Foundations of Statistical and Machine Learning for Actuaries
(Tree-based) Ensemble methods and Interpretability

Edward (Jed) Frees, University of Wisconsin - Madison
Andres M. Villegas, University of New South Wales

July 2025

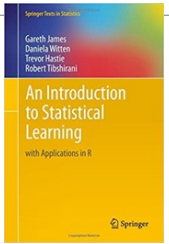
Schedule

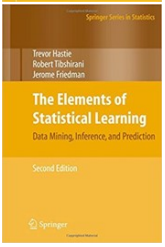
Day and Time	Presenter	Topics	Notebooks for Participant Activity
Monday Morning	Jed Jed	Welcome and Foundations; Hello to Google Colab Classical Regression Modeling	Auto Liability Claims Medical Expenditures (MEPS)
Monday Afternoon	Andrés Andrés	Resampling, cross-validation and regularisation Classification, Logistic Regression and Trees	Seattle House Sales Victoria road crash data
Tuesday	Andrés	(Tree-based) Ensembles methods and Interpretability	Victoria road crash data
Morning	Jed	Big Data, Dimension Reduction and Non-Supervised Learning	Big Data, Dimension Reduction and Non-Supervised Learning
Tuesday Afternoon	Jed Jed Fei	Neural Networks Graphic Data Neural Networks Fei Huang Thoughts on Ethics	Seattle House Prices, Claim Counts MNIST Digits Data
Wednesday Morning	Jed Jed	Recurrent Neural Networks, Text Data Artificial Intelligence, Natural Language Processing, and ChatGPT	Insurer Stock Returns
Wednesday After Lunch	Dani	Dani Bauer Insights	
Wednesday Afternoon	Andrés	Applications and Wrap-Up	

Tuesday Morning 3A - (Tree-based) Ensemble methods and Interpretability

- (Tree-based) Ensemble methods
  - Bagging
  - Random Forest
  - Boosting
- Interpretability
  - Variable importance measures
  - Partial dependence plots
  - Individual Conditional Expectation (ICE)
  - Shapley values
  - SHAP (SHapley Additive exPlanation)

Statistical Machine Learning: Resources



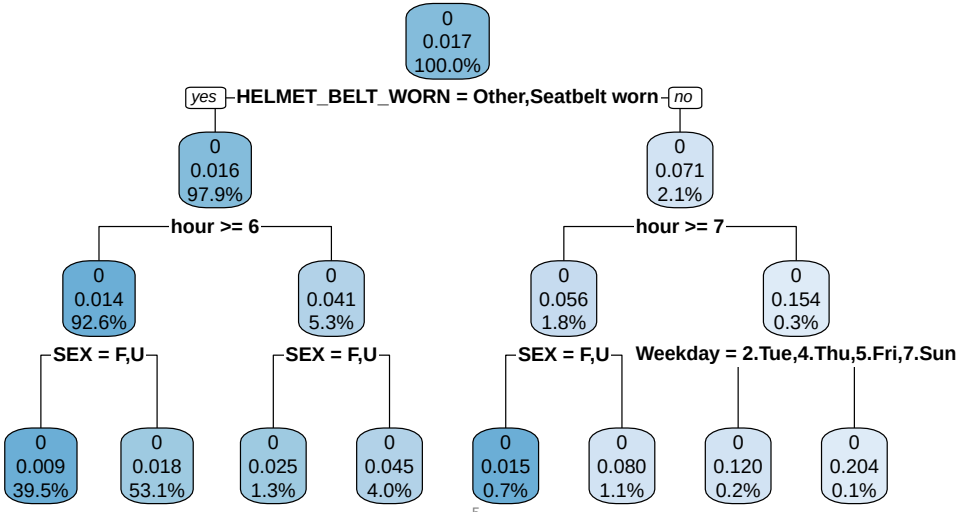


- Most of the discussion is based on this book:
  - Available at: https://www.statlearning.com/
  - Focus on intuition and practical implementation

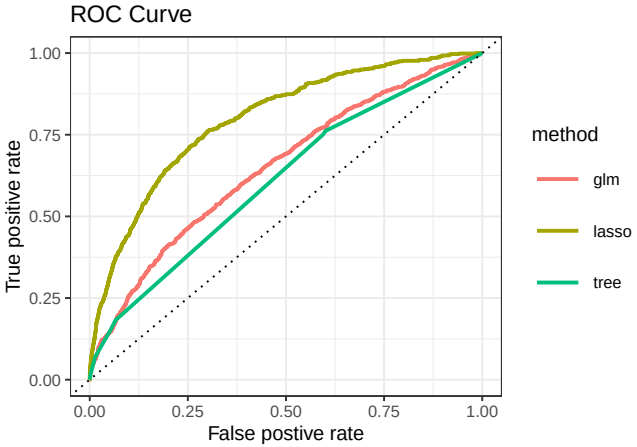
- This book can serve as reference for those interested in the math behind the methods
- Available at: http://web.stanford.edu/~hastie/ElemStatLearn/

