ECON3389 Machine Learning in Economics

Module 5
Decision trees

Alberto Cappello

Department of Economics, Boston College

Fall 2024

Overview

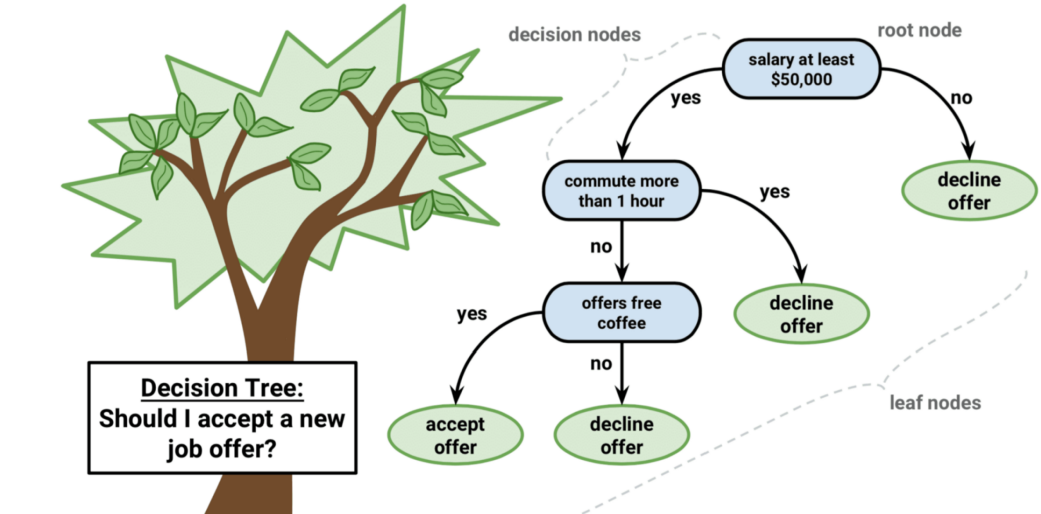
Agenda:

- Decision trees
- Bagging, Random Forests and Boosting

Readings:

- ISLR Chapter 8

Example: Job offer

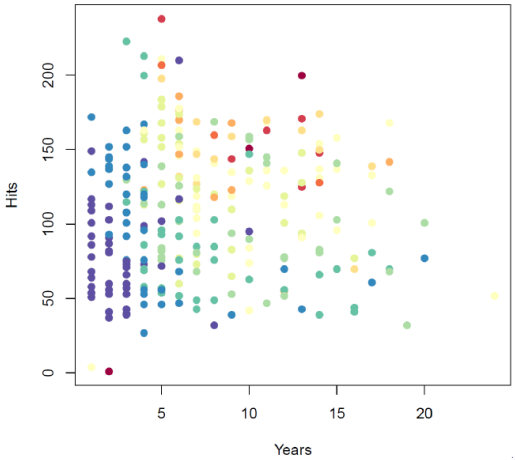


The basics of Decision Trees

- Decision trees are a non-parametric supervised learning method.
- The decision tree on the previous slide is an example of a classification decision tree.
- Decision trees can be applied to both regression and classification problems, which leads to them often being referred to as *CART* — Classification And Regression Trees.
- Just like with linear models, we will first start with regression trees, and then move to classification trees.
- The general principle is the same for both types: partition the feature space into several non-overlapping regions and make the same prediction for all observations that fall into one particular region

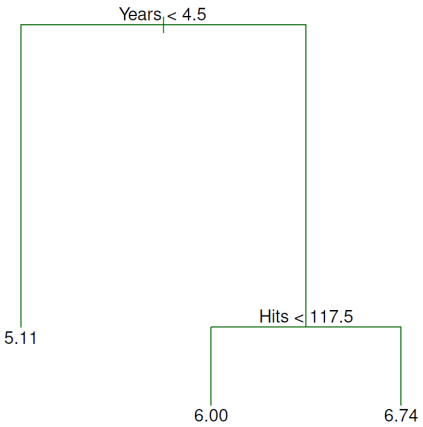
Example: Baseball Data

- Consider a subset of baseball data for Hitters with `log(Salary)` as the outcome variable and `Years` and `Hits` as the only two features.
- Salary values are colored according to a gradient, with lower salaries in purple, blue and higher salaries in orange, red.
- We want to build a regression tree for predicting the log of salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.



