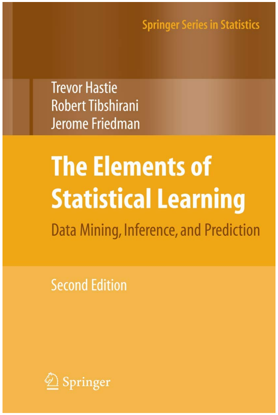
STA4026S – Honours Analytics
Section A – Theory and Application of Supervised Learning

Lecture 6 – Random Forests and Boosting

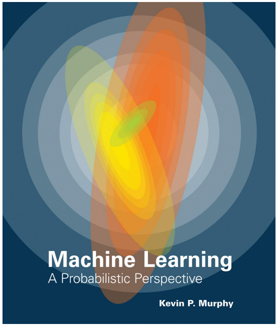
Stefan S. Britz
stefan.britz@uct.ac.za

Department of Statistical Sciences
University of Cape Town

Suggested Reading


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer

Chapter 15


Machine Learning
A Probabilistic Perspective
Kevin P. Murphy

Section 16.4

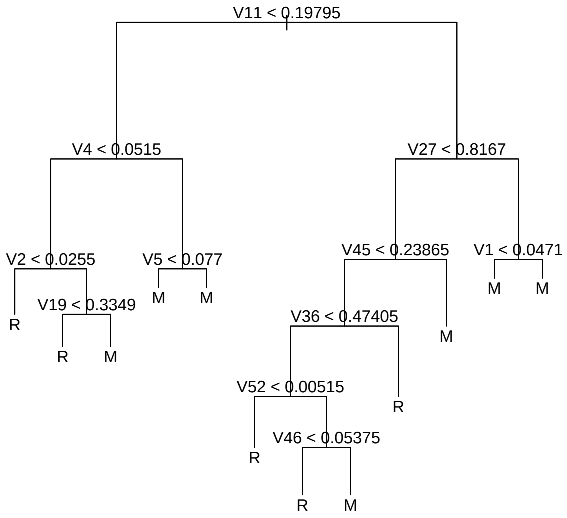
Decision Trees Recap

- Previously we considered single decision trees for both regression and classification purposes
- Although we saw that this method is straight-forward and yields nice, interpretable models...
- We also noted that it came at the cost of predictive accuracy
- The primary reason for this is that the decision trees suffer from **high sampling variability**
- If we were to take different samples from the same population and fit trees to each sample, the results could look quite different

High Variance of Decision Trees

```
library(mlbench)
library(tree)
data(Sonar)

run.plots <- function(iterations, pause){
 for (i in 1:iterations){
  train <- sample(1:nrow(Sonar), 0.7*nrow(Sonar))
  my_tree <- tree(formula = Class ~ ., data = Sonar, subset = train)

  yhat <- predict(my_tree, Sonar[-train,], type = 'class')
  y <- Sonar[-train, 'Class']
  accuracy <- sum(diag(table(y, yhat)))/length(y)*100

  plot(my_tree)
  text(my_tree, pretty = 0)
  mtext(paste('Classification accuracy: ', round(accuracy, 2), '%'), side = 1,
        padj = 3, cex = 1.5)
  Sys.sleep(pause)
 }
}

set.seed(1234)
run.plots(iterations = 20, pause = 1.5) # Set iterations & pause length
```

High Variance of Decision Trees



Classification accuracy: 73.02 %

High Variance of Decision Trees

- Here we see that the model's accuracy (or error) can vary extensively from one sample to the next
- This is in contrast to *low variance* procedures, such as linear regression, that yield similar results when applied to different samples from the same population
- We have already attempted to reduce the sampling variance of a tree by pruning
- It turns out that we can do better . . .

Bootstrap Aggregation / Bagging

Bagging

- Bootstrap aggregation, or **bagging**, is a general purpose procedure for reducing the variance of a statistical learning method
- Consider a set of n independent observations $Z_1, \ldots, Z_n$ each with variance σ^2. We have that $\mathrm{Var}\left(\bar{Z}\right) = \dfrac{\sigma^2}{n}$

  $\Rightarrow$ Averaging a set of observations reduces variance!

Bagging

- Bootstrap aggregation, or **bagging**, is a general purpose procedure for reducing the variance of a statistical learning method
- Consider a set of n independent observations $Z_1, \ldots, Z_n$ each with variance σ^2. We have that $\operatorname{Var}\left(\bar{Z}\right) = \frac{\sigma^2}{n}$

  $\Rightarrow$ Averaging a set of observations reduces variance!
- If we had B training sets, we could build a statistical model on each one and average the predictions on the test set:

$$\hat{y}_{\text{ave}} = \frac{1}{B} \sum_{b=1}^{B} \hat{y}_b$$

- The average prediction $\hat{y}_{\text{ave}}$ will have a lower sampling variance than the prediction $\hat{y}_b$ for any single training set

Bagging

- Of course, we only have one sample in practice
- But we can construct multiple bootstrap samples by repeated sampling **with replacement**
- We can then grow B **unpruned trees** on B bootstrap samples
- If the response variable is **continuous**, we simply average the B predictions of the regression tree for each observation
- For **categorical** response variables, we take a *majority vote*: the overall prediction for an observation is the most commonly occurring category among the B predictions
- Choose B to **minimise test error** (default in R is $B = 500$)
- A large value of B will **not** lead to overfitting

