymbol{\theta}_m$, e.g.
  - Shallow decision trees: $\boldsymbol{\theta}_m$ parameterizes the split variable/split-point (j, t) at the internal nodes and the predictions at the terminal nodes
  - Single-hidden-layer neural networks: $b(\boldsymbol{x};\boldsymbol{\theta}_m) = \boldsymbol{\sigma}(\boldsymbol{w}^T\boldsymbol{x})$, where $\boldsymbol{\sigma}(t) = 1/(1+e^{-t})$

Boosting (cont.)

- To fit such additive models, we minimize the sum of the loss functions across the basis expansions.

$$\min_{\{\beta_m,\boldsymbol{\theta}_m\}_1^M} \sum_{i=1}^{N} \ell\left(y_i, \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}_i; \boldsymbol{\theta}_m)\right) \tag{19}$$

- Choices for ℓ include: squared error, likelihood-based, exponential (Adaboost)
- Fitting can be efficiently done via the *forward stagewise additive modeling* algorithm[2]

[2]ESL 10.1–10.5

Forward stagewise additive modeling

1. Initialize $f_0(\boldsymbol{x}) = 0$
2. For $m = 1$ to M:
   a. Find:

   $$(\beta_m, \boldsymbol{\theta}_m) = \arg\min_{\beta, \boldsymbol{\theta}} \sum_{i=1}^{M} \ell(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i; \boldsymbol{\theta}))$$

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

If ℓ is specified as squared error (typical for regression), then:

$$\begin{aligned} \ell(y, f(\boldsymbol{x})) &= (y - f(\boldsymbol{x}))^2 && (20) \\ \ell(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i; \boldsymbol{\theta})) &= (y_i - f_{m-1}(\boldsymbol{x}_i) - \beta b(\boldsymbol{x}_i; \boldsymbol{\theta}))^2 && (21) \\ &= (r_{im} - \beta b(\boldsymbol{x}_i, \boldsymbol{\theta}))^2 && (22) \end{aligned}$$

where r_{im} is the **residual** of model m on the ith observation.

Boosting trees

Boosting can be used as a “slow learning” approach for decision trees

- First we generate B bootstrap samples
- Starting with an initial tree, we fit a subsequent tree to the residuals from the first model
- We then update the tree and its residuals using a fraction λ of the predictions of the new tree
- The estimation and residual update step is repeated until the Bth sample is used
- The boosted model is then given by the weighted sum of the fitted tree estimates

Boosted regression tree algorithm

1. Set $\hat{f}(\boldsymbol{x}) = 0$ and $r_i = y_i$ for all i in training dataset
2. For $b = 1, 2, \ldots, B$, repeat:
   a. Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to training data $(\boldsymbol{x}, r)$
   b. Update $\hat{f}$ by adding in a shrunken version of new tree:

$$\hat{f}(\boldsymbol{x}) \leftarrow \hat{f}(\boldsymbol{x}) + \lambda \hat{f}^b(\boldsymbol{x}) \tag{23}$$

   c. Update residuals:

$$r_i \leftarrow r_i - \lambda \hat{f}^b(\boldsymbol{x}_i) \tag{24}$$

3. Output boosted model:

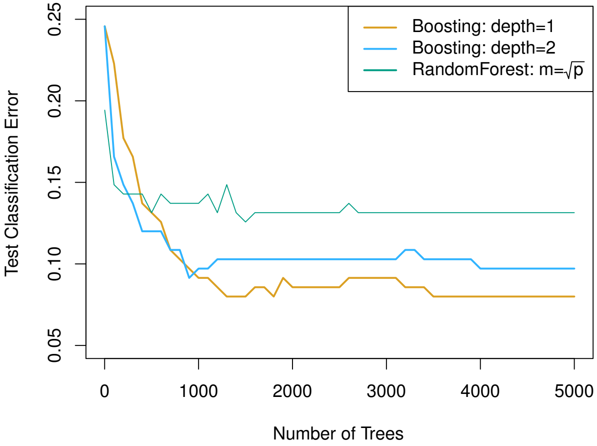
$$\hat{f}(\boldsymbol{x}) = \sum_{b=1}^{B} \lambda \hat{f}^b(\boldsymbol{x}) \tag{25}$$

Notes on boosting trees

- There are three tuning parameters to consider:
  - Number of trees B. If B is too large, model is prone to overfitting. We choose B by cross-validation
  - Shrinkage/learning parameter $\lambda > 0$ (typically 0.01 or 0.001)
  - Number of splits d in each tree (interaction depth); $d = 1$ is a typical choice (in this case, the trees are referred to as "stumps")
- The sequential fitting of $\hat{f}(\boldsymbol{x})$ is analogous to *gradient descent* (since we are minimizing a loss function in each split)
  - Thus, r_i can be updated via the gradient of the loss function
  - Hence, the procedure is referred to as **gradient boosting**
  - The popular modeling package `XGBoost` (Extreme Gradient Boosting) is based on this idea but introduces regularization and second-order gradients

Boosting perfomance

Boosting typically outperforms random forests and depth-1 trees perform better than other choices for d.



Further issues and topics

- The methods described in this lecture are generally referred to as the CART implementation (classification and regression tree).
- Other approaches exist, e.g. PRIM (Patient Rule Induction Method) and ID3 (and its successors)

Potential issues with trees and mitigations

- **Multilevel categorical variables:** the greater the number of levels, the more prone to the tree is to overfitting
- **Missing data:** dropping incomplete observations can reduce accuracy; imputation methods can be used
- **Tree instability:** errors or changes to data can drastically affect results *Bagging* can be used to address this (next lecture)
- **Lack of smoothness:** can be addressed using MARS (Multivariate Adaptive Regression Splines)

Reading

- **PMLI** 18
- **ESL** Section 9.2
- ISLR 8.3
- Bagging: ESL 8.7
- Random Forests: ESL 15
- Boosting: ESL 10 (Very dense chapter. You may want to focus on 10.1–10.3, 10.7, 10.10, 10.13, 10.14.)

Further reading/study:

- **PRIM**: ESL 9.3 and Friedman's paper at `http://statweb.stanford.edu/~jhf/ftp/prim.pdf`.
  PRIM/CART can be used in scenario discovery. For a brief overview, see Jan Kwakkel's blog **post**.
- **MARS**: ESL 9.4. This article provides a readily digestible overview: `http://uc-r.github.io/mars`.