Insurance analytics

A walk in the forest - tree-based machine learning methods

Katrien Antonio
LRisk - KU Leuven and ASE - University of Amsterdam

May 26, 2019

Acknowledgement

- Some of the figures in this presentation are taken from *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.

- Some of the figures in this presentation are from *Boosting insights in insurance tariff plans with tree-based machine learning* (available on arxiv, April 2019), written by Roel Henckaerts, Marie-Pier Côté, Katrien Antonio and Roel Verbelen.

Today's mission

- Today's mission is twofold:
  - a general discussion of decision trees, bagging, random forests, gradient boosting machines
  - a discussion of specific considerations to keep in mind when using these predictive modeling techniques with frequency/severity data.

Regression trees

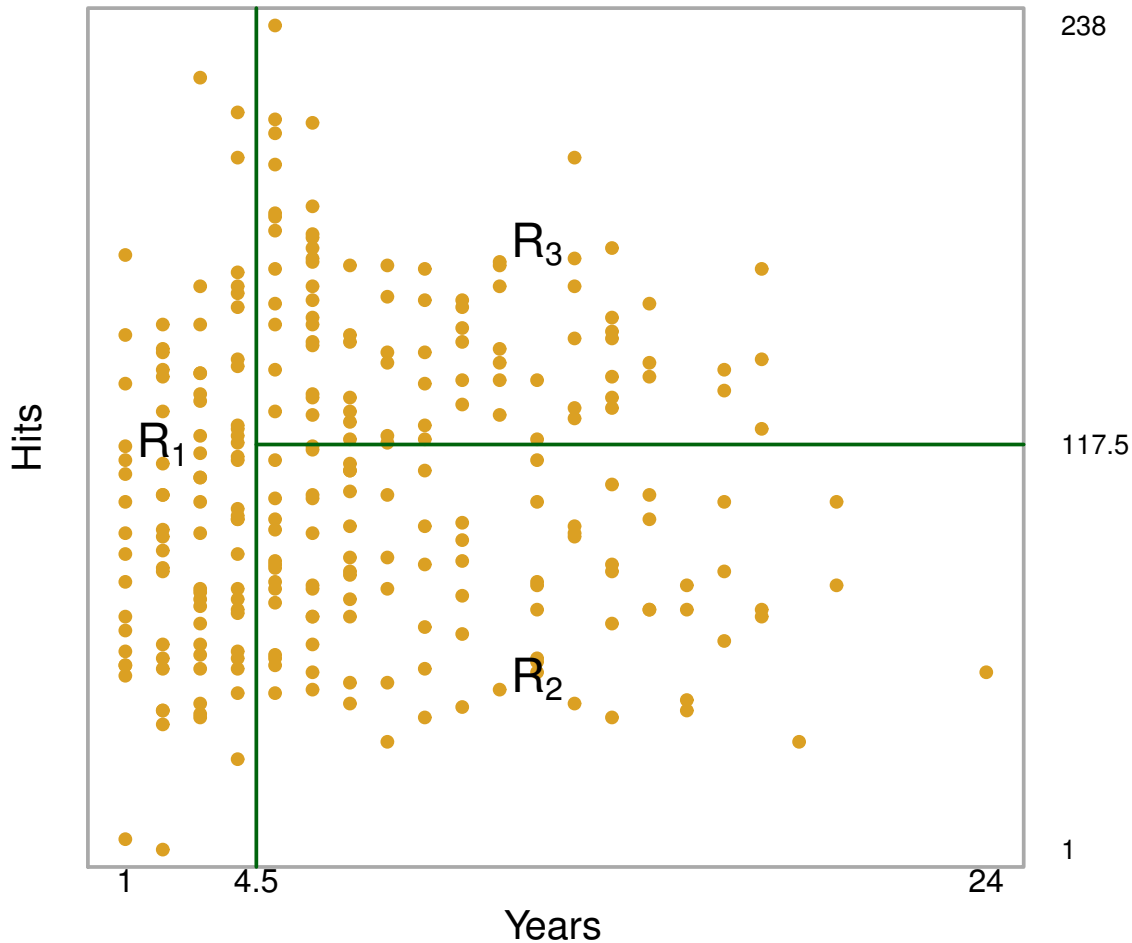
Intro

- CART: Classification And Regression Trees, introduced by Breiman et al. (1984).
- Consider the `Hitters` data to predict a player's `Salary` based on `Years` and `Hits`.

Regression trees

Intro

- The region partition for the `Hitters` data:



Regression trees

Intro

- The process of building a regression tree:

  1. divide the predictor space into J distinct, non-overlapping regions $R_1, R_2, \ldots, R_J$

     recall: the predictor space is the set of possible values for $X_1, X_2, \ldots, X_p$ (= the covariates).