Bagging

Training data

	y	x1	x2
1	0	2.9	3.5
2	1	7.9	3.7
3	0	4.1	2.9
4	0	8.8	0.8
5	1	9.4	3.7

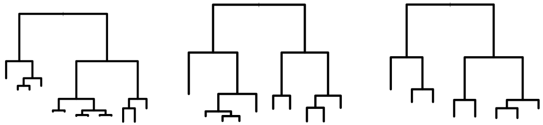
Bootstrap sampling

	y	x1	x2
3	0	4.1	2.9
2	1	7.9	3.7
3	0	4.1	2.9
4	0	8.8	0.8
5	1	9.4	3.7

	y	x1	x2
1	0	2.9	3.5
2	1	7.9	3.7
3	0	4.1	2.9
5	1	9.4	3.7
5	1	9.4	3.7

	y	x1	x2
4	0	8.8	0.8
2	1	7.9	3.7
5	1	9.4	3.7
4	0	8.8	0.8
5	1	9.4	3.7

Grow trees



Predict

obs	1	2	3	4	5
yhat	1	0	0	0	1

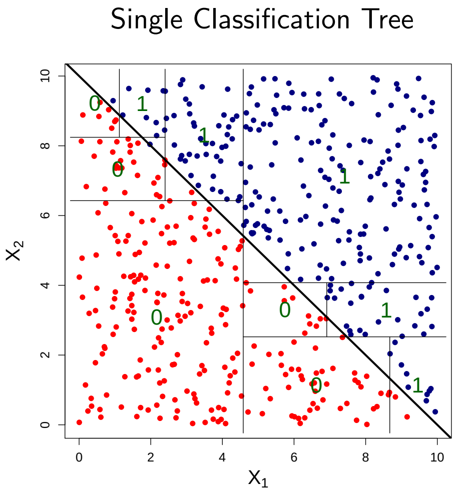
obs	1	2	3	4	5
yhat	0	1	0	0	1

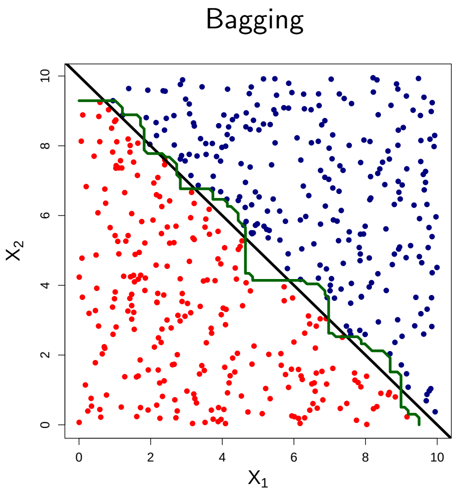
obs	1	2	3	4	5
yhat	1	1	0	0	1

Combine predictions

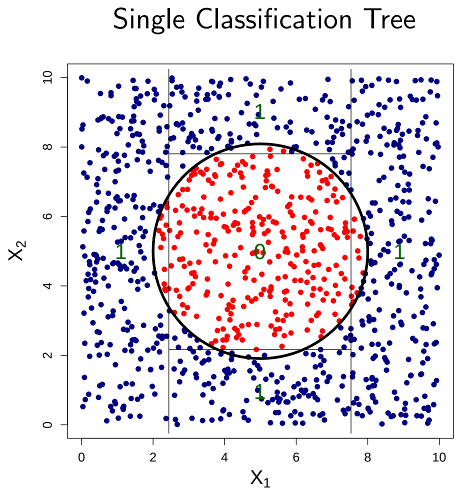
obs	1	2	3	4	5
yhat	1	1	0	0	1

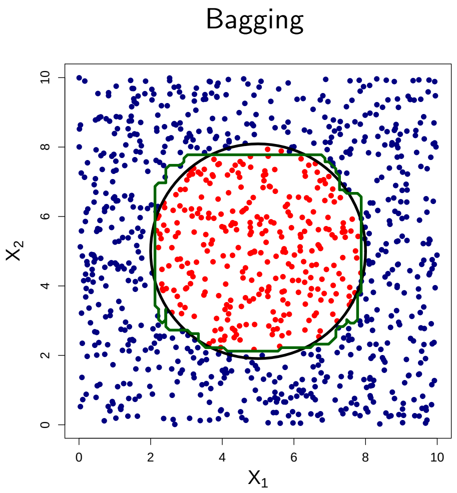
Decision Boundary Comparison





Decision Boundary Comparison





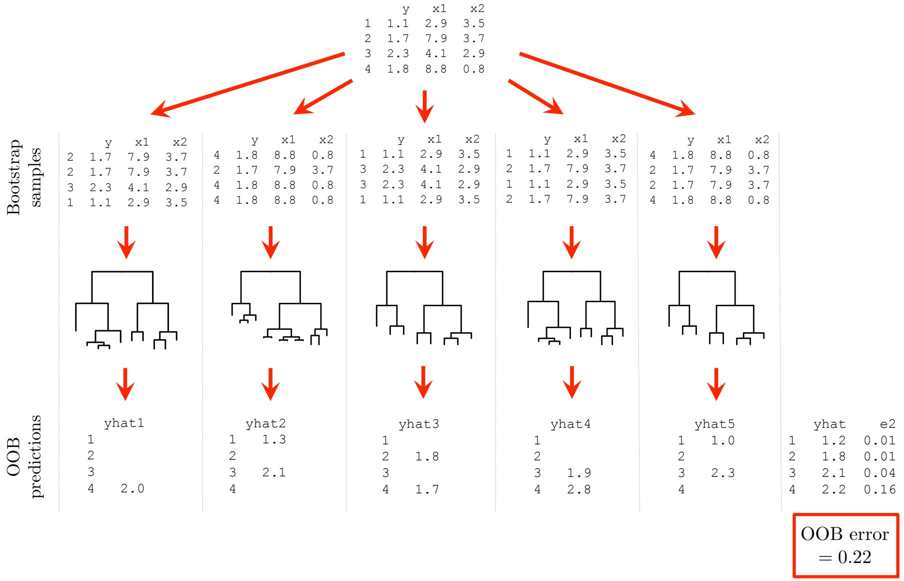
Out-of-Bag Error Estimation

- It is easy to estimate the test error of a bagged model **without** cross-validation
- It can be shown that each bagged tree uses approximately 2/3 of the observations on average
- The remaining 1/3 of observations not used to grow a bagged tree are called **out-of-bag (OOB)** observations

