

MA5832: Data Mining & Machine Learning

Collaborate Week 3: Tree Based Methods

Martha Cooper, PhD

JCU Masters of Data Science

2021-09-14

Housekeeping

- Collaborates = **Thursdays 6-7:30pm**

For my Collaborate Sessions, you can get the **slides & R code** for each week on Github:

<https://github.com/MarthaCooper/MA8532>



Today's Goals

- Regression Trees
- Decision Trees
- Bagging
- Random Forest
- Boosting

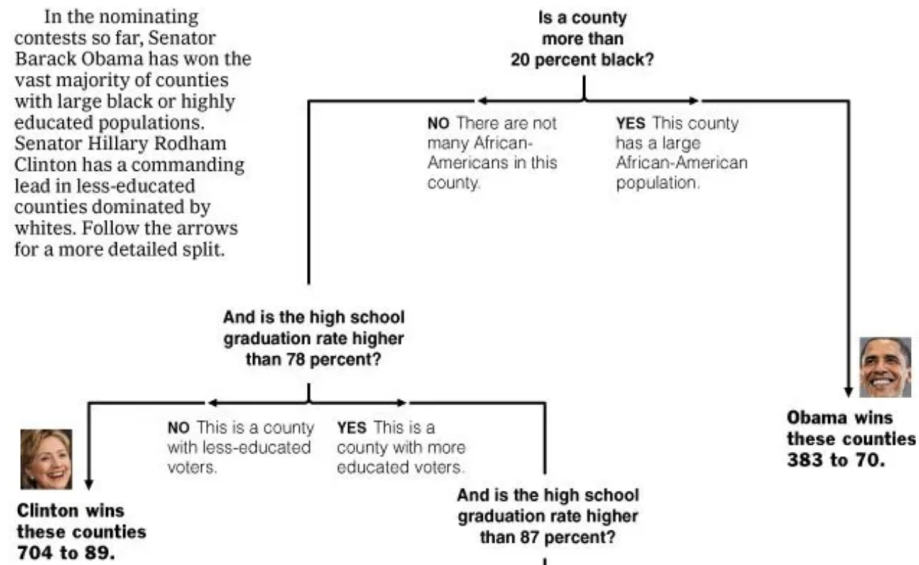
Tree Based Methods

Tree Based Methods

- Split samples into groups
- Test homogeneity or purity of each group
- Split again

Decision Tree: The Obama-Clinton Divide

In the nominating contests so far, Senator Barack Obama has won the vast majority of counties with large black or highly educated populations. Senator Hillary Rodham Clinton has a commanding lead in less-educated counties dominated by whites. Follow the arrows for a more detailed split.



New York Times

Jeff Leek's Practical Machine Learning

Tree Based Methods

- CART = classification & regression trees
- Stratifying or segmenting predictor space into a number of simple regions
- The set of splitting rules used to segment predictor space can be summarized as a *tree* or dendrogram.
- Trees are simple and easy to interpret, but have poor predictive accuracy.
- Bagging, Random Forest and Boosting are all approaches that combine multiple trees together to yield a consensus prediction. This improves predictive accuracy at the cost of interpretation.

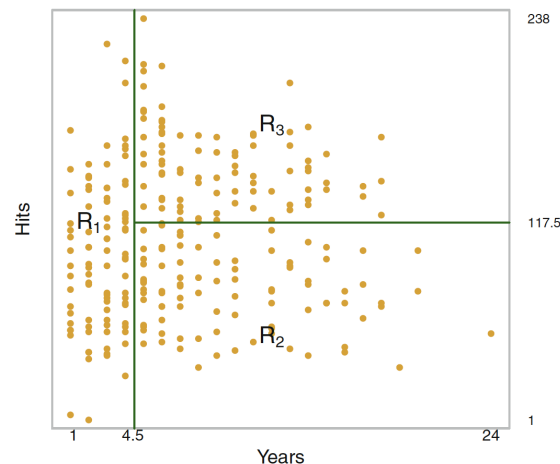
Regression Trees

- Predicting a continuous response
- Predicting baseball player's salaries using the number of years playing major league baseball and the number of hits made in the previous year.