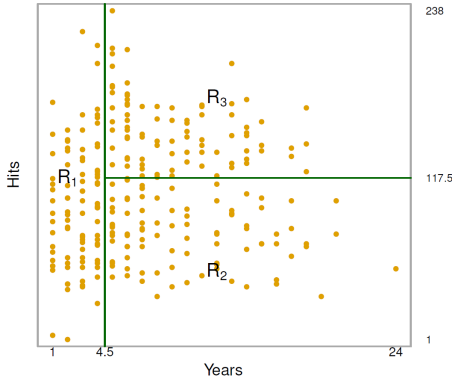
Example: Baseball Data

- For every *internal (decision)* node j the label of the form $X_k < t_j$ means that the feature space is split at this node with the left-hand branch corresponding to the subsample with $X_k < t_j$ and the right-hand branch to the subsample with $X_k \geq t_j$.
- For example, the split at the top corresponds to the left branch having observations only with `Years`$<$ 4.5 and the right branch only with `Years`$\geq$ 4.5.
- The tree has two *internal nodes* and three *terminal nodes*, or *leaves*. The number in each leaf is the mean of `log(Salary)` for the observations that fall there.



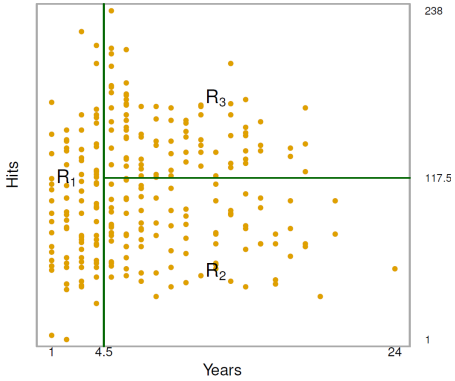
Example: Baseball Data

- The tree *stratifies* or segments all players into three regions of predictor space:
- R1 = $\{X | \texttt{Years} < 4.5\}$
- R2 = $\{X | \texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$
- R3 = $\{X | \texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5\}$
- How do we interpret this tree?



Example: Baseball Data

- `Years` is the most important factor in determining `Salary`, and players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced, the number of `Hits` that he made in the previous year seems to play little role in his `Salary`.
- But among players who have been in the major leagues for five or more years, the number of `Hits` made in the previous year does affect `Salary`, and players who made more `Hits` last year tend to have higher salaries.



Tree building process

- So how do we come up with a tree?
  1. We divide the predictor space (the set of possible values for $X_1, X_2, ..., X_p$) into J distinct and non-overlapping regions $R_1, R_2, ..., R_J$
  2. For every observation that falls into the region R_j we make the same prediction equal to the mean value of the outcome variable in the region R_j .
- The three main questions we need to answer are:
  1. What feature(s) X_k to use for splitting at each node?
  2. What should be the cutpoint t_j defining the region R_j in a form of $X_k < t_j$?
  3. How deep should out tree be, i.e. when do we stop doing splits?

Tree building process

- As a supervised learning algorithm, trees use the same concept of minimizing the loss function. For regression trees it the usual RSS (MSE), defined as

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

  where $\bar{y}_{R_j}$ is the mean response across all observations in region R_j .
- In theory regions R_j can have any shape, but the standard approach in tree methods is to use rectangular boxes only. What it means is that we use binary splits of the form $X_k < t_j$, as opposed to non-binary splits such as $X_{k1} + X_{k2} < t_j$ or $X_k^2 < t_j$.
- This is done both for simplicity of the estimation and for ease of interpretation of the resulting predictive model.

Binary vs Non-binary trees

Figure: Binary tree

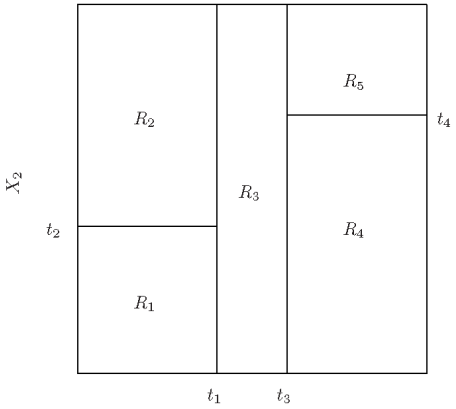
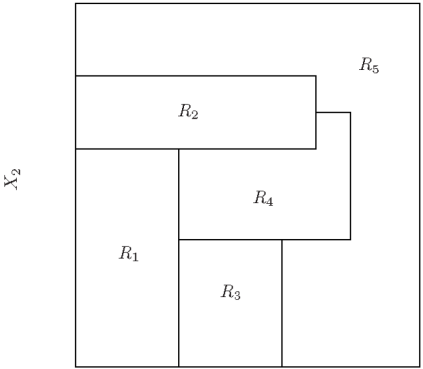