Out-of-Bag Error Estimation

- To obtain an estimate of the test error, we simply predict the response for observation i using each of the bagged trees for which that observation was OOB
- This will yield around $B/3$ predictions per observation, which are then combined into a single prediction per observation
- The out-of-sample RSS (regression trees) or classification error (classification trees) can then be computed
- Since this error rate is based on out-of-sample observations, it serves as an estimate of the test error, just like cross-validation errors

Out-of-Bag Error Estimation



```
     y   x1   x2
1  1.1  2.9  3.5
2  1.7  7.9  3.7
3  2.3  4.1  2.9
4  1.8  8.8  0.8
```

Bootstrap samples

```
     y   x1   x2
2  1.7  7.9  3.7
2  1.7  7.9  3.7
3  2.3  4.1  2.9
1  1.1  2.9  3.5
```

```
     y   x1   x2
4  1.8  8.8  0.8
2  1.7  7.9  3.7
4  1.8  8.8  0.8
4  1.8  8.8  0.8
```

```
     y   x1   x2
1  1.1  2.9  3.5
3  2.3  4.1  2.9
3  2.3  4.1  2.9
1  1.1  2.9  3.5
```

```
     y   x1   x2
1  1.1  2.9  3.5
2  1.7  7.9  3.7
1  1.1  2.9  3.5
2  1.7  7.9  3.7
```

```
     y   x1   x2
4  1.8  8.8  0.8
2  1.7  7.9  3.7
2  1.7  7.9  3.7
4  1.8  8.8  0.8
```

OOB predictions

```
  yhat1
1
2
3
4    2.0
```

```
  yhat2
1    1.3
2
3    2.1
4
```

```
  yhat3
1
2    1.8
3
4    1.7
```

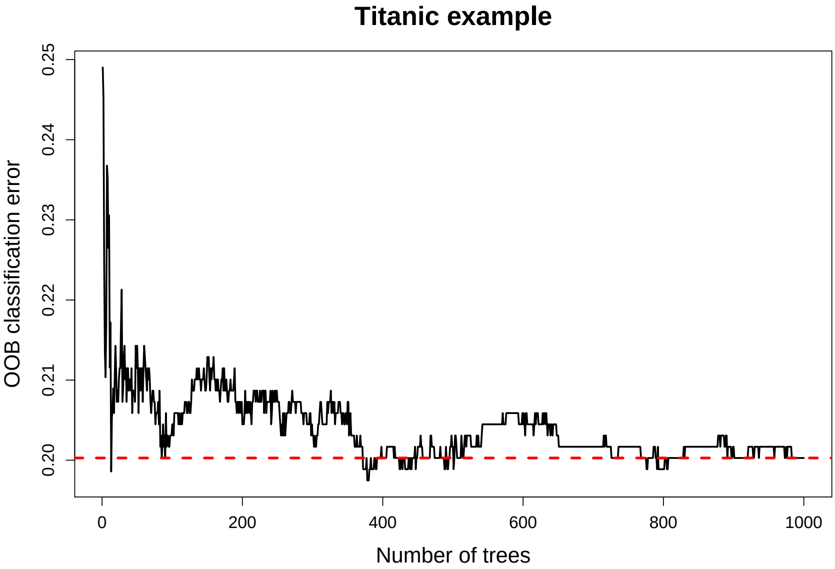
```
  yhat4
1
2
3    1.9
4    2.8
```

```
  yhat5
1    1.0
2
3    2.3
4
```

```
   yhat    e2
1   1.2  0.01
2   1.8  0.01
3   2.1  0.04
4   2.2  0.16
```

$\boxed{\text{OOB error} = 0.22}$

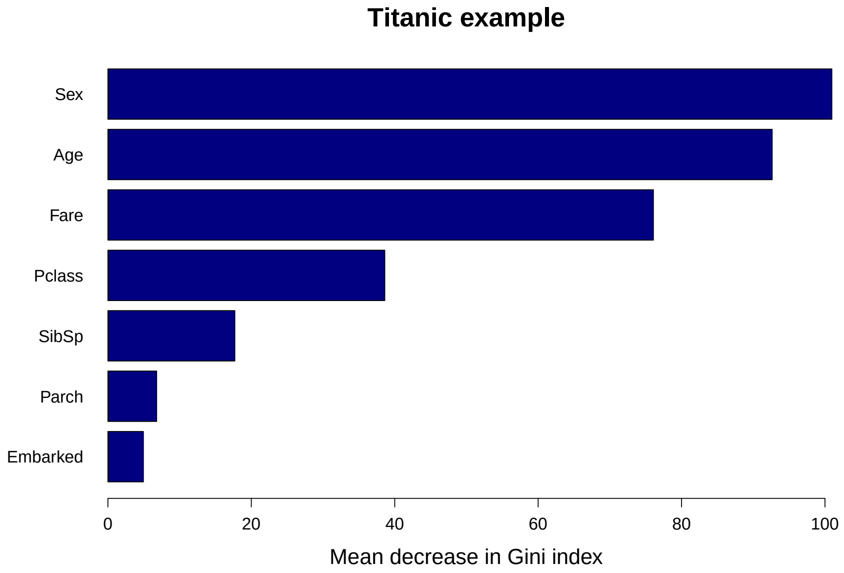
Example 7 – Titanic (continued)



Variable Importance

- The key advantage of a decision tree is ease of interpretation
- When we bag a large number of trees, it is no longer possible to represent the model with a single tree
- Improved prediction accuracy comes at the cost of reduced interpretability
- We can, however, provide a summary of the overall **importance** of each predictor
- For each predictor, we record the amount by which the splitting criterion is improved for each tree and take the average over all B trees
- A large **variable importance** value indicates an important predictor

Variable Importance



Random Forests

Motivation

- We have established that bagging reduces the sampling variability of predictions by averaging over many trees
- However, if the trees are highly **correlated**, this improvement will not be substantial
- Consider a set of correlated random variables $Z_1, \ldots, Z_n$ with common variance σ^2. We have that

$$\mathrm{Var}\left(\bar{Z}\right) = \frac{\sigma^2}{n} + \frac{2\sigma^2}{n^2} \sum_{i \neq j} \rho_{ij}$$

- **Random forests** provide an improvement over bagged trees by way of a small tweak that **decorrelates** the trees.