Regression Tree



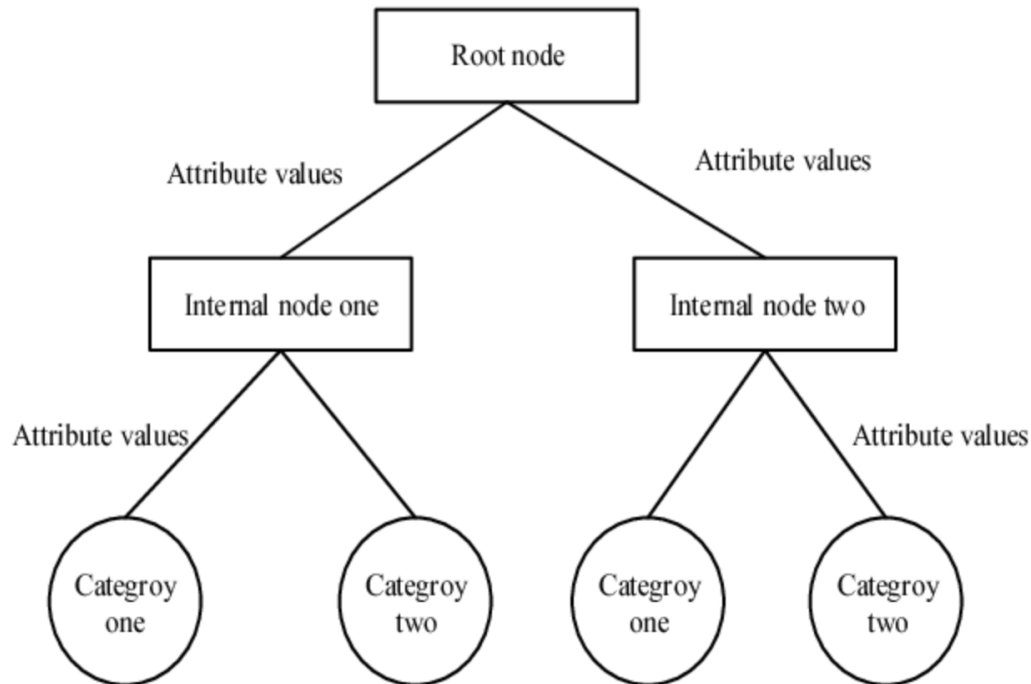
Three-region partition



Predicted salaries are given by the **mean value** for players in each region.

Tree terminology

- **Root node**: no incoming edge, zero or more outgoing edges
- **Internal node**: one incoming edge, two (or more) outgoing edges
- **Leaf or Terminal node**: each leaf node is assigned a class label, contain a response variable, y
- **Parent and Child nodes**: If a node is split, we refer to that node as a parent node, and the resulting nodes are called child nodes.
- **Branches**: Segments that connect the nodes.



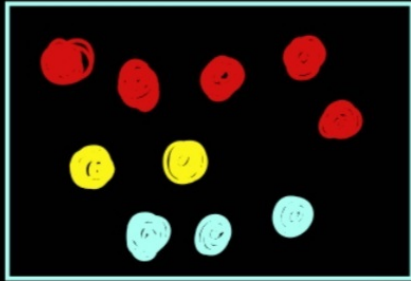
Building a regression tree

1. Divide the predictor space into j distinct and non-overlapping regions R_1, R_2, \dots, R_j
2. For every observation in region R_j we make the same prediction - the mean on the response values for training observations
 - Regions R_1, R_2, \dots, R_j are high-dimensional rectangles or *boxes*
 - Loss function: $\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$
 - where \hat{y}_{R_j} is the mean response for observation in the j th region.
 - **Greedy** algorithm - looks for best split at each step.

