

# Statistical Computing

Michael Mayer

March 2023



# Statistical Computing: What will we do?

## Chapters

1. R in Action
2. Statistical Inference
3. Linear Models
4. Model Selection and Validation
5. Trees
6. Neural Nets

## Remarks

- ▶ Chapters 3 to 6:  
Statistical ML in Action
- ▶ Two weeks per chapter
- ▶ Exercises at end of chapter notes

# Trees

# Outline

- ▶ Decision Trees
- ▶ Random Forests
- ▶ Gradient Boosted Trees

3 most important things for modeling:

- 1) Understand data structure  
→ what does a row mean
- 2) Mind your validation strategy  
→ how do I do it <sup>split</sup> → leakage
- 3) XAI explainable linear regression  
interpret your models

# Decision Trees

- ▶ Simple
- ▶ Easy to interpret
- ▶ Decision trees are like wolves:  
Weak alone, strong together
- ▶ Around since 1984  
(Breiman, Friedman)



<https://images.pexels.com/photos/3732527/pexels-photo-3732527.jpeg>

# What is a Decision Tree?

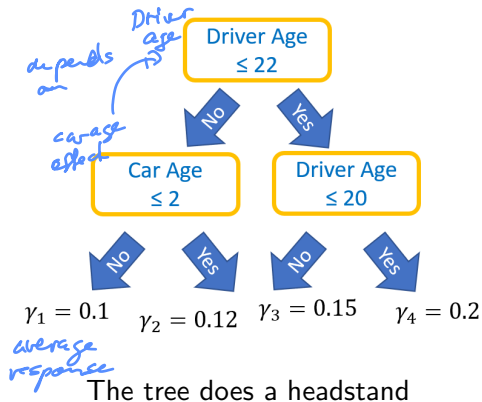
## Greedy recursive partitioning

1. Split: find best “yes/no” question on best feature to make total loss smaller
2. Apply Step 1 recursively

## Predictions

- ▶ Follow splits and use leaf value  $\gamma_j$
- ▶ Usually,  $\gamma_j$  is average response in leaf  $j$
- ▶ Terminal regions  $R_1, \dots, R_J$  *partition*
- ▶  $\mathbf{x}$  falls in leaf  $j \Leftrightarrow \mathbf{x} \in R_j$
- ▶  $\hat{f}(\mathbf{x}) = \sum_{j=1}^J \gamma_j \mathbf{1}\{\mathbf{x} \in R_j\}$

*leave node value*



## Example

# Properties of Decision Trees

outliers in response matters!

DT → interaction detection machines

- ▶ Outliers *no problems normally only the orders matter*
- ▶ Missing values *depends on implementation take avg or code it (-1)*
- ▶ Categorical covariates *unordered → every subset searching computational expensive*
- ▶ Greedy *think just 1 step ahead*
- ▶ Interactions *yes, there are.*
- ▶ Extrapolation *outside covariate range is not possible*
- ▶ Instability *↳ every split impacts the leave split*  
*→ same answer for 10y old cars and 20y old cars*

Most properties are inherited to groups/ensembles of decision trees  
*forest*

From nearest neighbors to decision trees: Short video by Jerome Friedman:

<https://www.youtube.com/watch?v=8hupHmBVvb0>

# Random Forests

- ▶ Combine many decision trees
- ▶ Perform very well
- ▶ Black Box
- ▶ Around since 2001 (Breiman)
- ▶ Why is combination of trees better than a single one?



<https://images.pexels.com/photos/1459534/pexels-photo-1459534.jpeg>



# Ensembling and Bagging

performance : multiple models  
→ avg  
practice : one model

## Ensembling

- ▶ Combine multiple models (**base learners**) to single one
- ▶ Example:  $k$ -nearest-neighbor with different  $k$
- ▶ Combined predictions have lower variance → better test performance (diversified stock portfolio, Bias-Variance Trade-Off)

## Algorithm: Bagging (**B**ootstrap **a**ggregating)

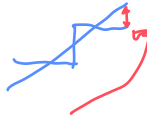
1. Select  $B$  bootstrapped training data sets from the original training data
2. Fit model  $\hat{f}^{*j}(\mathbf{x})$  on each of them
3. Return the bagged model  $\hat{f}(\mathbf{x}) = \frac{1}{B} \sum_{j=1}^B \hat{f}^{*j}(\mathbf{x})$