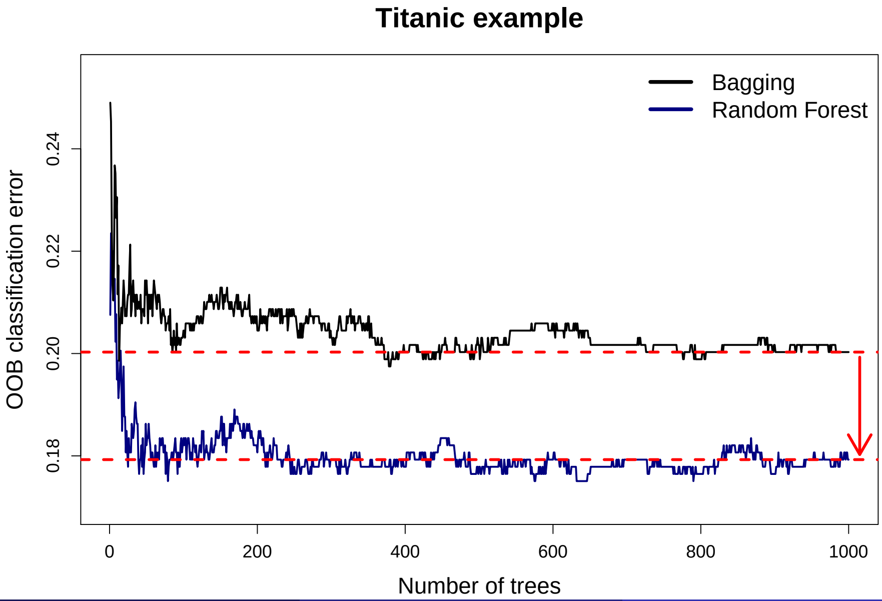
Random Forests

- Suppose that there is one very strong predictor in a dataset
- Most of the bagged trees will use this predictor
- Consequently, the bagged trees will look quite similar to each other and their predictions will therefore be highly correlated
- As with bagging, a **random forest** is constructed by growing B decision trees on B bootstrapped samples
- But, each time a split is considered, **a random sample of** $m < p$ **predictors** are chosen as split candidates

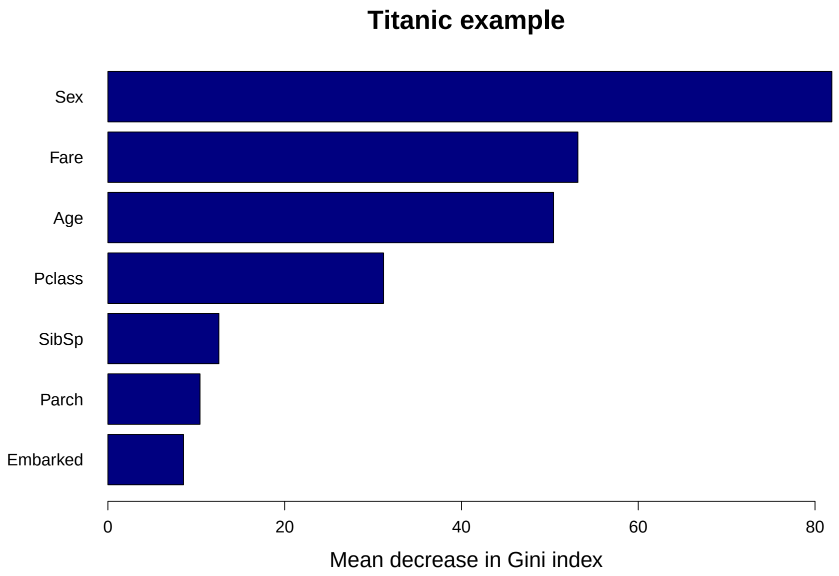
Random Forests

- Therefore, on average, $\frac{p-m}{p}$ of the splits will not even consider the strong predictor
- This has the effect of **decorrelating** the trees, so that the average predictions are less variable between samples and thus more reliable
- In the R package `randomForest` (amongst others), the default is $m \approx \sqrt{p}$ for classification trees and $m = \lfloor p/3 \rfloor$ for regression trees
- Note that bagging is a special case of a random forest with $m = p$
- As with bagging, we can estimate the testing error by means of the OOB errors
- Likewise, we can also report the importance of each predictor across all trees in the forest

OOB Errors Comparison



Variable Importance



Extending Random Forests

Although we have only covered the basic principle of random forests here, this methodology can be adapted and extended for more than just classification and regression:

- Anomaly detection using an Isolation Forest
- Feature selection – see the `VSURF` and `Boruta` packages in `R`
- Survival forests for analysing right-censored survival data

Extending Random Forests

Different variants and extensions are also constantly being developed, for example:

- Extremely Randomised Trees, or "Extratrees". Can increase prediction accuracy under certain conditions.
- Rotation Forests – first apply PCA before performing recursive partitioning
- "Deep Forests". Being touted as an alternative to Deep Neural Networks for similar tasks (primarily computer vision), these recently developed algorithms are usually based on Cascade Forests. This includes:
  - Dense Adaptive Cascade Forest – daForest
  - Multi-Grained Cascade Forest – gcForest
- These are still quite new and not widely tested. Explore at own risk!