Figure: Non binary tree

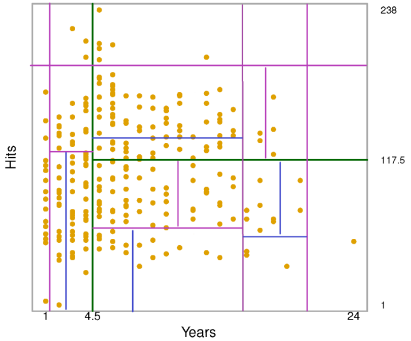


Tree building process

- There is one problem: finding a global optimum among all possible splits is computationally infeasible, similar to the best subset selection in linear regression models.
- Therefore the standard solution is to follow a *top-down*, *greedy* approach known as *recursive binary splitting*.
- The approach is *top-down* because it begins at the top of the tree and then successively splits the predictor space. Each split is indicated by the two new branches further down on the tree.
- It is *greedy* because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Tree Overfitting

- Why do we need a stopping criterion? What if we draw a tree until there are no more possible splits?
- Take our 2-split tree from before and add more splits.
- And then some more.
- We can continue this process until every single region in the picture has only 1 observation in it. This will give us exactly 0 RSS or 100% fit.



Tree Overfitting

- As in most other cases, a model with a 100% fit on a train data is likely *extremely overfitted* and thus will perform poorly on any new test data.
- A smaller tree with fewer splits (that is, fewer regions $R_1, R_2, ..., R_J$) might lead to a lower variance and better interpretation at the cost of a little bias.
- One possible alternative to the process described above is to grow the tree only so long as the decrease in the RSS due to each split exceeds some threshold.
- This strategy will result in smaller trees, but is too short-sighted: a seemingly worthless split early on in the tree might be followed by a very good split later on. That is, a split that seems bad at the current point may lead to a large reduction in RSS further down the tree.

Tree Pruning

- A better strategy is to grow a very large tree T_0, and then prune it back in order to obtain a *subtree*.
- This is usually done via *cost complexity pruning*, also known as *weakest link pruning*.
- The idea is to build a sequence of trees indexed by a non-negative tuning parameter α. For each value of α a subtree $T_\alpha \subset T_0$is grown that minimizes

$$\sum_{m=1}^{|T_\alpha|} \sum_{i \in R_m} (y_i - \bar{y}_{R_m})^2 + \alpha |T_\alpha| \tag{2}$$

where $|T_\alpha|$ is the number of terminal nodes in tree T_α, R_m is the region corresponding to the mth terminal node, and $\bar{y}_{R_m}$ is the mean outcome among observations in R_m.

Tree Pruning: Best Subtree

- The tuning parameter α controls a trade-off between the subtree's complexity and its fit to the training data.
- It acts in the similar way to the λ parameter in lasso/ridge regressions.
- Just like with lasso/ridge regressions, we need to use cross-validation to find the optimal value of α.
- Once that optimal value is obtained, we fit the corresponding tree T_α to the data.

Summary: Tree Algorithm

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 and/or the reduction in RSS is smaller than chosen threshold.
2. Apply cost complexity pruning to the large tree in order to obtain a sequence of best possible subtrees as a function of α.
3. Use K-fold cross-validation to choose optimal α. For each $k = 1, 2, ..., K$:
   1. Repeat Steps 1 and 2 on the $(K-1)/K^{th}$ fraction of the training data, excluding the k^{th} fold.
   2. Evaluate the mean squared prediction error on the data in the left-out k^{th} fold as a function of α.

   Average the results and pick α to minimize the average cross-validation error.
4. Return the subtree from Step 2 that corresponds to the optimal value of α.