1. Bootstrap training data  
2. Fit model on each  
3. Avg result.

## Example

# Bias-Variance Trade-Off

Decomposition of generalization error:


$$\underbrace{\mathbb{E}(y_o - \hat{f}(x_o))^2}_{\text{Expected test MSE of } x_o} = \underbrace{\text{Var}(\hat{f}(x_o))}_{\text{Variation by taking different training sets}} + \underbrace{[\text{Bias}(\hat{f}(x_o))]^2}_{\text{Irreducible error}} + \underbrace{\text{Var}(\varepsilon)}_{\text{Irreducible error}}$$

*like infinite data*

*know conceptually*

*cannot be improved, there is no model which is better than that*

- ▶ One specific observation  $(y_o, x_o)$
- ▶ Expectations and variances over large number of training sets
- ▶ Bias: Error introduced by approximating true model by  $f$
- ▶ Low bias  $\leftrightarrow$  high variance  $\rightarrow$  Bias-Variance Trade-Off
- ▶ Bagged decision trees: low bias (why?) and low variance

## Remarks on Bagging

- ▶ Works best with unstable base learners  $\rightarrow$  deep decision trees
- ▶ Out-of-bag (OOB) validation  $\rightarrow$
- ▶ Parallel computing  $\oplus$
- ▶ Performance versus complexity

20 models  
are not  
interpretable

1. obs  $\times_0$   
100 bagged trees:  $\approx 30$  are not in the bag  
*'free cross validation'*

# From Bagging to Random Forests

A random forest is a bagged decision tree with an extra twist

## Twist

- ▶ Additional source of randomness
- ▶ Each split considers only random feature subset (often  $p/3$  or  $\sqrt{p}$ )
- ▶ Additional decorrelation  $\rightarrow$  stronger diversification

## Algorithm: Random forest (regression)

1. Select  $B$  bootstrapped training data sets from the original training data
2. Fit (usually deep) decision tree  $\hat{f}^{*j}(\mathbf{x})$  on each of them. For each split, consider only random feature subset
3. Return the random forest  $\hat{f}(\mathbf{x}) = \frac{1}{B} \sum_{j=1}^B \hat{f}^{*j}(\mathbf{x})$

## Comments on Random Forests

- ▶ Number of trees
- ▶ Deep trees *low bias models!*
- ▶ Don't trust performance on training set → OOB performance
- ▶ Parameter tuning  
*defaults usually good*

Regarding **parameter tuning**: Short video by Adele Cutler on working with Leo Breiman: [https://www.youtube.com/watch?v=t8ooi\\_tJHSE](https://www.youtube.com/watch?v=t8ooi_tJHSE)

### Example

# Interpreting a Black Box

## Study

1. Performance
2. Variable importance
3. Effects

*one very important to check*

## XAI

- ▶ e**X**plainable **A**rtificial **I**ntelligence
- ▶ Collection of methods to interpret models
- ▶ Examples: Split-gain importance, ICE, PDP

*are those important which i highlight*

## Example

- ▶ Split gain importance of random forest
- ▶ Variable importance and linear regression?

*/count*

*study variable importance in tree based models*  
*how much did model gain in MSE per split*

*plots about effects*  
*→ apply to all models*

# Individual Conditional Expectation (ICE)

## Basic thinking

- ▶ In **additive** linear model  $f$ , the effect of  $X^{(j)}$  is fully described by its coefficient(s)
- ▶ It describes how  $f$  reacts on changes in  $X^{(j)}$  (Ceteris Paribus)
- ▶ What if model involves complex interactions?

multiple numeric values

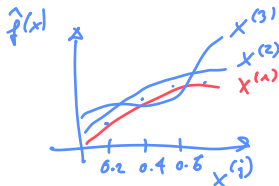
## Idea (Goldstein et al., 2015)

- ▶ Study (Ceteris Paribus) effect of  $X^{(j)}$  for **one** observation
- ▶ *ICE function* for feature  $X^{(j)}$  of model  $f$  and observation  $\mathbf{x} \in \mathbb{R}^p$

$$\text{ICE}_j : v \in \mathbb{R} \mapsto f(v, \mathbf{x}_{\setminus j})$$

replace  $x_j$  with  $v$

- ▶  $\mathbf{x}_{\setminus j}$  denotes all but the  $j$ -th component of  $\mathbf{x}$ , which is replaced by  $v$
- ▶ *ICE curve* represents graph  $(v, \text{ICE}_j(v))$  for grid of values  $v \in \mathbb{R}$



interested in shape  
can be different  
due to interactions

# ICE Plot: Visualize ICE Curves of many Observations

## Example

### Notes

- ▶ Curves with different shapes indicate interaction effects
- ▶ Parallel curves  $\Leftrightarrow$  additivity in  $X^{(j)}$
- ▶ Centered ICE plots
- ▶ Usually on link scale (why?)