Classification Trees

- Predicting a qualitative response
- Different loss functions to measure class impurity or at each node, m
- Gini Index: $1 - \sum_{k=1}^K (\hat{p}_{mk})^2$
- Cross entropy: $-\sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$
- where $\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k)$, or the proportion of observations in the m th region that are from the k th class

An example



Class	# observations	$\hat{p}_{m c}$	Gini
Red	5	0.5	} 0.68 ↑
Yellow	2	0.2	
Blue	3	0.3	

$$1 - (0.5^2 + 0.2^2 + 0.3^2)$$

CART in R

```
library(ggplot2) #for plotting
library(caret) #for trees
library(rattle) #plotting the tree

data(iris) #get data
head(iris) #view data
table(iris$Species) #groups to classify
set.seed(6)
test_index <- createDataPartition(iris$Species, p = 0.3, list = F)
training <- iris[-test_index, ] #training and test splits
testing <- iris[test_index, ]
set.seed(6)
fit1 <- train(Species ~., method = "rpart", data = training) #fit model
fancyRpartPlot(fit1$finalModel) #plot
preds <- predict(fit1, newdata = testing) #make predictions
table(preds, testing$Species) #compare preds
```

CART Summary

- Non-linear models
- Easy to interpret
- High variance - sensitive to small changes in training data and not very accurate at making predictions

Bagging, Random Forest and Boosting

- Making more powerful prediction models

Bagging

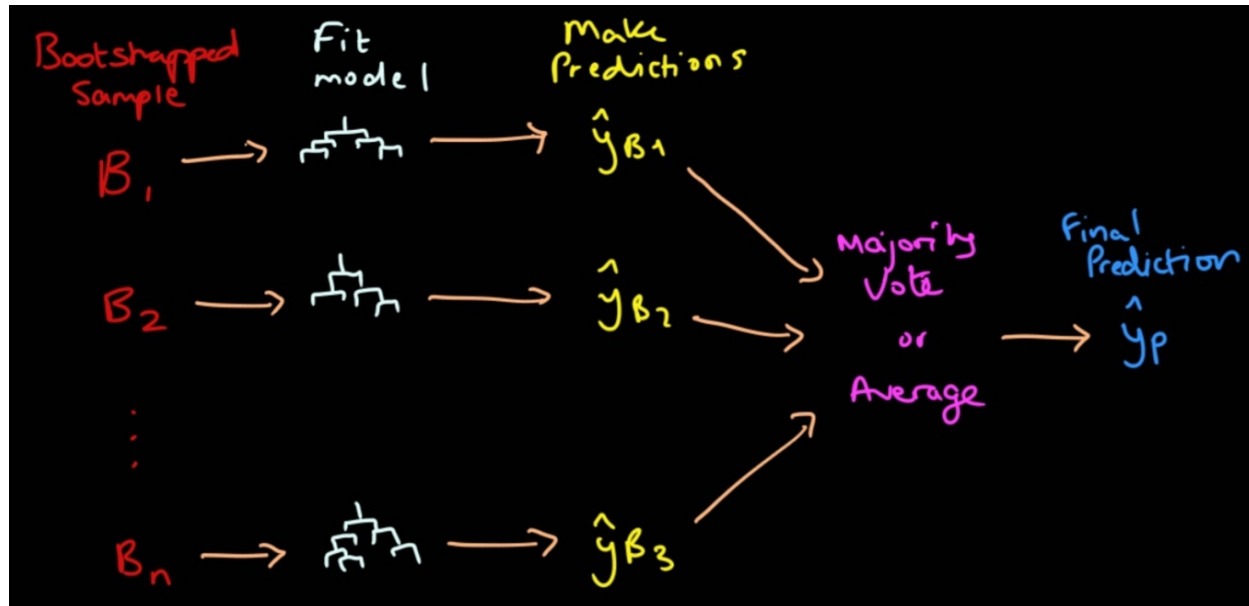
Bootstrap Aggregation = Bagging

1. *Bootstrap* the samples = draw many random samples with replacement
2. Refit the model (note: *any* model) to the bootstrapped samples and **aggregate** the results between all models.