  2. for every observation in region R_j we make the same prediction:

     the mean of the response values for the training observations in R_j.

- The prediction obtained with a regression tree:

$$f(X_1, \ldots, X_p) = \bar{y}_1 I_{\{\boldsymbol{x} \in R_1\}} + \ldots + \bar{y}_J I_{\{\boldsymbol{x} \in R_J\}},$$

where $\bar{y}_j = \text{ave}\,(y_i | \boldsymbol{x}_i \in R_j)$.

Regression trees

Intro

- We consider now step 1:

  how to construct the regions $R_1, \ldots, R_J$?

- We divide the predictor space into high-dimensional rectangles, or boxes.

- Find boxes $R_1, \ldots, R_J$ that minimize the Residual Sum of Squares (RSS) (actuarial reflections?)

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

  with $\bar{y}_{R_j}$ the mean response for the training observations within the jth box.

Regression trees

Intro

- Computationally infeasible to consider every possible partition of the feature space into J boxes.
- Therefore:

  use a top-down, greedy approach, known as recursive binary splitting.
- Motivation:
  - top-down because it begins at top of the tree
  - greedy because at each step the best split is made at that particular step, rather than looking ahead.

Regression trees

Intro

- To perform recursive binary splitting:
  - select the predictor X_j and cutpoint s such that

    $$\{X|X_j < s\} \text{ and } \{X|X_j \geq s\},$$

    leads to greatest possible reduction in RSS
  - thus, for any j and s we define the pair of half-planes

    $$R_1(j,s) \quad = \quad \{X|X_j < s\} \quad \text{and} \quad R_2(j,s) = \{X|X_j \geq s\},$$

    we seek j and s minimizing

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

Regression trees

Intro

- Next, we repeat the process:

  we look for the best predictor and best cutpoint to split data further,

  within each of the resulting regions.

- The process continues until a stopping criterion is reached, e.g. no region has > 5 observations.

- Given regions $R_1, \ldots, R_J$ we predict as follows: (step 2)

  - take a test observation
  - belongs to which region?
  - use the mean of the training observations in that region.

Regression trees

Loss functions

- What about alternatives for the Residual Sum of Squares (RSS)?
- Use a loss function $L(y_i, f(\boldsymbol{x}_i))$ and split the predictor space into $R_1(j,s)$ and $R_2(j,s)$ such that

$$\sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i)) = \sum_{i:\ \boldsymbol{x}_i \in R_1(j,s)} L(y_i, \bar{y}_{R_1}) + \sum_{i:\ \boldsymbol{x}_i \in R_2(j,s)} L(y_i, \bar{y}_{R_2})$$

  is minimized.
- Hereby $\bar{y}_{R_j}$ the 'mean' response for the training observations within the jth box.

Regression trees

Loss functions inspired by GLMs

- Recall the notion of the (scaled) deviance:

$$D(\boldsymbol{y}, \hat{f}(\boldsymbol{x})) \quad = \quad -2 \cdot \ln \left(\frac{\mathcal{L}(\hat{f}(\boldsymbol{x}))}{\mathcal{L}(\boldsymbol{y})} \right),$$

  where $\boldsymbol{y}$ is the vector of responses (or: targets) and $\hat{f}(\boldsymbol{x})$ is the vector of fitted values, $\mathcal{L}(\boldsymbol{y})$ is the likelihood of the saturated model and $\mathcal{L}(f(\boldsymbol{x}))$ the model likelihood.

- Now, use a loss function $L(y_i, f(\boldsymbol{x}_i))$ such that

$$D(\boldsymbol{y}, \hat{f}(\boldsymbol{x})) \quad = \quad \sum_{i=1}^{n} L(y_i, f(\boldsymbol{x}_i)).$$

Regression trees

Loss functions inspired by GLMs

- For example, loss function inspired by Poisson deviance

$$\begin{aligned} D(\boldsymbol{y}, \hat{f}(\boldsymbol{x})) &= 2 \cdot \ln \prod_{i=1}^{n} \exp(-y_i) \frac{y_i^{y_i}}{y_i!} \\ &\quad -2 \cdot \ln \prod_{i=1}^{n} \exp(-\hat{f}(\boldsymbol{x}_i)) \frac{\hat{f}(\boldsymbol{x}_i)^{y_i}}{y_i!} \\ &= 2 \sum_{i=1}^{n} \left(y_i \cdot \ln \frac{y_i}{\hat{f}(\boldsymbol{x}_i)} - (y_i - \hat{f}(\boldsymbol{x}_i)) \right). \end{aligned}$$

The corresponding (weighted) loss function $L(y_i, f(\boldsymbol{x}_i))$ is then

$$L(y_i, f(\boldsymbol{x}_i)) = 2 \cdot w_i \cdot (y_i \cdot \ln y_i - y_i \cdot \ln \hat{f}(\boldsymbol{x}_i) - y_i + \hat{f}(\boldsymbol{x}_i)).$$

When using an exposure measure d_i, $f(\boldsymbol{x}_i)$ is replaced by $d_i \cdot f(\boldsymbol{x}_i)$.