### Pros and Cons

- + Simple to compute
- + Easy to interpret (Ceteris Paribus)
- + Gives impression about interactions
- Ceteris Paribus can be unnatural
- Model applied to rare/impossible  $\mathbf{x}$



# Partial Dependence Plot PDP (Friedman 2001)

- ▶ Average of many ICE curves
- ▶ Ceteris Paribus effect of  $X^{(j)}$  averaged over all interaction effects
- ▶ (Empirical) partial dependence function of  $j$ -th feature

$$PD_j(v) = \frac{1}{n} \sum_{i=1}^n \hat{f}(v, \mathbf{x}_{i,\setminus j})$$

- ▶  $\mathbf{x}_{i,\setminus j}$  feature vector of  $i$ -th observation without  $j$ -th component
- ▶ PDP equals graph  $(v, PD_j(v))$  for grid of values  $v \in \mathbb{R}$
- ▶ Sum runs over reference data (=?)
- ▶ Pros/cons similar to ICE, but no info on interaction

## Example

# Gradient Boosted Trees

Regression case

train small tree on data

→ get residuals

train another tree on the residuals

→ iterate.

- ▶ Combine many decision trees
- ▶ Perform very well
- ▶ Black Box
- ▶ Around since 2001 (Friedman)

} Like random forests



<https://www.gormananalysis.com/blog/gradient-boosting-explained/>

# Boosting

## Basic idea of boosting (e.g. Schapire, 1990)

1. Fit simple model  $\hat{f}$  to data
2. For  $k = 1, \dots, K$  do:
  - a. Find simple model  $\hat{f}^k$  that corrects the mistakes of  $\hat{f}$
  - b. Update:  $\hat{f} \leftarrow \hat{f} + \hat{f}^k$

*base learner*

## How to find updates $\hat{f}^k$ ?

Use decision trees  $\rightarrow$  **boosted trees**

- ▶ Use reweighting heuristic for binary classification
- note: not to know*  $\rightarrow$  AdaBoost (Freund and Schapire, 1995) *(check which variables has the biggest residual.  $\rightarrow$  large weights)*
- ▶ Reduce total loss  $Q(\hat{f} + \hat{f}^k) = \sum_{i=1}^n L(y_i, \hat{f}(\mathbf{x}_i) + \hat{f}^k(\mathbf{x}_i))$   
 $\rightarrow$  **Gradient boosting** (Friedman, 2001) *L typically square loss*

*take base learner which minimise Q  
minimise a function of a function*

# Gradient Descent

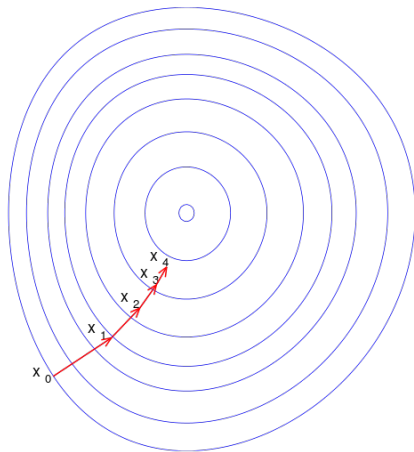
*not function space*

Minimize function  $h : \mathbb{R}^n \rightarrow \mathbb{R}$

1. Start at some value  $\hat{x} \in \mathbb{R}^n$
2. Repeat:  $\hat{x} \leftarrow \hat{x} - \lambda g$ 
  - ▶  $\lambda > 0$ : Step size or learning rate
  - ▶ Gradient  $g \in \mathbb{R}^n$  of  $h$  at  $\hat{x}$ :

$$g = \left[ \frac{\partial h(x)}{\partial x} \right]_{x=\hat{x}}$$

- ▶  $g$  points in direction of steepest ascent



[https://en.wikipedia.org/wiki/Gradient\\_descent](https://en.wikipedia.org/wiki/Gradient_descent)

# Gradient Boosting

## Gradient descent of $Q(f)$

►  $Q(f) = \sum_{i=1}^n L(y_i, f(\mathbf{x}_i))$

►  $f = (f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)) \in \mathbb{R}^n$   
*vector of predictions*

1. Start at some value  $\hat{f}$

2. Repeat:  $\hat{f} \leftarrow \hat{f} - \lambda g$  with

$$g = \left[ \frac{\partial Q(f)}{\partial f} \right]_{f=\hat{f}}$$

having components

$$g_i = \left[ \frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i)=\hat{f}(\mathbf{x}_i)}$$

*evaluation of  $f$  at all training data  $\rightarrow \in \mathbb{R}^n$*

For squared error?