Original Data	x_1	x_2	x_3	x_4	x_5
Bootstrap #1	x_3	x_3	x_5	x_1	x_2
Bootstrap #2	x_4	x_2	x_1	x_2	x_3
...					
Bootstrap #n	x_5	x_1	x_1	x_2	x_5

Bagging Tree

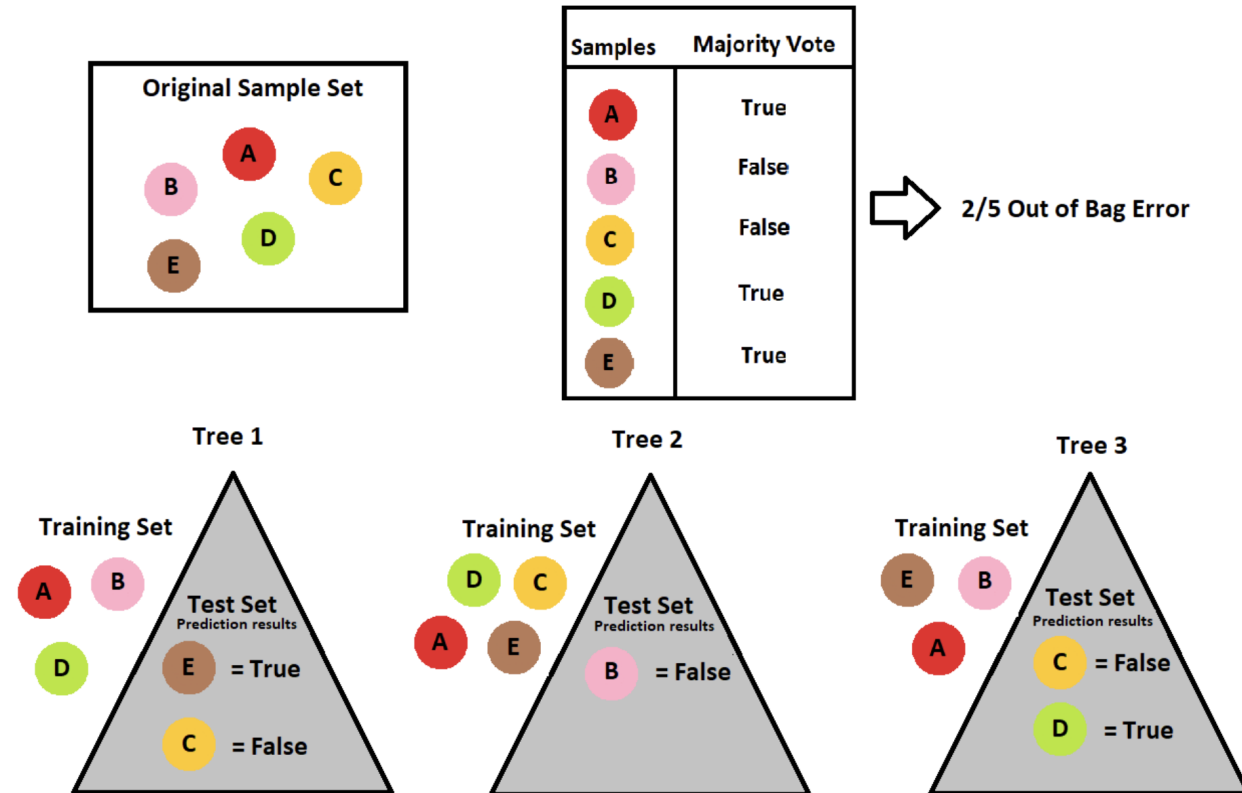
- Bagging can improve the accuracy of unstable models that tend to overfit



Out-of-Bag Error Estimation

- Estimating the error of a bagged model
- Each bagged tree uses $\sim 2/3$ rd of the training data
- The remaining $\sim 1/3$ rd are called *Out-of-Bag* observations
- We can predict the response for the i th observation using each of the trees for which that sample was *OOB*
- We then take the majority vote/ average result to get a prediction for that sample.
- Compile results to get overall estimate of OOB error.

Out-of-Bag Error Estimation



OOB Error

Bagging in R

```
set.seed(6)
fit2 <- ipred::bagging(Species~., data = training, coob = T, nbagg = 10)
preds <- predict(fit2, testing)
table(preds, testing$Species)
```

Random Forests

- Similar to bagging trees
- At each split, a random sample of m predictors is chosen as split candidates from the full set of P predictors
- We can select a value of m to start (e.g. \sqrt{P}), and tune by cross-validation
- Why?