The discussion builds on the UNSW Course Statistical Machine Learning for Risk and Actuarial Applications (https://unsw-risk-and-actuarial-studies.github.io/ACTL3142/)

Tree: VicRoads Crash Data



Tree: VicRoads Crash Data (ROC)



AUC

Method	Train	Test
glm	0.663	0.650
lasso	0.809	0.797
tree	0.629	0.610

Advantages and disadvantages of Trees

Advantages

- Easy to explain
- (Mirror human decision making)
- Graphical display
- Easily handle qualitative predictors

Disadvantages

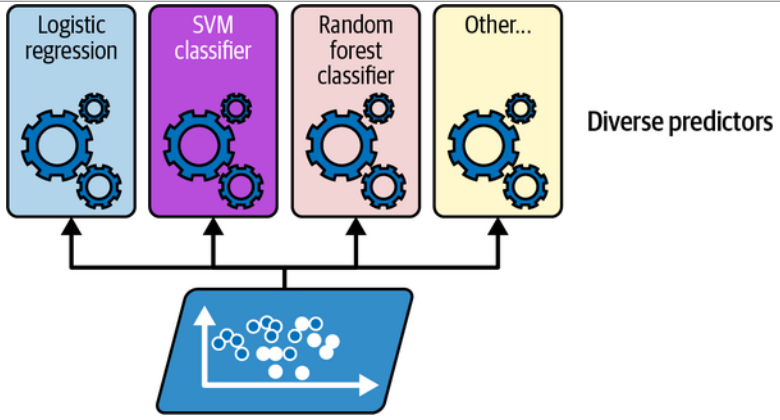
- Low predictive accuracy compared to other regression and classification approaches
- Can be very non-robust

Is there a way to improve the predictive performance of trees?

- Ensemble methods
- Bagging, random forest, boosting

Ensembles & Bootstrap

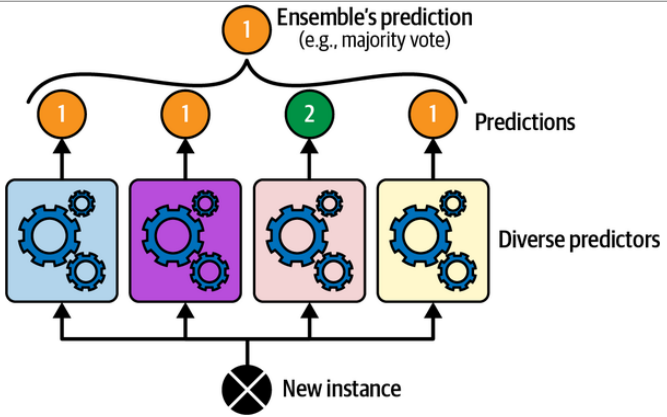
An ensemble is a group of models. . .



Training various different classifiers on the same dataset.

Source: Geron (2022), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd ed., Figure 7-1.

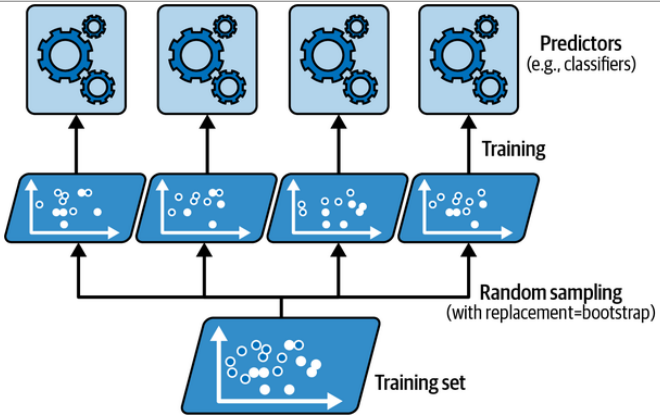
. . . & you combine their predictions



Make an overall prediction based on the majority vote of the models.

Source: Geron (2022), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd ed., Figure 7-2.

Bootstrapping



Train on different versions of the same data.

Source: Geron (2022), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd ed., Figure 7-4.

Bagging and Random Forest

Bootstrap Aggregation (Bagging)

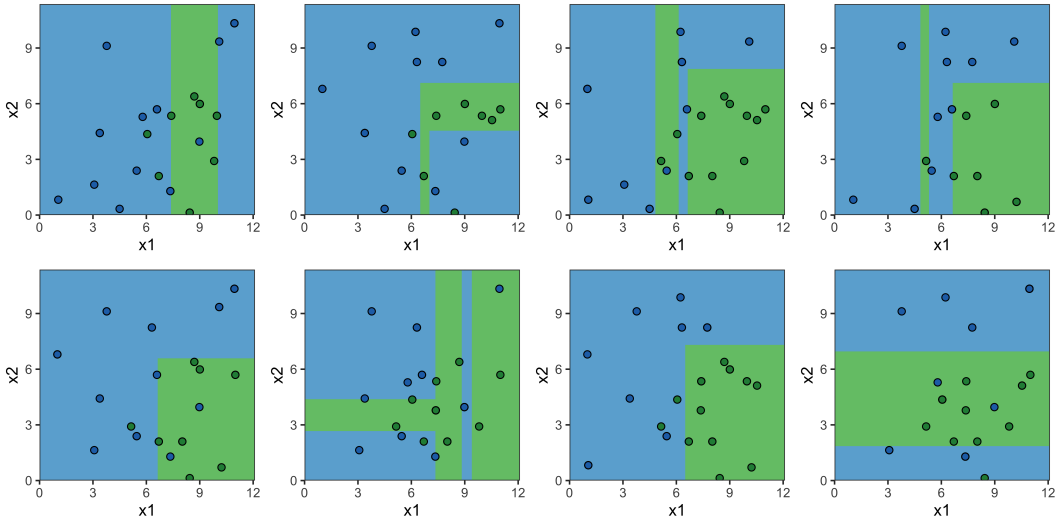
- A **general-purpose** procedure to reduce variance
  - particularly useful and frequently used in the context of decision trees