Extending Random Forests

Several different frameworks also exist in `R` within which to implement the algorithm. These include, but are not limited to:

- The `ranger` and `Rborist` packages allow fast random forest implementation for high-dimensional (many variables) and large datasets (many rows) respectively
- `caret` allows easy grid searches across all combinations of multiple hyper-parameters. It also contains handy pre-processing functions

Finally, we note that random forests are perfectly parallelisable, allowing us to use multiple CPU cores to speed up implementation

Although one could parallelise the algorithm manually using `doParallel` and `foreach`, the above packages automatically use multiple cores

Extending Random Forests

Several different frameworks also exist in `R` within which to implement the algorithm. These include, but are not limited to:

- The `ranger` and `Rborist` packages allow fast random forest implementation for high-dimensional (many variables) and large datasets (many rows) respectively
- `caret` allows easy grid searches across all combinations of multiple hyper-parameters. It also contains handy pre-processing functions

Finally, we note that random forests are perfectly parallelisable, allowing us to use multiple CPU cores to speed up implementation

Although one could parallelise the algorithm manually using `doParallel` and `foreach`, the above packages automatically use multiple cores

Ensemble Methods

- We have now seen the benefit of combining many simple base models into a single, stronger prediction model
- This is known as **ensemble learning**
- It can be broken into two tasks:
    1. Develop a population of base learners from the training data
    2. Combine them to form the composite predictor
- We will now consider an ensemble method that combines a large number of weak learners to improve predictive accuracy: **Boosting**

Boosting

Boosting

- Boosting describes a general purpose family of algorithms that ensemble many weak learners into strong learners by letting the base models **sequentially** learn from the mistakes made by the previous ones
- This is in contrast to bagging and random forests, which grow all trees **independently**
- There are several boosting variations, the more popular ones being **AdaBoost** and **gradient boosting**
- Our application will entail the latter, which is a more generic and flexible algorithm
- The focus of our understanding will be on boosting for regression trees, but it can be applied to classification trees as well

Boosted Trees: Intuition

- The main idea of boosting is to let the algorithm gradually learn from its mistakes
- First, a base prediction is made – the **mean** of the target variable
- A regression tree with a small number of splits, say d, is grown **on the residuals** of this base prediction
- The (scaled) **residuals** of this tree are then treated as the response variable used to grow the next tree with d splits
- The (scaled) residuals of the new tree are then used to grow another tree with d splits, and so on...
- This yields a sequence of B trees, each one accounting for some of the variation in y that was not explained by the previous trees, gradually bringing the residuals down to zero (therefore eventually overfits!)

Boosted Trees: Intuition

- The initial trees will capture the strongest patterns in the data and will contribute the most to RSS reduction
- Subsequent trees will lead to smaller improvements in RSS
- If smaller trees are fitted, as determined by d, the model will improve slowly with subsequent trees
- In general, methods that **learn slowly** tend to perform well
- To slow the process down further, we introduce a **shrinkage parameter** λ that down-weights the contribution of each tree
- Since each tree will now contribute less toward the reduction in RSS, we will need to grow many more trees to explain the variation in the response variable
- This is called **slow learning**

Gradient Boosting for Regression Trees

Let $\hat{y}^{(b)}$ be the predictions after fitting b trees
Let $\hat{r}^{(b)}$ be the predicted values (of the residuals) of tree b
Let $r^{(b)}$ be the residuals after fitting b trees

1. Set $\hat{y}^{(0)} = \bar{y}$ such that $r^{(0)} = y - \bar{y}$
2. For $b = 1, \ldots, B$, repeat:
   a. Fit a regression tree with d splits to $(X, r^{(b-1)})$
   b. Using the predictions $\hat{r}^{(b)}$ of the current tree, update the predictions

      $$\hat{y}^{(b)} = \hat{y}^{(b-1)} + \lambda \hat{r}^{(b)}$$

      such that the updated residuals are

      $$r^{(b)} = r^{(b-1)} - \lambda \hat{r}^{(b)}$$

Gradient Boosting for Regression Trees (Continued)

3. Final predictions of the boosted model are

$$\hat{y} = y - r^{(B)} = \bar{y} + \lambda \sum_{b=1}^{B} \hat{r}^{(b)}$$

Note that this algorithm differs slightly from what is described in ISLR, in order to align with the most common Gradient Tree Boosting implementations

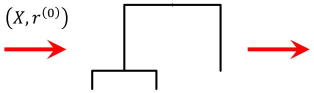
In the notes we confirm that this procedure is applied in the **gbm** package in `R`

Gradient Boosting for Regression Trees

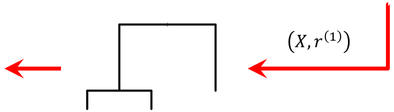
y	x_1	x_2
18	3	6
40	8	3
25	4	5
44	9	10
32	9	5
⋮	⋮	⋮

$\hat{y}^{(0)} = \bar{y} = 29$

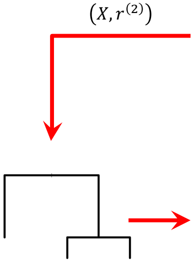
$r^{(0)}$
−11
11
−4
15
3



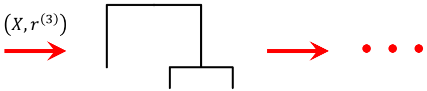
$\hat{r}^{(1)}$	$r^{(1)}$
−8.33	−10.93
4.70	10.94
−8.33	−3.93
14.19	14.85
14.19	2.85



$\hat{r}^{(2)}$	$r^{(2)}$
−8.25	−10.84
4.65	10.90
−8.25	−3.84
14.05	14.71
14.05	2.71



$\hat{r}^{(3)}$	$r^{(3)}$
−3.18	−10.81
7.24	10.82
−3.18	−3.81
7.24	14.64
7.24	2.64



Boosting algorithm with
$d = 2$ and $\lambda = 0.01$

$$\hat{y} = \bar{y} + \lambda \sum_{b=1}^{B} \hat{r}^{(b)}$$

Hyperparameters

Boosting has three hyperparameters:

1. The number of trees, B

   Unlike bagging and random forests, boosting can lead to overfitting if B is too large. We use cross-validation to select B.

2. The shrinkage parameter or learning rate, λ

   Typical values are 0.01 or 0.001 (default in R package `gbm`). Smaller values require more trees.

3. Number of splits in each tree, d

   d is also called the *interaction depth* of the boosted model, since d splits can involve at most d variables. Often $d = 1$ works well.

