# Week 13: Tree-based Methods for Classification

MATH-517 Statistical Computation and Visualization

Linda Mhalla

2024-12-13

## What is Classification?

Given data on predictor variables (covariates/features) $X \in \mathbb{R}^p$ and a **categorical response variable** $Y \in \{0, \dots, J-1\}$, build a model for

- predicting the value of the response (class) from the predictors
- understanding the relationship between predictors and the response

$\Rightarrow$ it is a supervised learning

**Examples**:

- $X$: diagnostic measurements and $Y$: presence/absence of disease
- $X$: credit score, age, marital status and $Y$: loan defaults (yes/no)

**Classification Methods**:

- Linear discriminant analysis (1930')
- Logistic regression (1944)
- Nearest neighbors classifiers (1951)

## What is Classification?

Given data which are realizations from

$$(X_1, Y_1), \dots, (X_N, Y_N) \quad \text{i.i.d.},$$

the goal is to assign probabilities

$$\pi_k(x) = P(Y = k \mid X = x), \quad \text{for } k = 0, \dots, J-1$$

where $x$ can be a newly observed predictor (prediction)

$\Rightarrow$ similar to the regression function $m(x) = \mathbb{E}(Y \mid X = x)$

## The Bayes Classifier

- A classifier $\mathcal{C} : \mathbb{R}^p \to \{0, \dots, J-1\}$ assigns to a predictor $X$ a class, i.e., its prediction for the corresponding $Y$
- The quality of a classifier can be measured by the expected 0-1 loss

$$P\{\mathcal{C}(X_{new}) \neq Y_{new}\}$$

- The optimal classifier wrt this loss is the **Bayes classifier**

$$\mathcal{C}_{Bayes}(x) = \arg\max_{0 \leq k \leq J-1} \pi_k(x)$$

$\Rightarrow$ the lowest risk is obtained by classifying $x$ to the most probable class

In practice, $\pi_k(\cdot)$ (depends on the joint df of $(X, Y)$) needs to be estimated and plugged into the classifier $\mathcal{C}_{Bayes}$

Let's estimate it non-parametrically while imposing some structural assumptions

## Tree-based Methods

Predict $y$ from a feature vector $x \in \mathbb{R}^p$ by dividing the feature space into (non-overlapping) rectangles $A_1, \dots, A_m$

$\Rightarrow$ works if $y$ is discrete (classification) or continuous (regression)

Rectangles can be achieved by making successive binary splits on the predictors $X_1, \dots X_p$

- choose a variable $X_j$, $j = 1, \dots p$
- divide up the feature space according to

