CSCE 633: Machine Learning

Lecture 18: Random Forest and Boosting

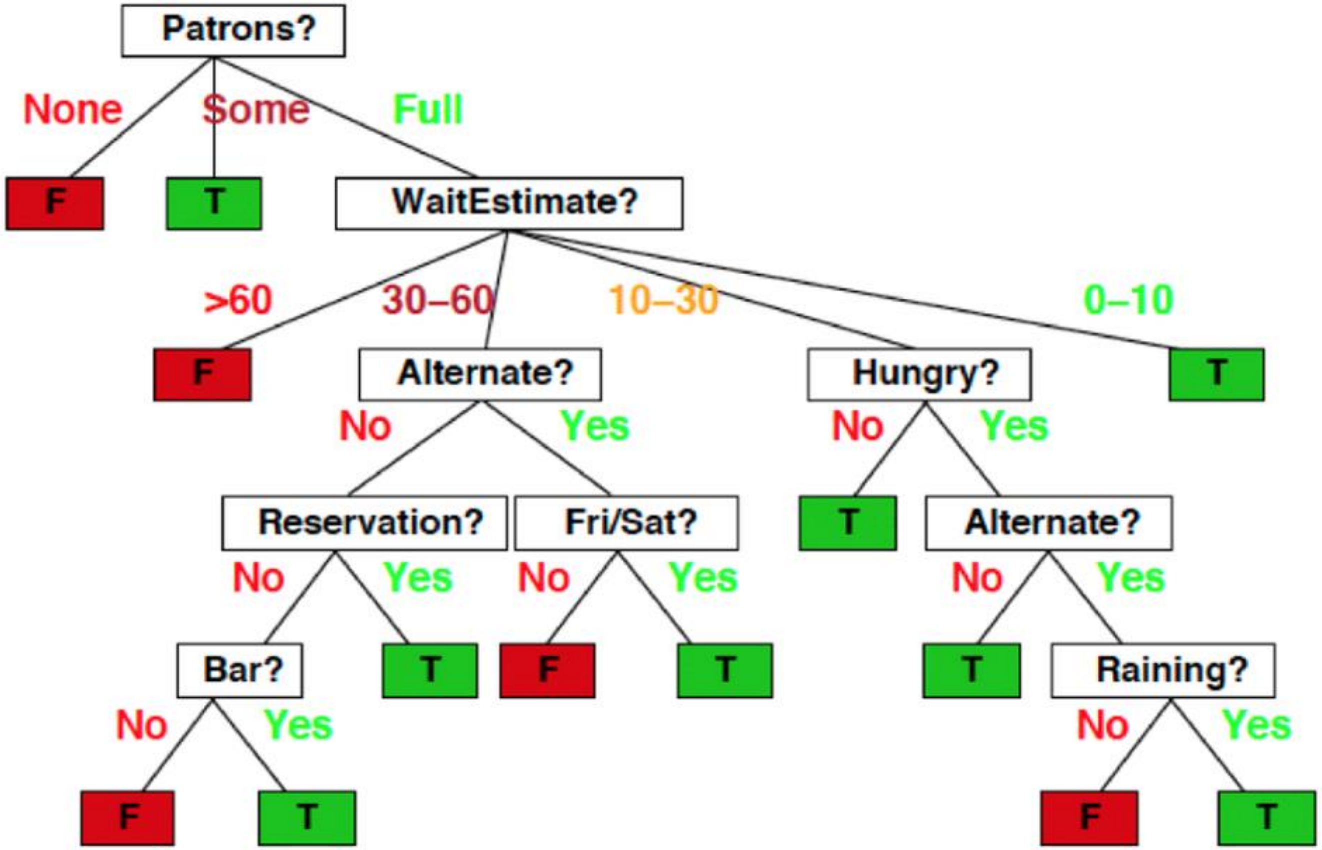
Texas A&M University
Bobak Mortazavi

Goals

- Review of decision trees
- Overcoming the limitations to decision trees
- Introduce random forests
- Introduce gradient descent boosting