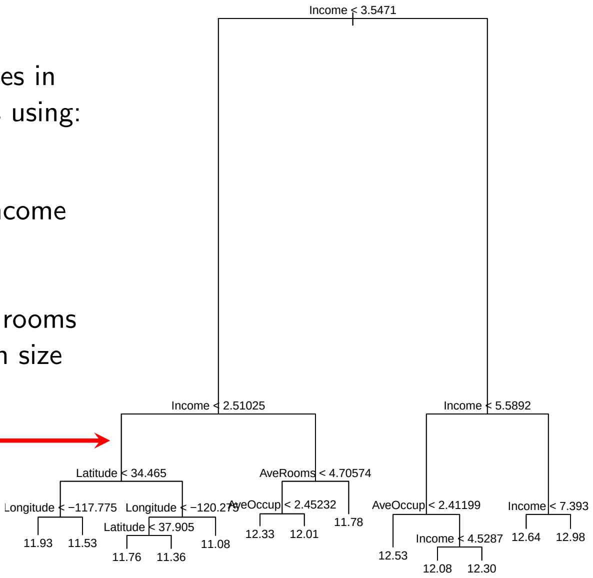
Example: California House Prices

<u>Aim:</u>
Predict median house prices in California neighbourhoods using:

Latitude and Longitude
Median Neighbourhood Income
Median House Age
Average Occupancy
Ave Num Rooms and Bedrooms
Neighbourhood population size

Best pruned single regression tree

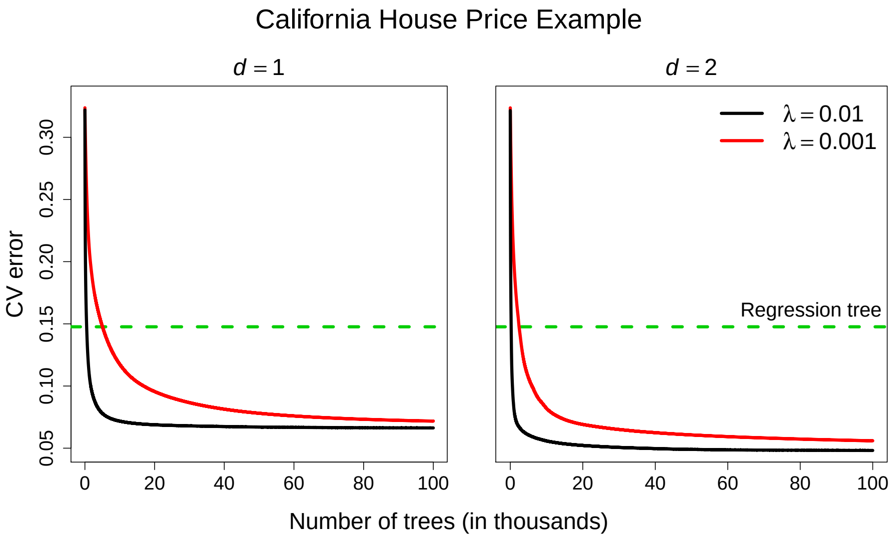
CV error = 0.149



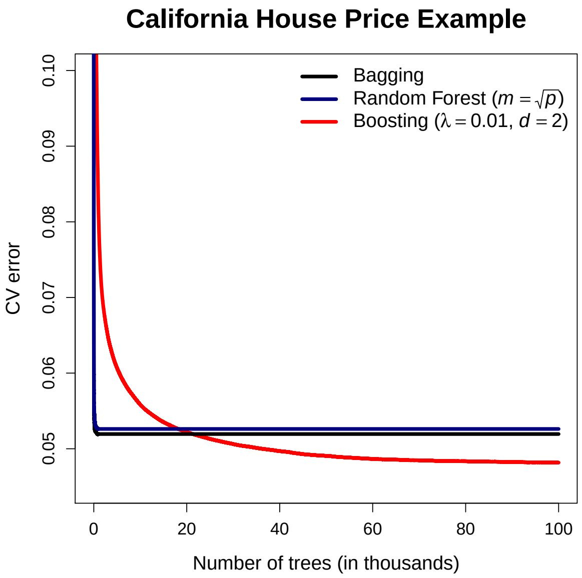
Cross-Validation of Tuning Parameters

- Using cross-validation, we can compare the effects of the hyperparameters on (estimated) predictive accuracy
- Specifically, we will consider different values for d and λ
- The CV error can now also be viewed as a function of the number of trees
- A grid search can be performed to find a preferable combination of hyperparameters for the problem at hand

Example 6 – California housing (continued)



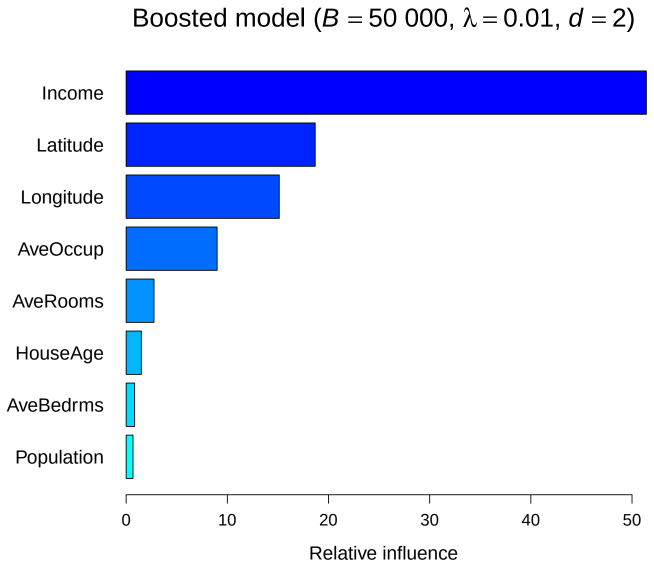
Example 6 – California housing (continued)