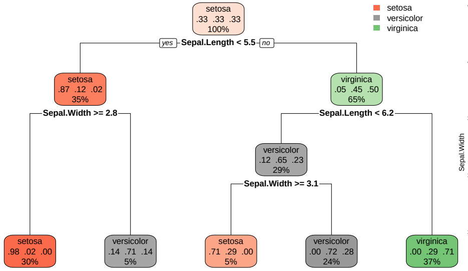
$$X_j \leq s \text{ and } X_j > s$$

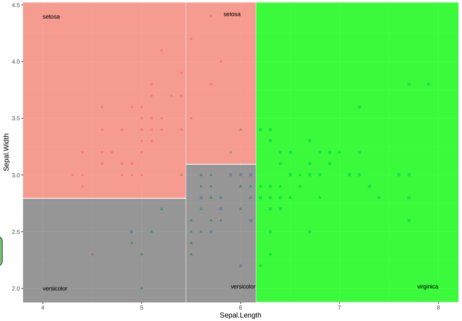
- proceed in each half

**Questions**: How to choose the splits? When to stop growing the tree?

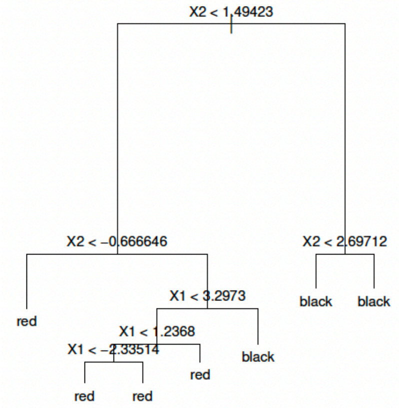
# Classification Tree: Example

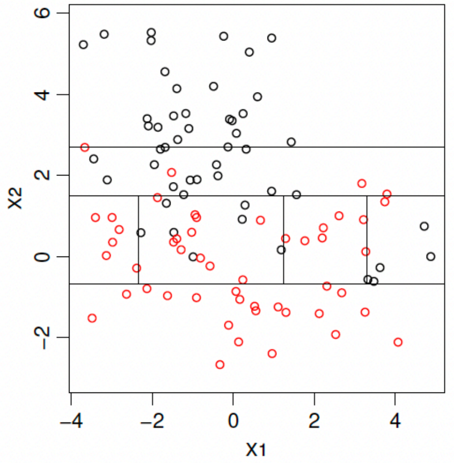
The iris dataset with four features (petal/sepal length and width) and three species





## Classification Tree: Simulated Example





## Terminology

- each split is called a **node**
- a terminal node is called a **leaf**
- interior nodes lead to **branches**

## Classification Trees

Classification trees are popular because they are interpretable and (perhaps) mimic the way (some) decisions are made

A classification tree can be thought of as defining $m$ regions (rectangles) $A_1, \dots A_M$, each corresponding to a leaf of the tree

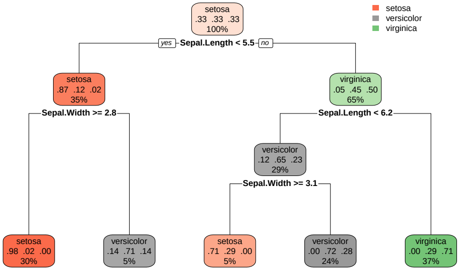
- each $A_m$ is assigned a class label $c_m \in \{0, \dots J-1\}$ by **majority vote** (the most common class in that region)
- then a new point $x_+ \in \mathbb{R}^p$ is classified by

$$T(x_+) = \sum_{m=1}^{M} c_m \cdot \mathbb{I}_{\{x_+ \in A_m\}} = c_m \text{ such that } x_+ \in A_m \subset \mathbb{R}^p$$

Finding out which region a given point $x$ belongs to is easy since the regions $A_m$ are defined by a tree: just scan down the tree

## Classification Trees

- Perform tests (sequentially) on the attributes of $x$
- Follow the branch that corresponds to the outcome of the tests
- Repeat until you reach a leaf node
- Predict the label of $x$ to be that of that leaf node



**Tricky part**: get a data-driven estimate of the partition: splitting variables? split points?

## Predicted Class Probabilities

We can get the predicted class for new points, but also the **predicted class probability**

For each class $k = 0, \dots, J-1$, we can estimate the probability that the class label is $k$ given that the feature vector lies in region $A_m$, $P(Y = k \mid X \in A_m)$ by

$$p_{mk} = \hat{p}_k(A_m) = \frac{1}{n_m} \sum_{x_i \in A_m} \mathbb{I}_{\{y_i = k\}}$$

the proportion of points in the region $A_m$ that are of class $k$, where $n_m = \#\{(x_i, y_i) \mid x_i \in A_m\}$

The predicted class (by majority vote) can be expressed as

$$c_m = \underset{k=0,\dots,J-1}{\arg\max}\, p_{mk}$$

## How to Grow a Tree?

The **CART algorithm** [1] estimates the tree model

$$T(x) = \sum_{m=1}^{M} c_m \cdot \mathbb{I}_{\{x \in A_m\}}$$

using a greedy approach (local optimality/stage) based on binary splits

Starting at the top, for each coordinate $j \in \{1, \dots, p\}$ we look for the best binary split defining

$$A_1(j,s) = \left\{x \in \mathbb{R}^p : x_j \leq s\right\} \quad \text{and} \quad A_2(j,s) = \left\{x \in \mathbb{R}^p : x_j > s\right\}$$