Regression trees

Loss functions inspired by GLMs

- For example, loss function inspired by normal deviance

$$\begin{aligned} D(\boldsymbol{y}, \hat{f}(\boldsymbol{x})) &= 2 \cdot \ln \prod_{i=1}^{n} \exp\left(-\frac{1}{2\sigma^2}(y_i - y_i)^2\right) \\ &\quad -2 \cdot \ln \prod_{i=1}^{n} \exp\left(-\frac{1}{2\sigma^2}(y_i - \hat{f}(\boldsymbol{x}_i))^2\right) \\ &= \frac{1}{\sigma^2} \sum_{i=1}^{n} (y_i - \hat{f}(\boldsymbol{x}_i))^2. \end{aligned}$$

The corresponding (weighted) loss function $L(y_i, f(\boldsymbol{x}_i))$ is then

$$L(y_i, f(\boldsymbol{x}_i)) = w_i \cdot (y_i - \hat{f}(\boldsymbol{x}_i))^2,$$

which is simply the squared error loss.

Regression trees

Finding optimal splits

- Finding the optimal split:
  - is straightforward for continuous predictors which can be ordered in a natural way
  - is easy for binary predictors
  - is way more complicated for categorical predictors with (e.g.) q levels, since $2^{q-1}-1$ possible partitions in two groups.
- Actuarial reflections? Postal code, multi-level factors?

Regression trees

Tree pruning

- Process as described above is likely to overfit the data.
- An alternative:

  build the tree so long as the decrease in the RSS due to each split exceeds some (high) threshold.

- Better strategy:

  grow a very large tree T_0 and prune it back to obtain a subtree.

- This is called cost complexity pruning.

Regression trees

Tree pruning

- The strategy:
  - consider a sequence of trees indexed by a nonnegative tuning parameter α
  - for each value of α there is a subtree $T \subset T_0$ such that

$$\sum_{j=1}^{|T|} \sum_{i:\ x_i \in R_j} (y_i - \bar{y}_{R_j})^2 + \alpha |T|$$

    is as small as possible.

- Hereby:
  - $|T|$ is the number of terminal nodes of the tree
  - $\bar{y}_{R_j}$ is the predicted response associated with R_j, the rectangle corresponding to the jth terminal node.