Variable Importance

- Variable importance is measured in a similar way to the bagged model
- Whenever a variable is selected at a split, the reduction in the loss function is determined
- The average reduction is then compared between variables (the default is usually to scale it such that the sum is 100)

Variable Importance



Partial Dependence Plots

- **Variable importance** plots tell us which features are most important for predicting the response
- However, they do not tell us *how* the predictors influence the predictions
- Unfortunately, visualisation of such relationships is limited to low-dimensional views
- To investigate how one predictor influences the predictions given a **fixed set of values for all other predictors**, we could plot $\hat{y}$ against X_i with $X_{-i} = x_{-i}$
- But this relationship may differ for other values of X_{-i}, especially if the model includes interactions

Partial Dependence Plots

- A better approach is to consider the average dependence of $\hat{y}$ on X_i, marginalising over all other predictors

$$\begin{aligned}\hat{y}(X_i) &= \mathrm{E}_{X_{-i}}[\hat{y}(X_1, \ldots, X_p)] \\ &= \int_{x_{-i}} \hat{y}(x_1, \ldots, x_p) f(x_{-i}) dx_{-i}\end{aligned}$$

- A plot of $\hat{y}(X_i)$ against X_i is called a **partial dependence plot**
- It represents the effect of X_i on the predictions after accounting for the effects of the other predictors
- The partial dependence function can be estimated as

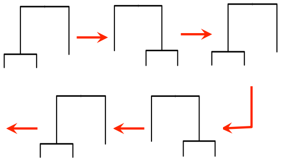
$$\hat{y}(X_i) = \frac{1}{n} \sum_{j=1}^{n} \hat{y}(X_i, x_{-i,j})$$

  This must be done for each value of X_i and can be computationally intensive, even for moderately-sized datasets

Constructing Partial Dependence Plots

```
        y    x1   x2   x3
1  35.70  2.17 1.77 5.78
2  52.28  2.42 5.63 6.46
3  38.18  0.78 2.74 4.36
4  35.99  0.09 3.04 3.45
5  21.19  2.21 0.50 3.40
6  54.38 -2.64 3.63 6.81
7  23.59  2.26 0.23 4.52
8  32.27  0.45 1.34 5.62
9  47.84  0.43 4.24 5.61
10 38.87 -0.84 2.60 4.84
```

Data



Predictive Model

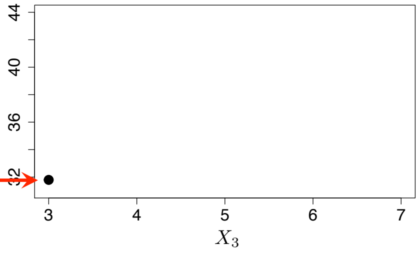
Construct a partial dependence plot for X_3

Constructing Partial Dependence Plots

```
       y    x1   x2   x3  yhat
1  35.70  2.17 1.77    3  25.87
2  52.28  2.42 5.63    3  42.86
3  38.18  0.78 2.74    3  32.48
4  35.99  0.09 3.04    3  34.93
5  21.19  2.21 0.50    3  20.10
6  54.38 -2.64 3.63    3  41.99
7  23.59  2.26 0.23    3  18.80
8  32.27  0.45 1.34    3  26.70
9  47.84  0.43 4.24    3  39.79
10 38.87 -0.84 2.60    3  34.44
```

$\hat{y}(X_3 = 3, X_{-3} = x_{-3,j})$

$\hat{y}(X_3 = 3) =$ 31.80

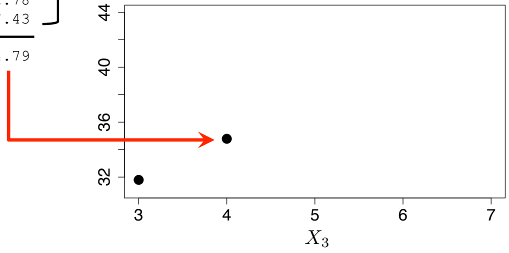


Constructing Partial Dependence Plots

```
       y    x1   x2   x3   yhat
1  35.70  2.17 1.77    4  28.86
2  52.28  2.42 5.63    4  45.85
3  38.18  0.78 2.74    4  35.47
4  35.99  0.09 3.04    4  37.93
5  21.19  2.21 0.50    4  23.09
6  54.38 -2.64 3.63    4  44.98
7  23.59  2.26 0.23    4  21.79
8  32.27  0.45 1.34    4  29.69
9  47.84  0.43 4.24    4  42.78
10 38.87 -0.84 2.60    4  37.43
```

$\hat{y}(X_3 = 4, X_{-3} = x_{-3,j})$

$\hat{y}(X_3 = 4) =$ 34.79

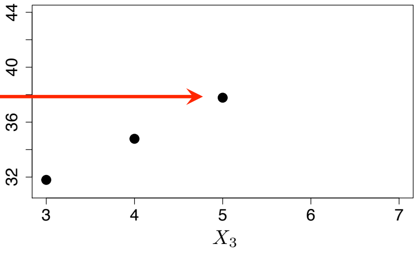


Constructing Partial Dependence Plots

```
       y    x1   x2   x3   yhat
1  35.70  2.17 1.77    5  31.85
2  52.28  2.42 5.63    5  48.84
3  38.18  0.78 2.74    5  38.47
4  35.99  0.09 3.04    5  40.92
5  21.19  2.21 0.50    5  26.08
6  54.38 -2.64 3.63    5  47.98
7  23.59  2.26 0.23    5  24.78
8  32.27  0.45 1.34    5  32.69
9  47.84  0.43 4.24    5  45.77
10 38.87 -0.84 2.60    5  40.42
```