Classification Trees

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree. For each terminal node we predict that each observation belongs to the most commonly occurring class of training observations in the region defined by that node.
- Similar to classification regression, we can use a variety of metrics to assess the accuracy of our trees.
- A natural alternative to RSS is the classification error rate, which is simply the fraction of the training observations in the region that do not belong to the most common class:

$$E = 1 - \max_k(\hat{p}_{mk}) \tag{3}$$

  Here $\hat{p}_{mk}$ represents the proportion of training observations in the m^{th} region that are from the k^{th} class.
- However classification error turns out to not be sufficiently sensitive for tree-growing, and in practice two other measures are used more commonly.

Gini Index and Cross-Entropy

- The *Gini index* is defined as

$$G = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{4}$$

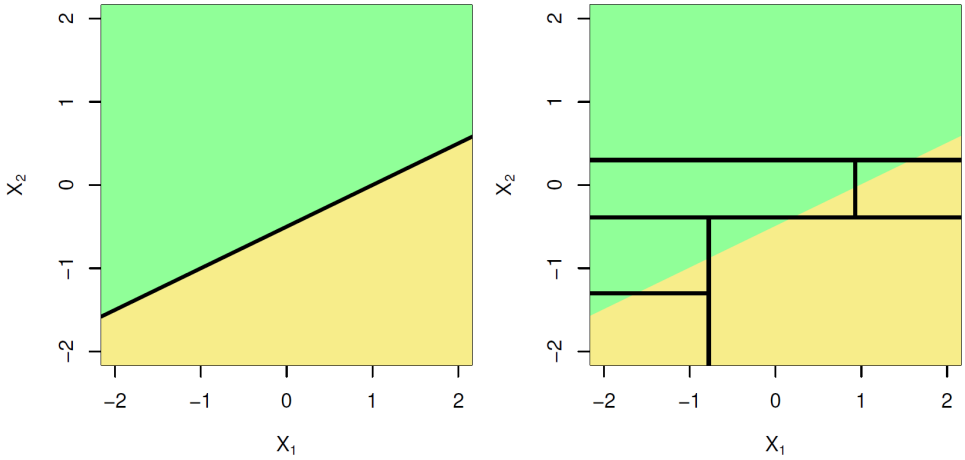
  which measures total variation across K classes. The Gini index is smaller if all of the $\hat{p}_{mk}$ values are close to 0 or 1.

- For this reason the Gini index is referred to as a measure of node *purity* — a small value indicates that a node contains predominantly observations from a single class.
- An alternative to the Gini index is *cross-entropy*, defined as

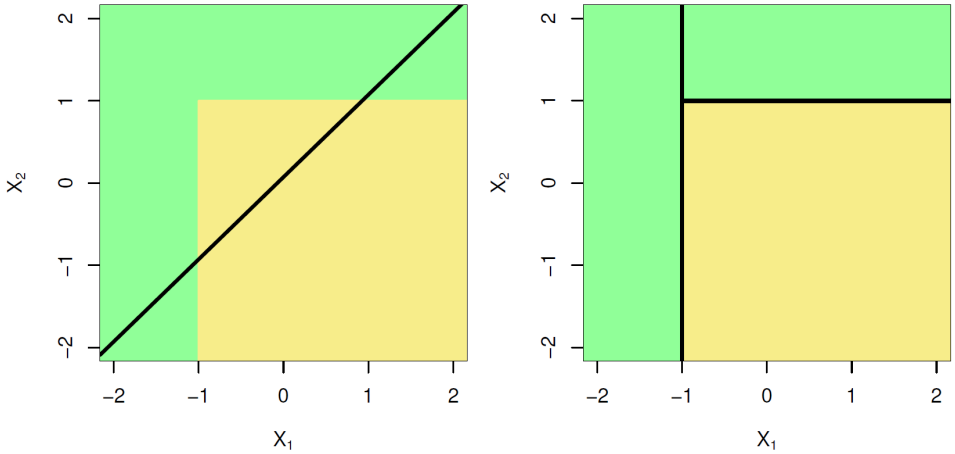
$$D = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}) \tag{5}$$

- With most datasets the Gini index and the cross-entropy are usually very similar numerically.

Trees vs Linear Models: True Linear Boundary



Trees vs Linear Models: True Non-Linear Boundary



Decision Trees: Advantages

- Trees are very easy to explain to people. In fact, they are even easier to explain than linear regression!
- Some people believe that decision trees more closely mirror human decision-making than do the regression and classification approaches seen in previous lecture.
- Trees can be displayed graphically, and are easily interpreted even by a non-expert (especially if they are small).
- Trees can easily handle qualitative predictors without the need to create dummy variables.
- The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.