Regression trees

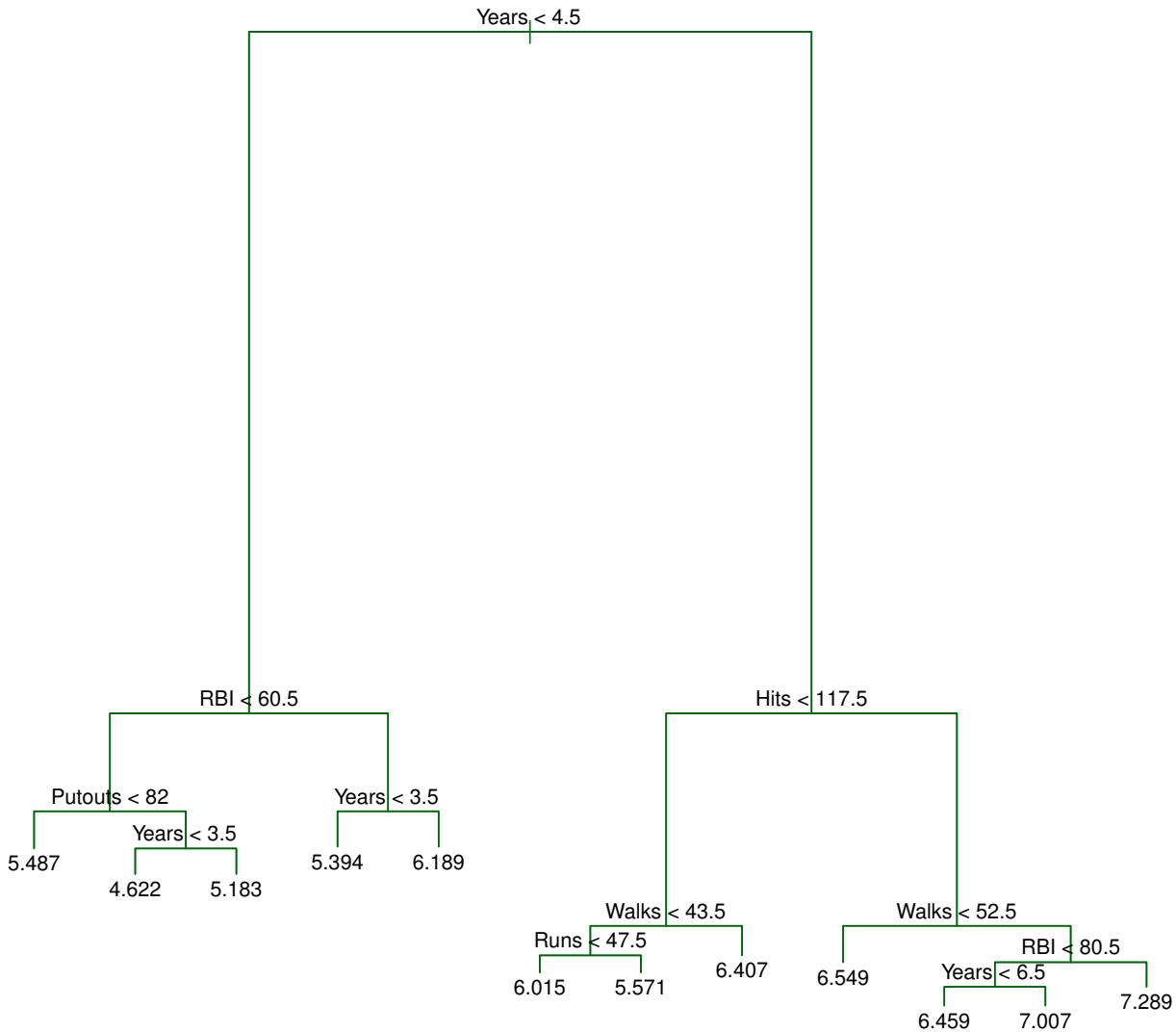
Tree pruning

- Tuning parameter α controls a trade-off between the subtree's complexity and its fit to training data.
- With $\alpha = 0$ the subtree T is equal to T_0.
- When α increases, there is a price to pay for having a tree with many terminal nodes.
- We can select a value of α using a validation set or using cross-validation.
- We return to the full data and obtain the subtree corresponding to α.

Regression trees

Tree pruning

Return to the `Hitters` data set: unpruned tree.