$\Rightarrow$ The values of $j \in \{1, \dots, p\}$ and $s \in \mathbb{R}$ are found by minimizing

$$\min_{j,s} \{Q_1(T) + Q_2(T)\}$$

where $Q_m(T)$ is a **node impurity measure** (loss function)

[1] Breiman et al. (1984), "Classification and Regression Trees"

## Node Impurity Measures for Classification

Recall that $p_{mk}$ is the proportion of training observations in $A_m$ that are from class $k$

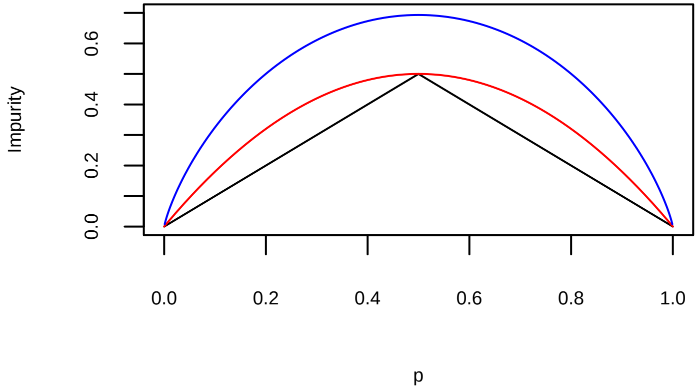
- misclassification error: $\frac{1}{n_m}\sum_{i:x_i\in A_m}\mathbb{I}_{\{y_i\neq c_m\}} = 1 - p_{mc_m}$
- Gini index: $\sum_{k\neq k'} p_{mk}p_{mk'} = \sum_{k=0}^{J-1} p_{mk}(1-p_{mk})$
- Cross-entropy (or deviance): $-\sum_{k=0}^{J-1} p_{mk}\log(p_{mk})$

For two classes ($J=2$)

- misclassification error: $1-\max(p, 1-p)$ (black)
  - is non-differentiable (bad for numerical optimization)
- Gini index: $2p(1-p)$ (red)
- Cross-entropy (or deviance): $-p\log(p)-(1-p)\log(1-p)$ (blue)

Growing a tree is based on either the Gini index or cross-entropy

## Why not minimize the misclassification error?



- Gini index and cross-entropy are more sensitive to small changes: going from $p = 0.8$ to $p = 0.9$ is better than going from $p = 0.5$ to $p = 0.6$ (these are equal changes for the misclassification error)

$\Rightarrow$ the Gini index and the cross-entropy will favour pure nodes with $p_{mk} \approx 0$ or $p_{mk} \approx 1$

## How large should we grow the tree?

- very large tree might overfit the data
- small tree might not capture the important structure

$\Rightarrow$ Tree size is a tuning parameter reflecting the model's complexity

**Pruning**:

- built a large tree $T_0$, stopping only when the number of observations in each leaf is small (for ex. 5)
- prune this large tree, i.e., collapse some of its leaves into the parent nodes (backward elimination)

**Alternative to pruning**: grid search for the optimal maximal depth of the tree by cross-validation (minimizing the misclassification rate)

## Pruning, by how much?

For any subtree $T \subset T_0$ that can be obtained by pruning $T_0$, we define the **cost-complexity pruning**:

$$C_\lambda(T) = \mathrm{err}_T + \lambda|T|, \quad \lambda \geq 0$$

where $|T| = \#$ leaves in $T$ and $\mathrm{err}_T$ is the misclassification rate

For a fixed value of $\lambda$, we need to find the tree $T_\lambda$ minimizing $C_\lambda(T)$

$\rightarrow$ done efficiently by slowly pruning the tree, i.e., constructing the sequence of pruned trees that slowly increase the misclassification rate

- successively delete the terminal node in the fully grown tree that yields the smallest increase of the misclassification rate. This yields a sequence of subtrees that must contain $T_\lambda$

**Choice of** $\lambda$ : trade-off between goodness-of-fit and complexity

- a larger size means smaller bias and high variance
- a smaller tree means larger bias and smaller variance

$\Rightarrow$ the value of $\lambda$ will be chosen by $K$-fold $\mathrm{CV}$ error rates