►  $L(y, z) = (y - z)^2 / 2$

*$g_i = -r_i$*

► Plugging in:  $g_i = -\underbrace{(y_i - \hat{f}(\mathbf{x}_i))}_{\text{Residual } r_i}$

► Repeat:  $\hat{f}(\mathbf{x}_i) \leftarrow \hat{f}(\mathbf{x}_i) + \underbrace{\lambda r_i}_{\hat{f}^k?}$

*no! can't use the response*

Boosting with  $\hat{f}^k = -\lambda g_i$ ? Now way...

1.  $y_i$  unknown in application *cannot evaluate the  $r_i$*

2. Should work for all  $\mathbf{x}$

→ replace  $-g_i$  by predictions of tree

# Gradient Boosted Trees for Squared Error Loss

## Algorithm

1. Initialize  $\hat{f}(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^n y_i$  *save boosting rounds*
2. For  $k = 1, \dots, K$  do:
  - a. For  $i = 1, \dots, n$ , calculate residuals  $r_i = y_i - \hat{f}(\mathbf{x}_i)$
  - b. Model the  $r_i$  as a function of the  $\mathbf{x}_i$  by fitting a regression tree  $\hat{f}^k$
  - c. Update:  $\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda \hat{f}^k(\mathbf{x})$  *just any model*
3. Output  $\hat{f}(\mathbf{x})$

## General loss functions?

- ▶ Replace residuals by negative gradients of loss function
- ▶ Leaf values might be suboptimal  $\rightarrow$  replace by optimal values

# Gradient Boosted Trees for General Losses

MAE

small radius

1. Initialize  $\hat{f}(\mathbf{x}) = \operatorname{argmin}_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$  *best constant model*
2. For  $k = 1, \dots, K$  do:
  - a. For  $i = 1, \dots, n$ , calculate negative gradients (pseudo-residuals)

$$r_i = - \left[ \frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)} \right]_{f(\mathbf{x}_i) = \hat{f}(\mathbf{x}_i)}$$

*derive this once for every loss fct.*

- b. Model  $r_i$  as function of  $\mathbf{x}_i$  by regression tree  $\hat{f}^k$  with terminal regions  $R_1, \dots, R_J$  *const. in terminal region*
- c. For each  $j = 1, \dots, J$ , use line-search to find the optimal leaf value *would fall in the leaf*  
*partition of the feature space*

*expensive*

$$\gamma_j = \operatorname{argmin}_{\gamma} \sum_{\mathbf{x}_i \in R_j} L(y_i, \hat{f}(\mathbf{x}_i) + \gamma)$$

*Line search  
what value should  $\gamma$  get to be optimal*

- d. Update:  $\hat{f}(\mathbf{x}) \leftarrow \hat{f}(\mathbf{x}) + \lambda \underbrace{\sum_{j=1}^J \gamma_j \mathbf{1}\{\mathbf{x} \in R_j\}}_{\text{modified tree}}$

3. Output  $\hat{f}(\mathbf{x})$

## Remarks

- ▶ Predictions are <sup>shrunk</sup>sum of short decision trees (with modified leaf values)
- ▶ Random forest: average of deep trees
- ▶ How to select learning rate  $\lambda$ , number of trees  $K$ , ...?
- ▶ (AdaBoost is gradient tree boosting with exponential loss)



# Modern Implementations

## Timeline

1. XGBoost (2014)
2. LightGBM (2016) *msft dev*
3. CatBoost (2017) *good on gpu by yandex*  
*not easy to interpret*

## Example

## Differences to Friedman's original

- ▶ Use of second order gradients  
→ no line-search necessary *→ newton steps*
- ▶ Histogram binning *caching*  
→ speeds up tree growth
- ▶ Penalized objective function  
*# leaf nodes, L2 penalty etc.*

# Parameter Tuning is Essential

1. Number of boosting rounds/trees  $K$   
→ find by early stopping (validation/CV)
2. Learning rate  $\lambda$   *$\lambda \cdot K \approx \text{const.}$   
 $1 \cdot 10 \rightarrow 100 \cdot 0.5 \rightarrow K = 200$*   
→ to get reasonable number of rounds
3. Regularization
  - ▶ Tree depth, number of leaves, loss penalties, etc.
  - ▶ → Grid/Randomized search and iterate process

## Example

- ▶ XGBoost
- ▶ LightGBM

## Comments

- ▶ Why not one big grid search on all parameters? *but  $\lambda$  and  $K$  are linked  
→ small prob. to get good results*
- ▶ Objective/metrics *B. Step.  
→ choose loss function  
perhaps more robust  
↳ MAE → Squared error,  
but minimise MAE  
in cross validation*