Bagging procedure:

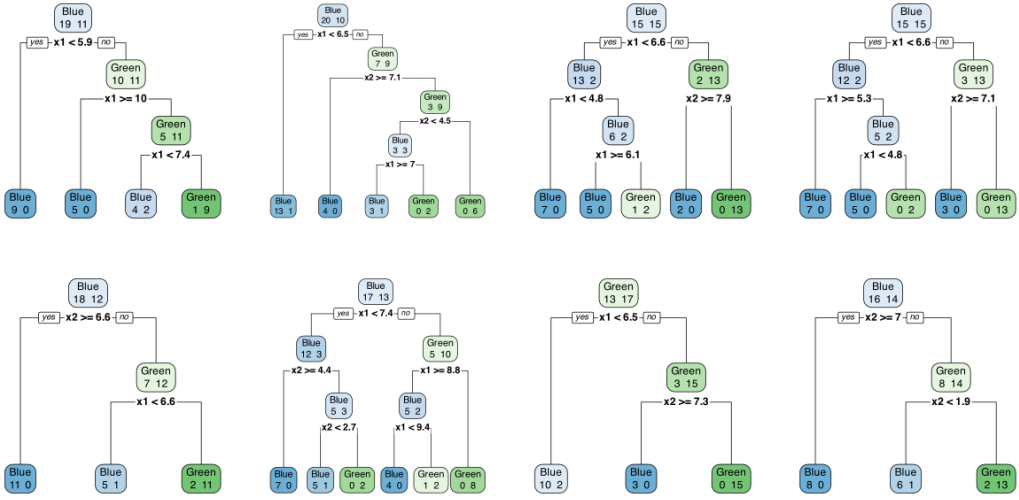
1. Bootstrap
   - sample with replacement repeatedly
   - generate B different bootstrapped training data sets
2. Train
   - train on the bth bootstrapped training set to get $\hat{f}^{*b}(x)$
3. Aggregate (Regression: average, Classification: majority vote)

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

Bagging: Illustration



Bagging: Illustration



Samples that are in the bag

Let's say element i, j of the matrix is 1 if the ith observation is in the jth bootstrap sample and 0 otherwise; i.e. it is "in the bag".

Boot_1	Boot_2	Boot_3	Boot_4	Boot_5
1	0	1	0	0
1	0	1	0	0
1	0	1	0	1
1	0	1	1	0
1	1	1	0	0
0	1	1	1	1
1	1	0	1	1
0	0	0	1	1
1	1	1	1	1
0	1	1	1	1

Samples that are out of bag

Now consider the inverse, element i, j of the matrix is 1 if the ith observation is **not** in the jth bootstrap sample, it is "out of the bag".

Boot_1	Boot_2	Boot_3	Boot_4	Boot_5	#OOB
0	1	0	1	1	3
0	1	0	1	1	3
0	1	0	1	0	2
0	1	0	0	1	2
0	0	0	1	1	2
1	0	0	0	0	1
0	0	1	0	0	1
1	1	1	0	0	3
0	0	0	0	0	0
1	0	0	0	0	1

Can perform "out of bag evaluation" by using the out of bag samples as a test set. This is cheaper than cross-validation.

Out-of-Bag Error Estimation

There is a very straightforward way to estimate the test error of a bagged model

- On average, each bagged tree makes use of around two-thirds of the observations
- The remaining one-third of the observations are referred to as the out-of-bag (OOB) observations
- Predict the response for the ith observation using each of the trees in which that observation was OOB
  - $\sim B/3$ predictions for the ith observation
- Take the average or a majority vote to obtain a single OOB prediction for the ith observation
- Turns out this is very similar to the LOOCV error.