## Pruning details

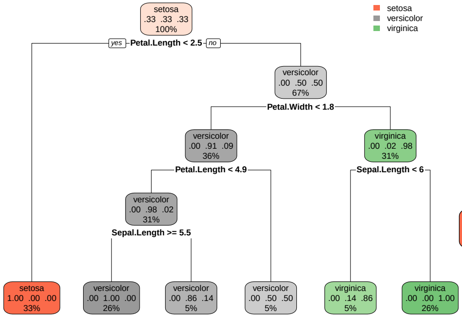
1. Use recursive binary splitting (e.g., using Gini index) to grow a large tree on the training data, stopping only when each terminal node has fewer than some minimum number of observations
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of $\lambda$
3. Use $K$-fold CV to choose $\lambda$. For each $k = 1, \dots, K$:
    1. Repeat Steps 1 and 2 on the $\frac{K-1}{K}$-th fraction of the training data, excluding the $k$-th fold
    2. Evaluate the error rate on the data in the left-out $k$-th fold, as a function of $\lambda$

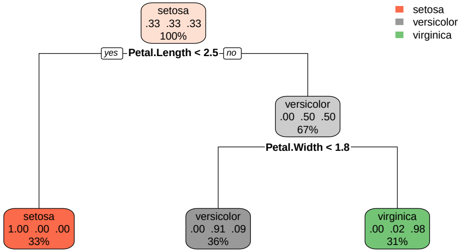
   Average the results, and pick $\lambda$ to minimize the average error
4. Return the subtree from Step 2 that corresponds to the chosen value of $\lambda$

## Pruning: Example

Left: fully grown classification tree (using Gini index)

Right: pruned tree found by CV (using misclassification error)





## Questions

- Are there infinitely many splits to consider when growing?
    - No, as the split points $s$ are from the set of mid-points between observed values

## Questions

- Are there infinitely many splits to consider when growing?
    - No, as the split points $s$ are from the set of mid-points between observed values
- Pros of classification trees?
    - variable selection done automatically (part of the split selection)
    - missing values are dealt with by "surrogate splits" (exploit correlations between covariates)
    - model free and easy to interpret
    - able to handle both numerical and categorical data
    - qualitative covariates are easily handled

## Questions

- Are there infinitely many splits to consider when growing?
  - No, as the split points $s$ are from the set of mid-points between observed values
- Pros of classification trees?
  - variable selection done automatically (part of the split selection)
  - missing values are dealt with by "surrogate splits" (exploit correlations between covariates)
  - model free and easy to interpret
  - able to handle both numerical and categorical data
  - qualitative covariates are easily handled
- Cons of classification trees?
  - rely on a greedy search (local optimal decisions) $\Rightarrow$ no guarantee to return globally optimal tree
  - classification accuracy is not great
  - tend to have high variance: small changes in the training data can produce big changes in the estimated tree
    - $\rightarrow$ this can be fixed if we are willing to give up interpretability

## How to Fix This?

- Let's think back to CV, and why it gives much better results than the validation set approach
- Validation set: if you pick a different random split, you can get wildly different estimates of test error
- $K$-fold CV produces much more stable error estimates by averaging over $K$ separate estimates of error
- The idea of **Bagging (Bootstrap AGGregatING)** has a similar motivation: to decrease the variance of a high-variance estimator, we can average across a bunch of estimators

## Bagging [2]

For a model $\hat{f} : x \mapsto \hat{f}(x) = \hat{y}$, e.g., $\hat{f} = \widehat{T}$

- resample the training data $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ to create $B$ artificial datasets $\mathcal{D}^{(b)} = \left\{ (x_i, y_i)^{(b)} \right\}$
  - $\mathcal{D}^{(b)}$ might have the same size $N$ (sample with replacement: bootstraping)
  - or $\mathcal{D}^{(b)}$ might be smaller than $N$ (sample without replacement: subsampling)
- train a model $\hat{f}^{(b)}$ on each $\mathcal{D}^{(b)}$
- perform **bagging**: "aggregate" the models $\left\{ \hat{f}^{(b)} \right\}$, i.e., for an input $x_+$, predict by majority vote:

$$\hat{y}_+ = \arg\max_k \# \left\{ \hat{f}^{(b)} \left( x_+ \right) = k \right\}$$