$\hat{y}(X_3 = 5, X_{-3} = x_{-3,j})$

$\hat{y}(X_3 = 5) =$ 37.78

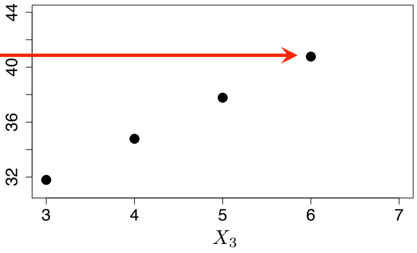


Constructing Partial Dependence Plots

```
       y    x1   x2   x3   yhat
1  35.70  2.17 1.77    6  34.84
2  52.28  2.42 5.63    6  51.84
3  38.18  0.78 2.74    6  41.46
4  35.99  0.09 3.04    6  43.91
5  21.19  2.21 0.50    6  29.07
6  54.38 -2.64 3.63    6  50.97
7  23.59  2.26 0.23    6  27.77
8  32.27  0.45 1.34    6  35.68
9  47.84  0.43 4.24    6  48.77
10 38.87 -0.84 2.60    6  43.42
```

$\hat{y}(X_3 = 6, X_{-3} = x_{-3,j})$

$\hat{y}(X_3 = 6) =$ 40.77

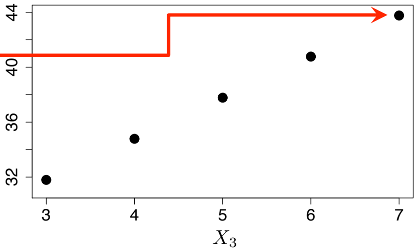


Constructing Partial Dependence Plots

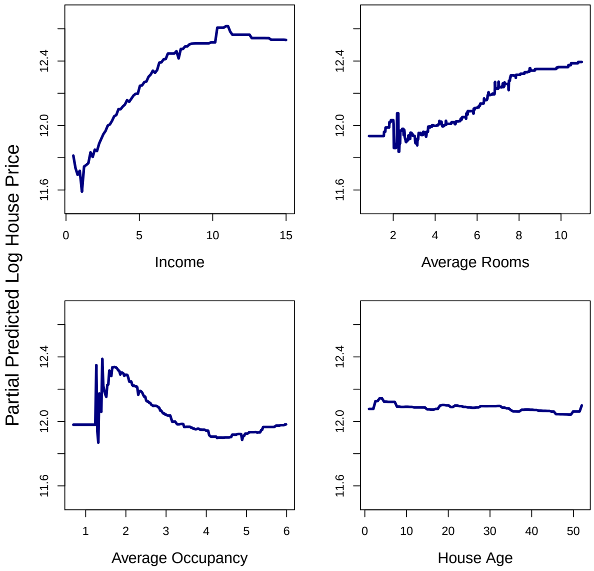
```
       y     x1   x2   x3   yhat
1  35.70  2.17 1.77    7  37.84
2  52.28  2.42 5.63    7  54.83
3  38.18  0.78 2.74    7  44.45
4  35.99  0.09 3.04    7  46.90
5  21.19  2.21 0.50    7  32.07
6  54.38 -2.64 3.63    7  53.96
7  23.59  2.26 0.23    7  30.77
8  32.27  0.45 1.34    7  38.67
9  47.84  0.43 4.24    7  51.76
10 38.87 -0.84 2.60    7  46.41
```

$\hat{y}(X_3 = 7, X_{-3} = x_{-3,j})$

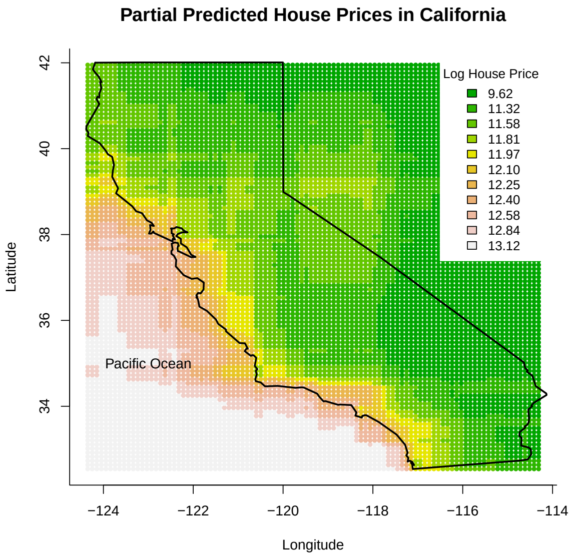
$\hat{y}(X_3 = 7) =$ 43.76



Example 6 – California housing (continued)



Example 6 – California housing (continued)



XGBoost

- General frameworks for gradient boosted models exist in `R`, although the most common application thereof is on trees
- One such package is **gbm** (Generalized Boosted Modelling)
- A more recent, very powerful implementation of the gradient boosted method currently widely in use is **Extreme Gradient Boosting** (XGBoost)
- This method performs especially well in structured/tabular data competitions like Kaggle
- Here we only summarise how the algorithm builds on gradient boosting; the details can be found in this paper

XGBoost

Some of the key differences between XGBoost and regular gradient boosted trees are:

1. **Regularisation**: XGBoost incorporates L_1 and L_2 regularisation terms into its objective function to prevent overfitting. This helps in controlling the complexity of the learned trees and can be tuned using hyperparameters.

2. **Newton boosting**: Computes the second derivative (Hessian) of the loss function with respect to the predicted scores, allowing for more precise updates to the tree structure and yielding faster convergence

3. **Tree pruning**: Trees are grown to a maximum depth and then pruned iteratively, which helps in reducing overfitting and computational complexity

XGBoost

4. **Parallel processing**: It is designed for parallel processing, making it faster and more scalable
5. **Subsampling**: Row subsampling helps prevent overfitting; column subsampling adds extra randomisation, similar to random forests
6. **Missing values**: It automatically learns how to partition data points with missing values and assign them to the appropriate child nodes during tree construction

Conclusion

- As with many ML algorithms, the application in R is relatively straight-forward
- The skill lies in understanding what the different arguments control, and how to adjust them
- The course notes contain extensive examples of how to apply these methods, predominantly through the `caret` framework
- Make sure you go through these examples! You will have opportunity to practice it yourself in the assignment

That concludes our journey through the theory and application of supervised learning. Next up – Neural Networks!