Random forest

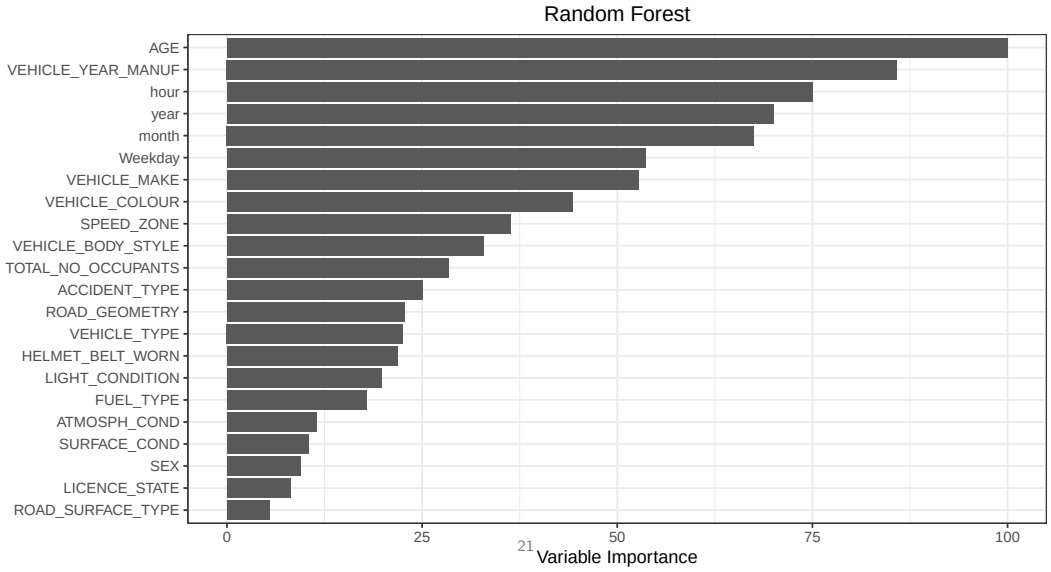
Random forests decorrelates the bagged trees

- At each split of the tree, a fresh random sample of m predictors is chosen as split candidates from the full set of p predictors
- Strong predictors are used in (far) fewer models, so the effect of other predictors can be properly measured.
  - Reduces the variance of the resulting trees
- Typically choose $m \approx \sqrt{p}$
- Bagging is a special case of a random forest with $m = p$

Bagging/Random Forest: Variable Selection and Importance

- Bagging and Random Forest can lead to difficult-to-interpret results, since, on average, no predictor is excluded
- Variable importance measures can be used
  - Bagging classification trees: Gini index reduction for each split (measure of node purity)
- Pick the ones with the highest variable importance measure

Random forest: Variable Importance VicRoads Data

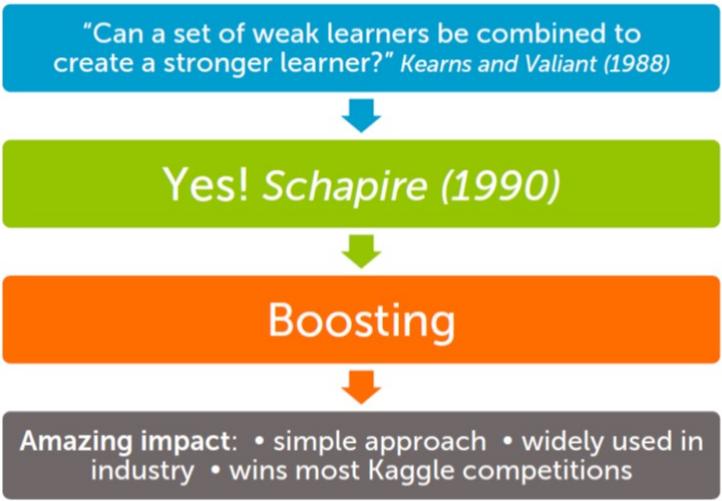


Bagging and Random forest in R

- `randomForest` package
- `mtry` Number of variables randomly sampled as candidates at each split, i.e., m.
- Bagging implemented if we set `mtry` to the number of variables ($m = p$)

Boosting

Boosting motivation



Boosting procedure

- A general approach that can be applied to many statistical learning methods for regression or classification.
- We focus on boosting for **regression trees**.
- It also combines a large number of decision trees, but:
  - It does **not** rely on independent bootstrap samples.
  - Trees are grown **sequentially**: each new tree uses information from previously grown trees.
  - Each tree is fitted on a modified version of the original data, namely, updated **residuals** from the previous tree.
- By fitting residuals, each step specifically targets areas where the previous tree performed poorly.
- Unlike standard trees, **boosting enforces slow learning**.