[2]Breiman (1996) "Bagging predictors"

## Bagging

**Bias-variance tradeoff of bagging**:

- typically reduces variance
  - $f^{(b)}$ are dependent: if they are highly correlated then the variance reduction will be small
- typically increases bias
- generally, the increase in bias is smaller than the reduction in variance

## Bagging Trees

**Bagging tree** algorithm:

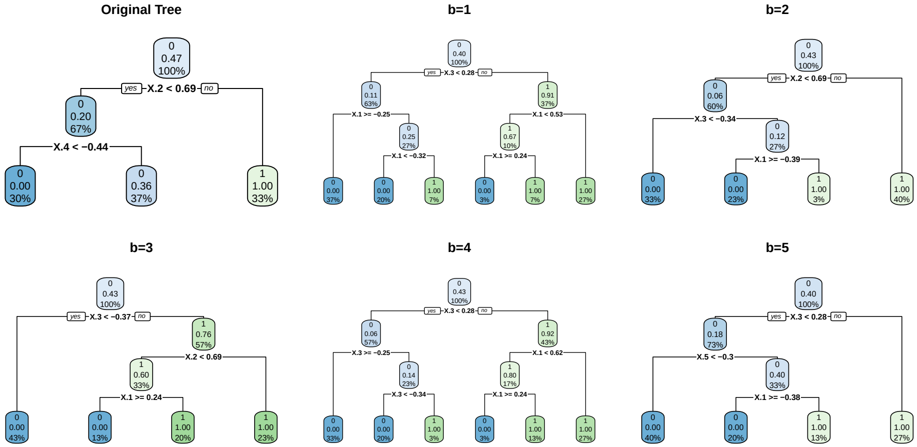
- choose $B$ large (usually 500)
- for $b = 1, \dots, B$, fit **unpruned trees** $\widehat{T}^{(b)}$ to the $b$th bootstrap sample (or subsample)
- "aggregate" the trees $\left\{\widehat{T}^{(b)}\right\}$, i.e., for an input $x_+$, predict by majority vote (from the $B$ trees)

$$\hat{y}_+ = \arg\max_k \# \left\{\widehat{T}^{(b)}\left(x_+\right) = k\right\}$$

Why does it work well?

- each unpruned tree has low bias but high variance
- the correlation between the trees is typically small when using bootstrap samples

# Bagging Trees: Example



## Predicted Class Probabilities?

**Aim**: probability estimate $\hat{\pi}_k(x)$ from the bagging tree

- we can consider the proportion of bootstrapped trees that voted for class $k$

$$\hat{\pi}_k^{vote}(x) = \frac{1}{B} \sum_{b=1}^{B} \{\widehat{T}^{(b)}(x) = k\}$$

$\Rightarrow$ bad idea...

Suppose we have two classes, and the true probability that $y_0 = 1$ when $X = x_0$ is 0.75

Suppose each of the bagged trees correctly classifies $x_0$ to class $1$

$\Rightarrow \hat{\pi}_1^{vote}(x_0) = 1$, which is wrong!

Instead, we can use each tree's predicted class probabilities: probability bagging

## Predicted Class Probabilities?

Instead of just looking at the class predicted by each tree (the classification itself), look at the predicted class probabilities $\hat{\pi}_k^{(b)}(x)$

- Define the bagging estimate of class probabilities:

$$\hat{\pi}_k^{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{\pi}_k^{(b)}(x) \quad k = 0, \dots J-1$$

- Given an input vector $x_0$, we can classify it according to

$$\hat{y}_0^{\text{bag}} = \underset{k=0,\dots J-1}{\arg\max}\, \hat{\pi}_k^{\text{bag}}(x)$$

$\Rightarrow$ preferred if we want to estimate class probabilities, and it may improve overall classification accuracy (compared to majority vote)