Regression trees

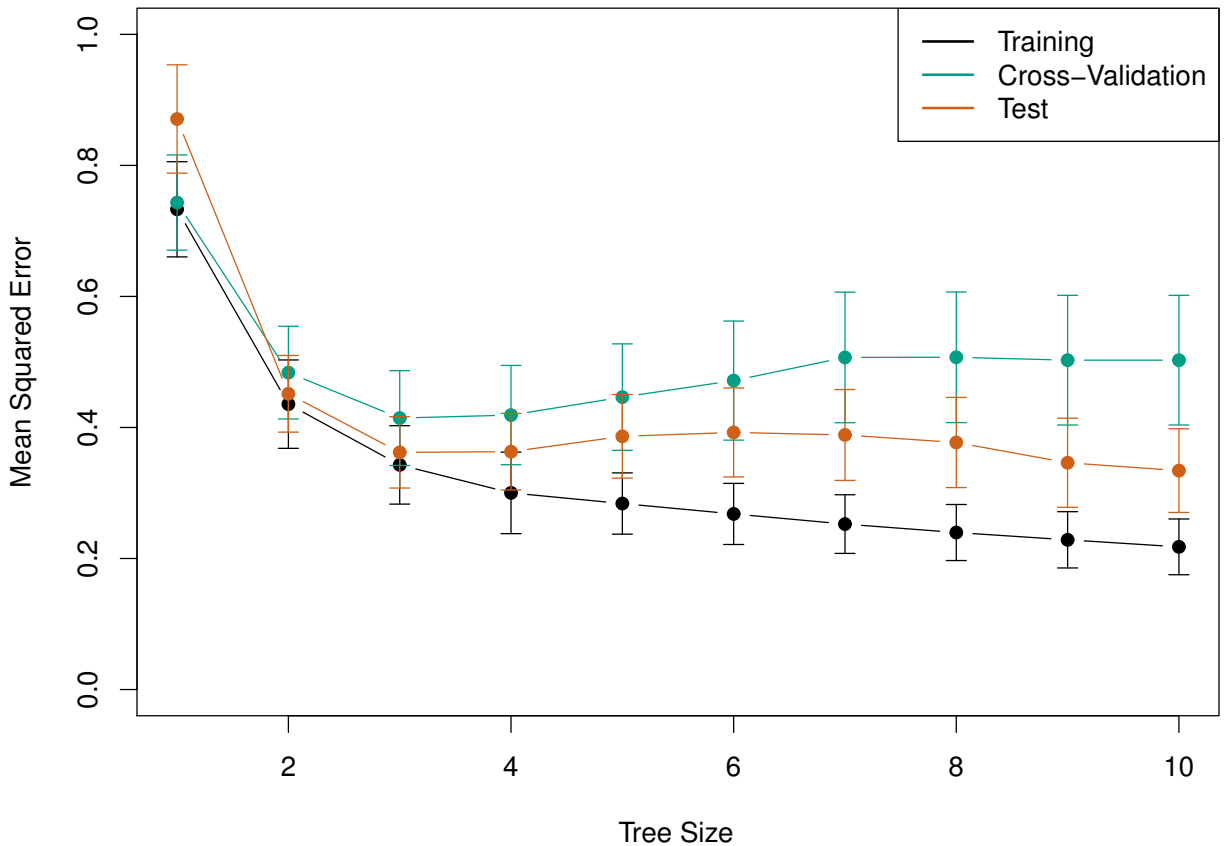
Tree pruning

- Return to the `Hitters` data set: unpruned tree.
- Strategy:
  - randomly divide the data in half, yields 132 observations in the training set and 131 in the test set
  - build a large tree on the training data and vary α to create subtrees
  - perform 6-fold cross validation and estimate cross-validated MSE of the trees as function of α.

Regression trees

Tree pruning

Return to the `Hitters` data set: training, cross-validation and test error.



CV error is minimal for the three-node tree (see earlier).

Regression trees

Tree pruning

- Return to the `Hitters` data set: pruned tree.

Regression trees

Tree in R with `rpart`

- Optimize performance of the tree by minimizing the following quantity:

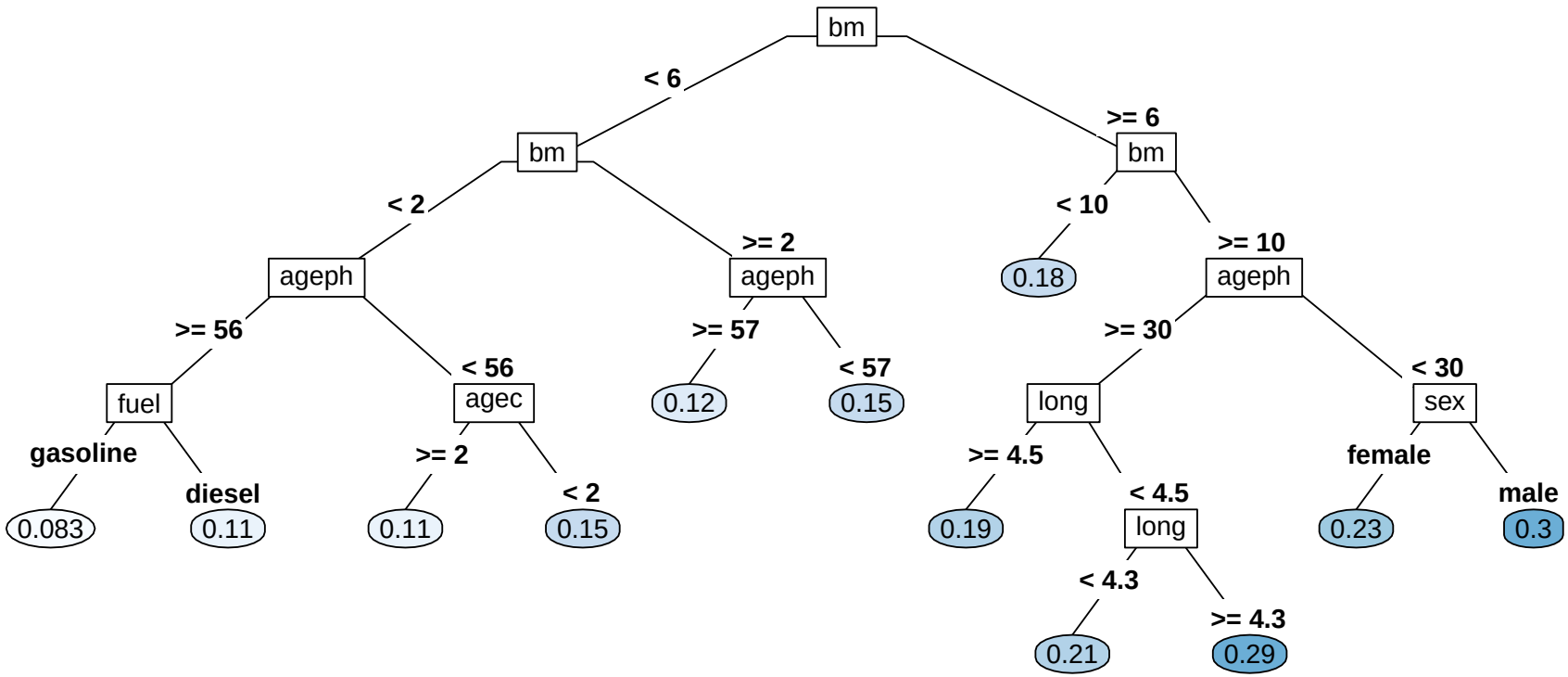
$$\sum_{j=1}^{J} \sum_{i\,:\,\boldsymbol{x}_i \in R_j} L(y_i, \hat{y}_{R_j}) \;+\; J \cdot cp \cdot \sum_{i\,:\,\boldsymbol{x}_i \in R} L(y_i, \hat{y}_R)$$