Boosting Algorithm for Regression Trees

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

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

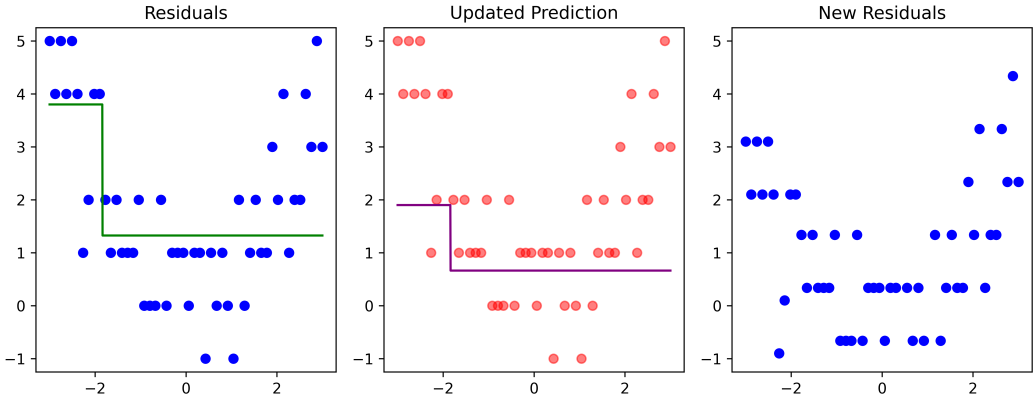
   (c) Update the residuals

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

3. Output the boosted model

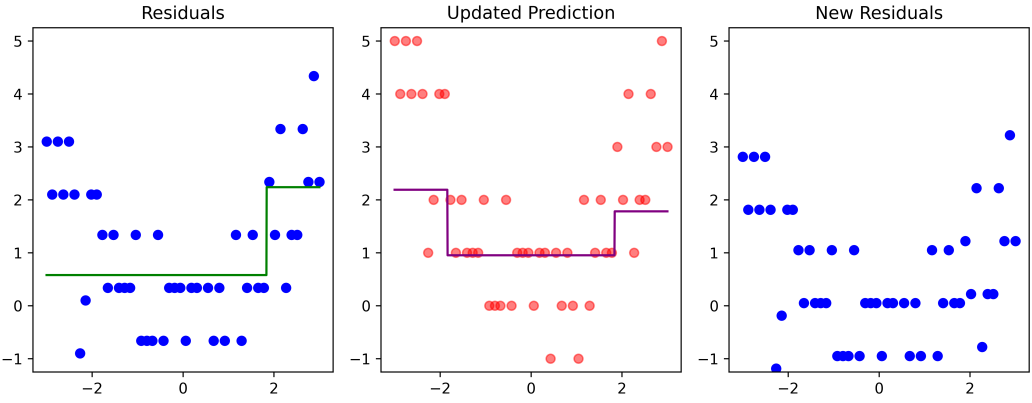
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x)$$

Boosting (iteration 1)



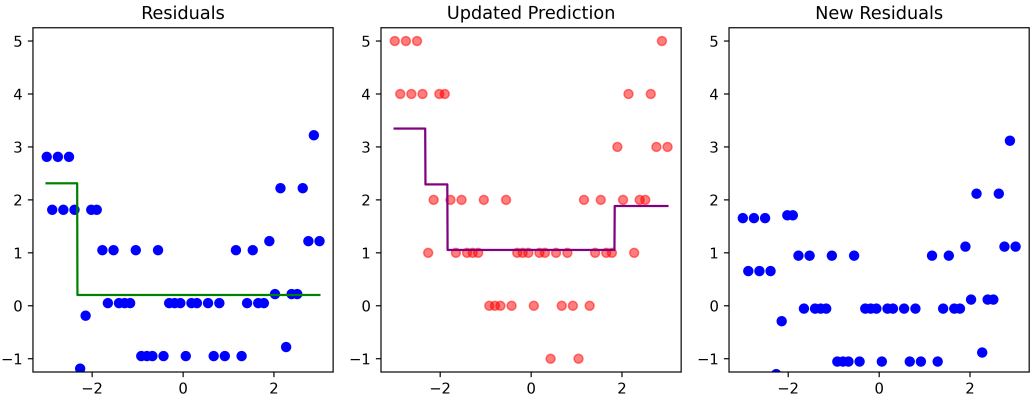
Here, $\lambda = \frac{1}{2}$ is the learning rate, and $d = 1$ is the number of splits in each tree (i.e., each tree is a stump).

Boosting (iteration 2)



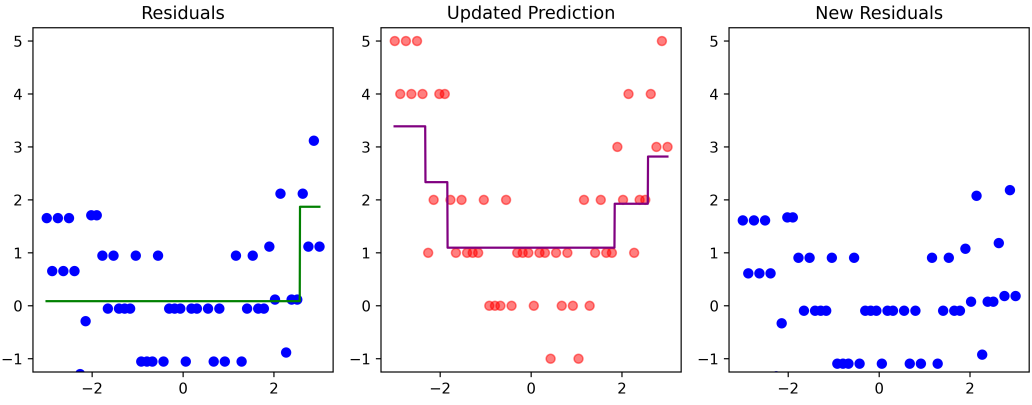
Here, $\lambda = \frac{1}{2}$ is the learning rate, and $d = 1$ is the number of splits in each tree (i.e., each tree is a stump).