Decision Trees: Disadvantages

- Prone to overfitting even if test/train splits are used. Require mechanisms such as pruning or stopping criteria to overcome this problem.
- Decision trees can be unstable. Small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- Classification tree algorithms can create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.
- Do not predict very accurately compared to other kinds of machine learning models, which is due to the greedy nature of the tree construction algorithm.

Improving Decision Trees

- The three most commonly used methods to improve the performance of tree methods and alleviate some/most of their disadvantages are *bagging, random forests and boosting*.
- *Bagging* is a form of model averaging used to significantly decrease the variance of tree predictions.
- *Random forests* combine bagging with random selection of features to use for splits.
- *Boosting* changes the tree growing algorithm to a slower, but more precise one.

The Power of Averaging

- Stats 101 has taught us that given a set of n independent (i.e. with correlation equal to 0) observations $Y_1, Y_2, ..., Y_n$ each with a variance of σ^2, the variance of the sample mean $\bar{Y}$ is given by σ^2/n
- In other words, averaging a set of observations reduces variance. If we were able to grow hundreds of trees on different datasets, the average of those trees would likely give us very precise predictions.
- Unfortunately, we rarely have access to multiple datasets. That is where ***b**ootstrap **agg**regation*, or ***bag**ging*, comes into play. It is a general-purpose procedure for reducing the variance of any statistical learning method, and is very often used in the context of decision trees.

Bagging

- Instead of trying to find multiple different datasets to average over, we can do bootstrap - take repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on the b^{th} bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction for f at point x. We then average all the predictions to obtain

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x) \tag{6}$$

- The above formula works for regression trees. With classification trees, for each test observation we record the class predicted by each of the B trees, and take a majority vote: the overall prediction is the most commonly occurring class among the B predictions.

Test Error Estimation with Bagging

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the out-of-bag (OOB) observations.
- We can predict the response for the ith observation using each of the trees in which that observation was OOB. This will yield around B/3 predictions for the ith observation, which we then average to obtain OOB test error.
- This estimate works in the similar way to LOO cross-validation error, especially when B is large.

Random Forests

- *Random forests* provide an improvement over bagged trees by way of a small tweak that *decorrelates* the trees. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, a *random selection of m predictors* is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors.
- A fresh selection of m predictors is taken at each split, and typically we choose m $= \sqrt{p}$ predictors to be considered at each split.

Random Forests

- Random forests are a relatively simple algorithm that usually turns out to be surprisingly hard to beat when it comes to prediction accuracy. Why?
- Variance reduction: the trees are more independent because of the combination of bootstrap samples and random draws of predictors. Recall that the power averaging reduces the variance to σ^2/n only in fully independent samples with 0 correlation.
- Bias reduction: a very large number of predictors (even more that the number of observations) can be considered, and local feature predictors can play a role in the tree construction.

Boosting

- Like bagging, *boosting* is a general approach that can be applied to many statistical learning methods for regression or classification.
- Recall that bagging involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on its own bootstrapped data set, independent of the other trees.
- Boosting works in a similar way, except that the trees are grown *sequentially*: each tree is grown using information from previously grown trees.

Boosting Algorithms for Regression trees

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

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

   3. Update the residuals

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x)$$

3. Output the boosted model

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

Boosting Algorithms: Details

- Unlike fitting a single large decision tree to the data, which amounts to *fitting the data hard* and potentially overfitting, the boosting approach instead *learns slowly (or iteratively)*.
- Given the current model, we fit a decision tree to the residuals from the current model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we iteratively improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.

Boosting Algorithms: Tuning Parameters

- The *number of trees B*. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
- The *shrinkage parameter* λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
- The *number of splits* d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a *stump*, consisting of a single split and resulting in an additive model. More generally d is the *interaction depth*, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Summary of Tree Methods

- Decision trees are simple and interpretable models for regression and classification.
- However, they are often not competitive with other methods in terms of prediction accuracy.
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- Random forests and boosting are among the state-of-the-art methods for supervised learning. Compared to standard decision trees, they achive the gains in prediction accuracy, but at the expense of interpretability.