  - complexity parameter cp will determine the size of the tree
  - $cp = 0$ gives biggest possible tree
  - $cp = 1$ gives root tree without splits
  - cp is an important tuning parameter.

- We employ a tuning strategy and search grid to find the optimal value for cp, e.g. via cross-validation.

Regression trees

Example of a frequency tree with `rpart.plot`



Regression trees

Advantages and disadvantages of trees

+ Trees are very easy to explain.
+ Decision trees closely mirror human decision-making.
+ Trees can be displayed graphically, easily interpreted by non-expert.
+ Trees can easily handle qualitative predictors.
– Trees generally do not have same level of predictive accuracy as some other predictive modeling techniques discussed today.

Regression trees

Advantages and disadvantages of trees

- By aggregating many decision trees into ensembles of trees, the predictive performance can be substantially improved.
- These techniques are known as bagging, random forests and boosting (see further).

Bagging

- A natural way to reduce the variance and increase the prediction accuracy of a statistical learning method:
  - take many training sets from the population
  - build a separate prediction model using each training set
  - average the resulting predictions.
- Thus, calculate $\hat{f}^1(x),\ \hat{f}^2(x), \ldots, \hat{f}^T(x)$ using B separate training sets and average them

$$\hat{f}_{\text{avg}}(x) \quad = \quad \frac{1}{B}\sum_{b=1}^{B} \hat{f}^b(x).$$

However, usually we do not have access to multiple training sets.

Bagging

- Instead, we can bootstrap and take repeated samples from the (single) training data set.
- Thus, we generate B different bootstrapped training data sets.
- We train our method on the bth bootstrapped training set and get $\hat{f}^{\star b}(x)$. Finally, we average all the predictions

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{\star b}(x).$$

  This is called bagging and goes back to Breiman (1996). It mainly reduces the estimation variance.

- Bagging refers to Bootstrap Aggregating.

Bagging

- Bagging with regression trees:
  - construct B regression trees using B bootstrapped training sets
  - these trees are grown deep, and are not pruned
  - we then average the resulting predictions.
- Bagging with classification trees:
  - for a given test observation, we record the class predicted by each of the B trees and take a majority vote.

Random forests

- Random forests (Breiman, 2001) provide an improvement over bagged trees by way of a small tweak that decorrelates the trees.
- Random forests:
  - we build a number of decision trees on bootstrapped training samples
  - each time a split in a tree is considered, a random sample of m predictors is chosen as split candidates (from the full set of p predictors)
  - typically, $m \approx \sqrt{p}$.

Random forests

- Rationale:
  - suppose there is a very strong predictor in the data set
  - in the collection of bagged trees, most or all of the trees will use this predictor as top split
  - all bagged trees will look quite similar to each other
  - predictions are highly correlated; no substantial reduction in variance over a single tree.
- Random forests overcome this problem.

Random forests

Details

- The inventors of random forests recommend:
  - for classification, default value for m is $\lfloor \sqrt{p} \rfloor$ and the minimum node size is one
  - for regression, default value for m is $\lfloor p/3 \rfloor$ and the minimum node size is five.
- Instead of using defaults, better practice to use a tuning strategy and search grid.
- Possible tuning parameters in a random forest:
  - number of trees B
  - number of sample variables m
  - minimum node size n_{min}

Boosting

- Boosting is:
  - an iterative method that combines many weak learners into one powerful prediction, thus: boost your weak learner!
  - one of the most powerful shallow machine learning techniques.
- We focus on boosting with decision trees.
- Trees are grown sequentially; each tree is grown using information from previously grown trees.