Boosting (iteration 3)



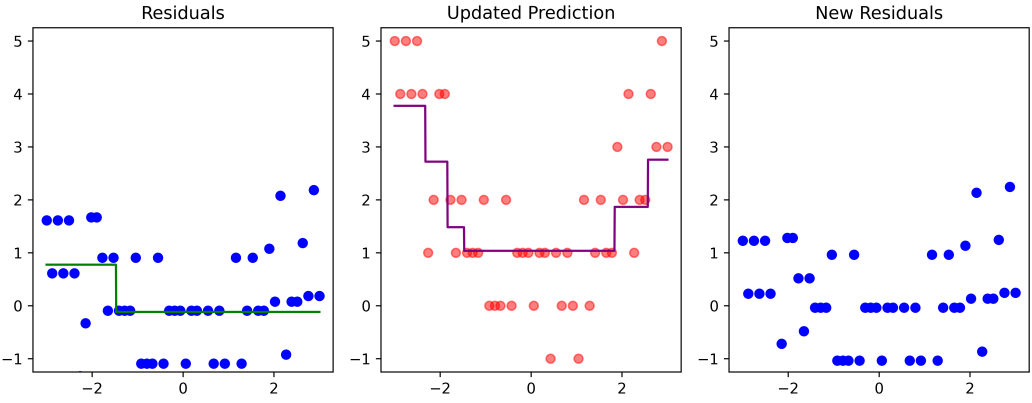
Here, $\lambda = \frac{1}{2}$ is the learning rate, and $d = 1$ is the number of splits in each tree (i.e., each tree is a stump).

Boosting (iteration 4)



Here, $\lambda = \frac{1}{2}$ is the learning rate, and $d = 1$ is the number of splits in each tree (i.e., each tree is a stump).

Boosting (iteration 5)

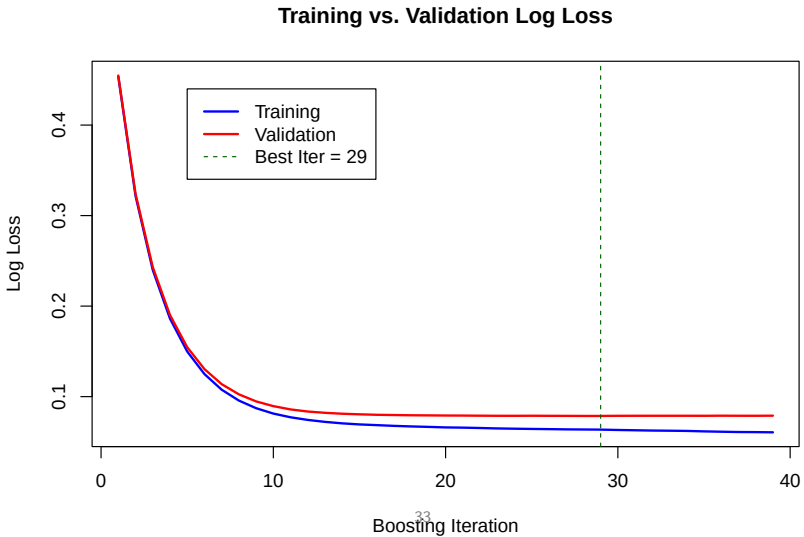


Here, $\lambda = \frac{1}{2}$ is the learning rate, and $d = 1$ is the number of splits in each tree (i.e., each tree is a stump).

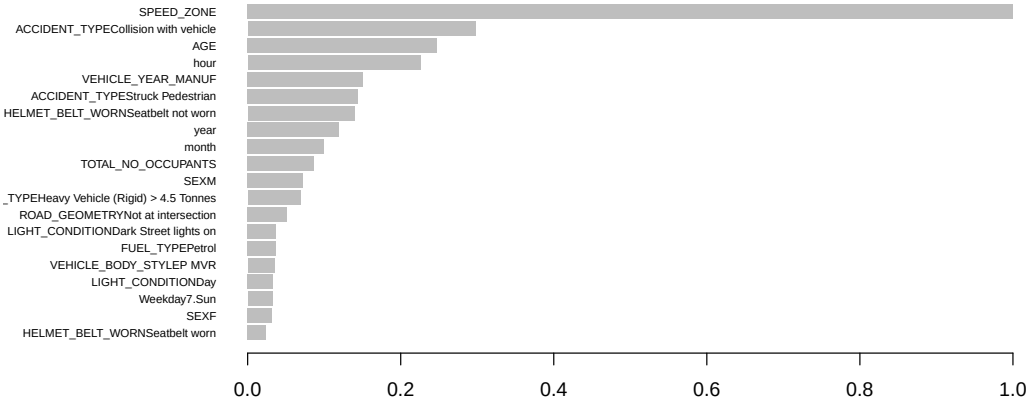
Boosting Tuning Parameters

- The number of trees B
  - overfit if B is too large
  - use cross-validation or validation set to select B
- The shrinkage parameter λ
  - a small positive number
  - controls the rate at which boosting learns
  - typical values are 0.01 or 0.001
- The number d of splits in each tree
  - $d = 1$ often works well, each tree is a stump