Random Forests

- Similar to bagging trees
- At each split, a random sample of m predictors is chosen as split candidates from the full set of P predictors
- We can select a value of m to start (e.g. \sqrt{P}), and tune by cross-validation
- Why? *Decorrelates* the trees

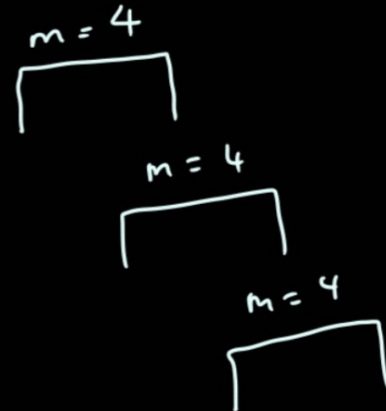
Random Forests

Say we have 12 predictors :

- Each bagged tree uses 12 predictors



- Each tree in the Random Forest uses $\sim m = \sqrt{P}$



Random forest in R

```
tc <- trainControl(method = "repeatedcv", number = 10, repeats = 3) #set
tg <- expand.grid(mtry = seq(2,ncol(training)-2,1)) #set up tune grid
set.seed(6)
fit3 <- train(Species ~., method = "rf", data = training, trControl = tc)
fit3$finalModel
randomForest::getTree(fit3$finalModel, k = 3) #get individual trees
preds <- predict(fit3, testing) #make predictions
table(preds, testing$Species) #compare predictions
plot(varImp(fit3, scale = F)) #variable importance
```

Boosting

Note: *not just for trees*, but we will focus on trees here

- Combine weak learners to create strong learners. A weak learning is a classifier that performs poorly, but is still better than a random guess.
- Weak the weak learners and add them up
- Sequential - each tree is grown using information from the previous tree. This is an iterative process, where samples in the training set are re-weighted at each iteration, so that the second iteration can correct errors from the model in the first iteration. Trees are added until the training set is predicted perfectly

e.g. AdaBoost & Gradient Boosting

Adaboost for Decision Trees

1. Combine weak learners (small trees or stumps) to create strong learners.
2. Each individual tree is weighted based on how well it classifies samples
3. Each sequential tree learns from the previous tree's mistake by adjusting the weight of samples. If the observation is misclassified by the previous tree, the weight of the observation increases; conversely, the weight decreases if correctly classified. The updated weights are then used in the next iteration for tree fitting.

Adaboost for Decision Trees

Height	Weight	Sex	Initial Weight	Predicted	New Weight
1.65	50	1 (M)	1/6	0	↑
1.82	72	1 (M)	1/6	1	↓
1.70	80	1 (M)	1/6	1	↓
1.55	65	1 (M)	1/6	0	↓
1.75	55	0 (F)	1/6	0	↑
1.68	54	1 (M)	1/6	0	↑

Weak learner:

Height < 1.70

F (0) M (1)

① Compute error:

$$\frac{w_1 \times 1 + w_2 \times 0 + w_3 \times 0 + w_4 \times 0 + w_5 \times 0 + w_6 \times 1}{\sum w_i} = \frac{1}{3}$$

$$\textcircled{2} \alpha_m = \log \frac{(1 - \text{err}_m)}{\text{err}_m} = \log \left(\frac{1 - 1/3}{1/3} \right)$$

$$\textcircled{3} \text{ Update weights } w_i = w_i \exp(\alpha_m \cdot I(y_i \neq G_m(x_i)))$$

Gradient boosting

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \dots, B$, 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 shrunk version of the new tree:

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \quad (8.10)$$

- (c) Update the residuals,

$$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \quad (8.11)$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^B \lambda \hat{f}^b(x). \quad (8.12)$$

Boosting in R

- `gbm`
- `xgboost`

Homework to try this in R...

Extra reading

- Chapter 8 ISLR
- Chapters 9.2 & 10 ESL
- StatQuest!

References

- Chapter 8 ISLR
- Jeff Leek's Practical Machine Learning

Slides

- xaringhan, xaringantheme, remark.js, knitr, R Markdown