Boosting

A bit of history

- In the early 1990s boosting algos appeared combining (or: boosting) a number of weak classifiers into an ensemble classifier with superior misclassification error rate.
- AdaBoost (1999) algorithm brought an effective implementation.
- Friedman et al. (2000):
  - connected AdaBoost to the statistical concepts of loss functions, additive modelling, and logistic regression
  - interpreted AdaBoost as forward stagewise additive model minimizing exponential loss.

Boosting

A bit of history

- This fundamental understanding led to a new view of boosting, enabling several extensions.
- Friedman's (2001) gradient boosting machines (GBMs):
  - for classification as well as regression
  - trees make an excellent weak or base learner for boosting
  - remedy greediness by employing shrinkage or regularization
  - inspired by bagging, update GBM with a random sampling scheme to become stochastic gradient boosting.

Boosting

Forward stagewise additive modeling

- We mentioned the connection between boosting and forward stagewise additive modeling.
- Forward stepwise linear regression of a response y with predictors $x_1, \ldots, x_p$:
  1. choose x_j giving the smallest RSS $\sum_i (y_i - \hat{\beta}_j x_{ij})^2$, where $\hat{\beta}_j$ is from regressing y on x_j
  2. choose x_k giving the smallest additional RSS $\sum_i (r_i - \hat{\beta}_k x_{ik})^2$, where $\hat{\beta}_k$ is from regressing $r_i = y_i - \hat{\beta}_j x_{ij}$ on x_k
  3. repeat the last step.

Boosting

Fitting an additive model

- Fit an additive expansion in a set of 'basis' functions:

$$f(x) \quad = \quad \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}; \gamma_m),$$

with the $b(\boldsymbol{x}; \gamma_m)$ simple functions of $\boldsymbol{x}$, with parameters $\boldsymbol{\gamma}$.

- We fit these by minimizing a loss function over training data

$$\min_{\{\beta_m,\ \gamma_m\}_1^M} \sum_{i=1}^{n} \mathcal{L}\left(y_i, \sum_{m=1}^{M} \beta_m b(\boldsymbol{x}_i; \gamma_m)\right).$$

- To make this computationally feasible, fit just a single basis function:

$$\min_{\beta,\ \gamma} \sum_{i=1}^{n} \mathcal{L}\left(y_i, \beta b(\boldsymbol{x}_i; \gamma)\right).$$

Boosting

Fitting an additive model

- The forward stagewise additive model works like this:
  - squared-error loss $\mathcal{L}(y, f(x)) = (y - f(x))^2$
  - stagewise additive model

$$\begin{aligned} L(y_i, f_{m-1}(\boldsymbol{x}_i) + \beta b(\boldsymbol{x}_i; \gamma)) &= (y_i - f_{m-1}(\boldsymbol{x}_i) - \beta b(\boldsymbol{x}_i; \gamma))^2 \\ &= (r_{im} - \beta b(\boldsymbol{x}_i; \gamma))^2, \end{aligned}$$

    where $r_{im} = y_i - f_{m-1}(\boldsymbol{x}_i)$ is the residual of the current model on observation i.

- Forward stepwise linear regression is a simple example of this general approach.

Boosting

Other loss functions

- AdaBoost is equivalent to forward stagewise additive modelling with

$$\mathcal{L}(y, f(x)) = \exp\left(-yf(x)\right),$$

  the exponential loss function.

- Squared-error loss (regression) and exponential loss (classification) lead to very intuitive boosting algorithms:
  - fit base learner to residuals from current model
  - do a weighted fit of base learner to output y_i with weights $w_i = \exp\left(-y_i f_{m-1}(\boldsymbol{x}_i)\right)$.

- Other loss functions do not give rise to such simple boosting algorithms!

Boosting

Boosting trees

- Predictive rule in a decision tree (see College 6)

$$\boldsymbol{x} \in R_j \Rightarrow f(\boldsymbol{x}) = \gamma_j,$$

where γ_j is typically $\bar{y}_{R_j}$, the mean response of the training observations in R_j.

- The boosted tree is then a sum of trees

$$f_T(\boldsymbol{x}) = \sum_{t=1}^{T} f_{\text{tree}}(\boldsymbol{x}, \Theta_t),$$

induced in a forward stagewise procedure. Here, $\Theta_t = \{R_{jt}, \gamma_{jt}\}_1^{J_t}$, the regions and fitted values of the tree.