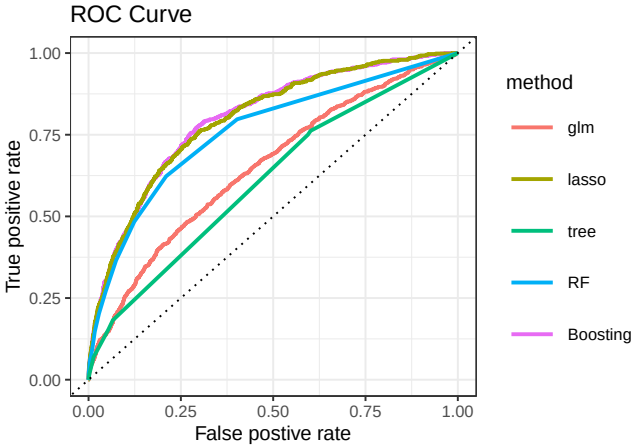
Boosting: VicRoads Data - number of trees B



Boosting: Variable Importance VicRoads Data



Comparison of methods: VicRoads Crash Data



AUC

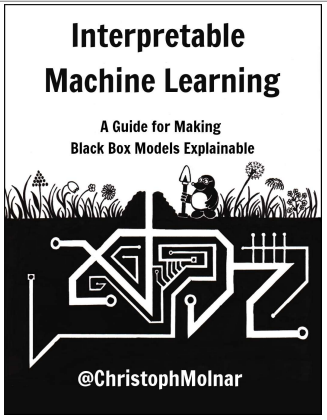
Method	Train	Test
glm	0.663	0.650
lasso	0.809	0.797
tree	0.629	0.610
RF	0.676	0.759
Boosting	0.8691	0.8045

Boosting in R

- `gbm`: Easy to use but slower and not available outside R.
- `xgboost`: Fast and powerful.
  - Popular choice for real-world **tabular data**.
  - R, Python, and other languages.
- `lightGBM`: Fastest and most memory-efficient.
  - Ideal for large datasets.
  - Developed by Microsoft,
  - Available in R, Python, and more.

Interpretability

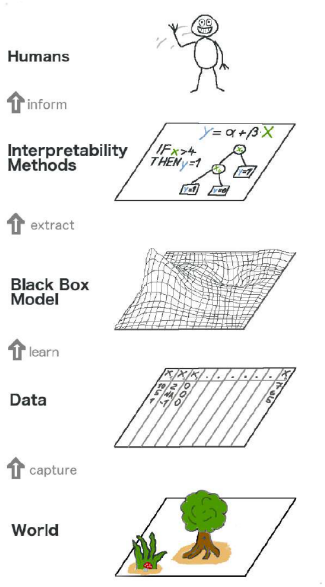
Interpretability: Resources



- This part of the discussion is based on this book
- Available at: https://christophm.github.io/interpretable-ml-book/

Interpretability: Motivation

- "Interpretability is the degree to which a human can understand the cause of a decision"
- Interpretable models
  - GLMs (linear regression, logistic regression)
  - Decision Trees
- Machine learning models improve prediction perfomance at the cost of interpretability
  - Black-box models



Partial Dependence Plots (PDPs)

- PDPs show the **marginal effect** one or two features have on the predicted outcome of a machine learning model.
- They help interpret whether the relationship between a feature and the target is **linear**, **monotonic**, or **nonlinear**.
- They average out the influence of all other features by marginalising over the data distribution.

Partial Dependence Plots (PDPs)

Integral (theoretical) form:

$$f_{\text{PDP}}(x_S) = \mathbb{E}_{X_C}[\hat{f}(x_S, X_C)] = \int \hat{f}(x_S, x_C)\, p(x_C)\, dx_C$$

Empirical (sample-based) form:

$$\hat{f}_{\text{PDP}}(x_S) = \frac{1}{n} \sum_{i=1}^{n} \hat{f}(x_S, x_C^{(i)})$$

- x_S: feature(s) of interest
- X_C: all other features
- $\hat{f}$: prediction function
- $p(x_C)$: marginal distribution of the complement features

Partial Depence Plots (pdps): Boosting VicRoads Crash Data

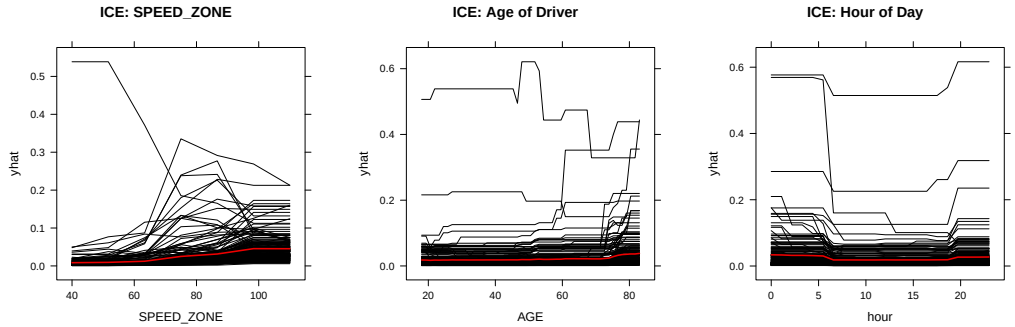
Implemented in `R` in the package `pdp` with the function `partial()`