## Predicted Class Probabilities?

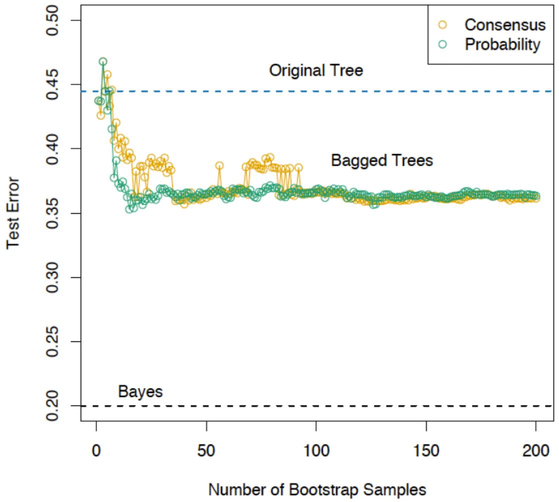


Figure 8.10 from ESL

## Bagging: Summary

There are two strategies for aggregating predictions

- taking the class with the majority vote
- evraging the estimated class probabilities and then voting
- reduces the variance of the base learner
- is most effective if we use strong base learners with little bias but high variance
- bagging a good classifier can improve predictive accuracy, but bagging a bad one can seriously degrade predictive accuracy
- the final bagged classifier is not a tree $\rightarrow$ we lose interpretability
- increased computational complexity

## Improvements? Random Forests (Breiman, 2001)

Random forests extend bagging by incorporating a small tweak

$\Rightarrow$ decrease correlations of bagged trees by making them “more random”

$\Rightarrow$ decreases the variance

**Random forest** algorithm:

- bootstrap the data $B$ times
- to grow a bagged tree, before performing each split, randomly select $m$ of the $p$ variables to be used for the split
    - the subset of variables changes at each split
    - grow full, unpruned trees
- for prediction: majority vote from the $B$ trees

## Random Forests

**Intuition**: if one variable is much more important than the others then all bagged trees will select this variable for the first split, making these trees similar (hence correlated). Selecting a random subset of $m$ variables for each split avoids this!

**Choice of** $m$: $m = \lfloor \sqrt{p} \rfloor$ for classification seems to work well in practice

- e.g., if we have 100 predictors, each split will be allowed to choose 10 randomly selected predictors

**Note**: bagging is a special case of random forests with $m = p$

## Pros and Cons of Random Forests

- **Pros**:
  - great predictive performance
  - stable: small change in the data might change the individual trees but the forest is relatively stable
  - almost no tuning required
  - out-of-bag (oob) error estimates (no CV)
    - use the $e^{-1}\% \approx 37\%$ data not selected in the $b$th bootstrap sample to estimate the prediction error from the $b$th tree
    - can be shown to be equivalent to CV
  - variable importance
    - compute the importance of the $j$th variable $X_j$ by randomly shuffling its values for the oob data and then measuring the increase in prediction error/decrease in accuracy
    - the higher the increase, the most important is the variable
- **Cons**:
  - lose the interpretability of a single tree

## Final Thoughts

- Bagging
  - improves the prediction accuracy for high variance (and low bias) models (such as classification trees) at the expense of interpretability and computational speed
  - consists of independent processes ⇒ algorithm is easily parallelizable
  - results in (very) correlated trees ⇒ variance reduction is limited
- Random Forests
  - decrease the correlation between bagged trees by considering a random subset of the features/predictors/covariates

  ⇒ faster than bagging
  - little theory but consistency was proved and a method to obtain CI was proposed
- We didn't discuss **Boosting** that builds up the ensemble sequentially
  - e.g., to boost trees, we grow small trees, one at a time, at each step trying to improve the model fit in places we've done poorly so far
  - still lose interpretability but like RF and bagging, captures complex structures in the data (vs additive models, e.g., logistic regression)

## References

- T. Hastie, R. Tibshirani and J. Friedman (2008) The Elements of Statistical Learning (2nd Edition)
- G. James, D. Witten, T. Hastie and R. Tibshirani (2013) An Introduction to Statistical Learning, with applications in *R*