Boosting

Boosting trees

- Fitting trees in a forward stagewise procedure, solve:

$$\hat{\Theta}_t = \arg\min_{\Theta_t} \sum_{i=1}^{n} \mathcal{L}(y_i, f_{t-1}(\boldsymbol{x}_i) + f_{\text{tree}}(\boldsymbol{x}; \Theta_t)),$$

  where $\Theta_t = \{R_{jt}, \gamma_{jt}\}_1^{J_t}$, the regions and fitted values of the tree.

- With squared-error loss:

  fit a regression tree to the current residuals $y_i - f_{t-1}(\boldsymbol{x}_i)$.

- With other loss functions, no simple fast boosting algorithms.

Boosting

Gradient boosting machines

- To generalize the boosted tree strategy to any differentiable loss function $\mathcal{L}$, consider

$$\mathcal{L}(f) = \sum_{i=1}^{n} \mathcal{L}(y_i, f(\boldsymbol{x}_i)).$$

- The goal is to minimize the loss wrt f (where f is constrained to be a sum of trees).

- We rely on the analogy with numerical optimization

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} \mathcal{L}(\boldsymbol{f}).$$

Boosting

Gradient boosting machines

- Use gradient descent to solve

$$\hat{\boldsymbol{f}} = \arg\min_{\boldsymbol{f}} \mathcal{L}(\boldsymbol{f}).$$

- Move in the local direction for which $\mathcal{L}(\boldsymbol{f})$ is most rapidly decreasing at $\boldsymbol{f} = \boldsymbol{f}_{t-1}$. Then,

$$\boldsymbol{f}_t = \boldsymbol{f}_{t-1} - \rho_t \boldsymbol{g}_t,$$

with ρ_t a step length and $\boldsymbol{g}_t$ the gradient vector

$$g_{it} = \left[\frac{\partial \mathcal{L}(y_i, f(\boldsymbol{x}_i))}{\partial f(\boldsymbol{x}_i)}\right]_{f(\boldsymbol{x}_i)=f_{t-1}(\boldsymbol{x}_i)}.$$

Boosting

Gradient boosting machines

- The gradient tree-boosting algorithm:
  - initializes to the optimal constant model, which is just a single terminal node tree
  - fits a small tree of depth d to the pseudo-residuals $\rho_{it} = -\frac{\partial \mathcal{L}(y_i, f(x_i))}{\partial f(x_i)}$ evaluated at current model fit f_{t-1}
  - a shrinkage parameter λ controls the learning speed by shrinking updates $f_{\text{new}}(\boldsymbol{x}) = f_{\text{old}}(\boldsymbol{x}) + \lambda \text{update}$.
- Stochastic gradient boosting injects randomness in the training process by subsampling the data at random without replacement in each iteration.

Boosting

Gradient boosting machines

initialize fit to the optimal constant model: $f_0(\boldsymbol{x}) = \arg\min_b \sum_{i=1}^n \mathscr{L}(y_i, b)$;
for $t = 1, \ldots, T$ **do**
  randomly subsample data of size $\delta \cdot n$ without replacement from data $\mathcal{D}$;
  for $i = 1, \ldots, \delta \cdot n$ **do**
    $\rho_{i,t} = -\left[\frac{\partial \mathscr{L}\{y_i, f(\boldsymbol{x}_i)\}}{\partial f(\boldsymbol{x}_i)}\right]_{f=f_{t-1}}$
  end
  fit a tree of depth d to the pseudo-residuals $\rho_{i,t}$ resulting in regions $R_{j,t}$ with $j = 1, \ldots, J_t$;
  for $j = 1, \ldots, J_t$ **do**
    $\hat{b}_{j,t} = \arg\min_b \sum_{i\,:\,\boldsymbol{x}_i \in R_{j,t}} \mathscr{L}\{y_i, f_{t-1}(\boldsymbol{x}_i) + b\}$
  end
  update $f_t(\boldsymbol{x}) = f_{t-1}(\boldsymbol{x}) + \lambda \sum_{j=1}^{J_t} \hat{b}_{j,t} \mathbb{1}(\boldsymbol{x} \in R_{j,t})$;
end
$f_{\text{gbm}}(\boldsymbol{x}) = f_T(\boldsymbol{x})$;

Algorithm 2: Procedure to build a (stochastic) gradient boosting machine.

Boosting

Gradient boosting machines

- Tuning parameters are:
  1. the number of trees; boosting can overfit if this number is too large.
  2. the number d of splits in each tree, which controls the complexity of the boosted ensemble. $d = 1$ uses stumps (a single split) and often works well. d is the interaction depth.
- Hyper-parameters are:
  1. the shrinkage parameter λ, which controls the rate at which boosting learns.
  2. the fraction δ such that at each step a randomly selected subsample of size $\delta \cdot n$ is used.