Individual Conditional Expectation (ICE) Plots

- ICE plots show **one line per instance**, visualising how that instance's prediction changes as a single feature varies.
- ICE is like a **personalised PDP**: it captures the model's local behaviour for individual cases.
- PDPs average over all individuals and may **hide heterogeneous effects** caused by feature interactions.
- ICE plots are especially useful when:
  - There are **interactions** between the feature of interest and other features.
  - You want to understand **individual prediction dynamics**, not just global trends.

PDP = average of ICE curves across individuals

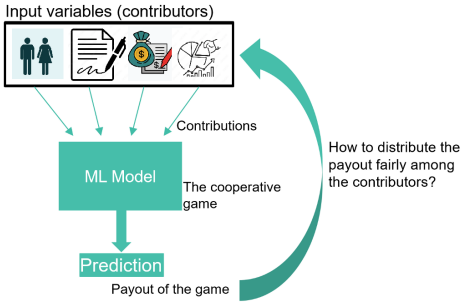
ICE plots: Boosting VicRoads Crash Data



Implemented in `R` in the package `pdp` with the function `partial()` and the argument `ice = TRUE`

Shapley Framework for Model Explanation

- Introduced by **Lloyd Shapley (1953)** in cooperative game theory.
- Allocates payouts to players based on their **marginal contributions** across all possible coalitions.
- In machine learning, Shapley values are used to **fairly attribute a model's prediction to input features**.



Game Theory	Model Explanation
Game	Prediction task
Players	Input features
Payout	Model prediction
Value of a player	Feature contribution (SHAP)

Shapley Value Formula

Given a model $\hat{f}$, the Shapley value ϕ_j for feature j is defined as:

$$\phi_j = \sum_{S \subseteq N \setminus \{j\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[\hat{f}(S \cup \{j\}) - \hat{f}(S)\right]$$

- N: the full set of input features
- S: a subset of features that does **not** include j
- $\hat{f}(S)$: the model prediction using only features in S
- ϕ_j: the **marginal contribution** of feature j, averaged across all possible subsets

The SHAP value is the **average added value** of feature j across all possible feature coalitions.

SHAP Values for a Given Instance

For a specific input x, the model prediction can be decomposed as:

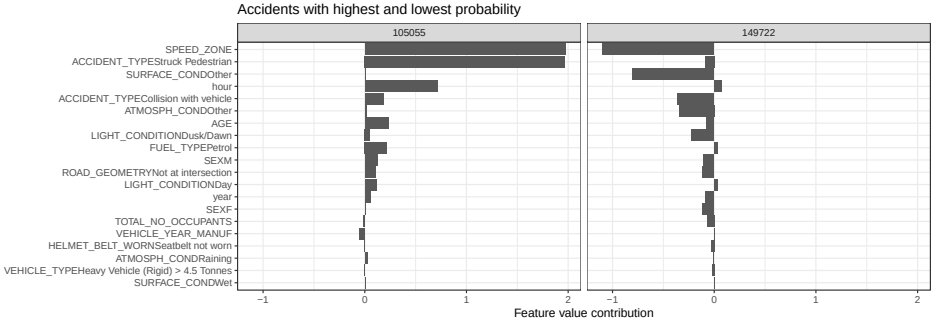
$$\hat{f}(x) = \phi_0 + \sum_{j=1}^{p} \phi_j(x)$$

- ϕ_0: the **baseline prediction**, usually the expected model output over the dataset
- $\phi_j(x)$: the **contribution of feature** j for this particular instance x

SHAP provides a **local explanation**: how much each feature j contributed to the prediction $\hat{f}(x)$, relative to the baseline.

In `R`, this is implemented for `xgboost` models in the `SHAPforxgboost` package.

SHAP values: Boosting VicRoads Crash Data



Variable	Max	Min
Probability	0.904114663600922	0.00032422156073153
SPEED_ZONE	110	50
ACCIDENT_TYPE	Struck Pedestrian	Collision with vehicle
hour	2	6
AGE	36	51
SEX	M	F

Session 3A - (Tree-based) Ensemble methods and Interpretability - Summary

- Bagging: Bootstrap aggregation
  - Reduces variance of a statistical learning method
- Random Forest: Decorrelates the bagged trees
  - Randomly selects predictors at each split
- Boosting: Sequentially builds trees
  - Very good performance on tabular data
- Interpretability
  - Partial dependence plots
  - ICE plots
  - SHAP values
- During lab, participant may follow the notebook VicRoads Crash - RF and Boosting