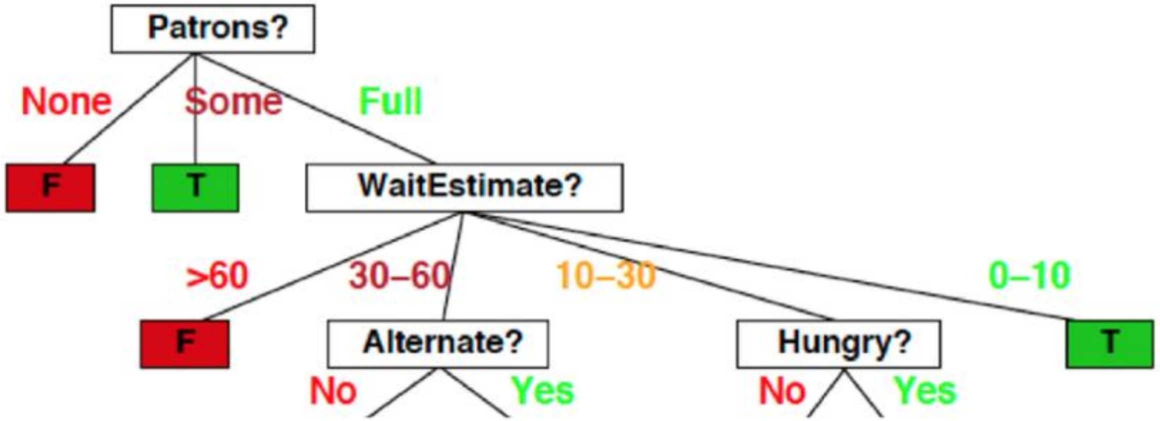
Final decision tree

Greedily we build the tree and looks like this



Pruning classification trees

We should prune some of the leaves of the tree to get a smaller depth



- If we stop here, not all training samples are classified correctly
- How do we classify a new instance?
  - We label the leaves of this smaller tree with the label of the majority of training samples

Decision trees vs. other models

- Advantages:
  - Models are transparent: easily interpretable!
  - Data can contain combination of feature types: Qualitative predictors without dummy variables
  - Decision trees more closely mirror human decision making
  - Graphical representation
- Disadvantages:
  - Usually not same level of predictive accuracy
  - Not robust (small change in the data can change the tree a lot)

Bagging: Bootstrapped Aggregating

- Accuracy of Decision Trees suffer from high variance
- For example if we split data in half, tree for both halves could be very different

Tree 1: Restaurants

Tree 2: Restaurants

Bagging: Why does it work?

- Given n independent observations $Z_1, \ldots, Z_n$ each with variance σ^2
- Variance of the mean $\bar{Z} = \frac{\sigma^2}{n}$
- That's lower variance!
- So, how do we take advantage of this?

Bagging Trees

- Take B different training sets
- Train f^1 on training set 1
- Train f^2 on training set 2
- …

Bagging Trees

- Take B different training sets
- Train f^1 on training set 1
- Train f^2 on training set 2
- …

- Can average the result over B trees, as a single, low-variance model

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

Bagging Trees

- Take B different training sets
- Train f^1 on training set 1
- Train f^2 on training set 2
- …

- Can average the result over B trees, as a single, low-variance model

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

- But where do we come up with B training sets??

Bootstrapping

- Take B different bootstraps of our one datasets
- $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B*}\hat{f}^{b}(x)$
- Turns out, you can grow these trees without pruning them
- For Regression – average the values from each tree
- For Classification – take the majority vote across trees
- Test error can be plotted as a function of B

Bootstrapping

- Take B different bootstraps of our one datasets
- $\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B*}\hat{f}^{b}(x)$
- Turns out, you can grow these trees without pruning them
- For Regression – average the values from each tree
- For Classification – take the majority vote across trees
- Test error can be plotted as a function of B

- Great Fact: B turns out not to be a critical parameter, so large B does not mean we overfit!

Measuring Error

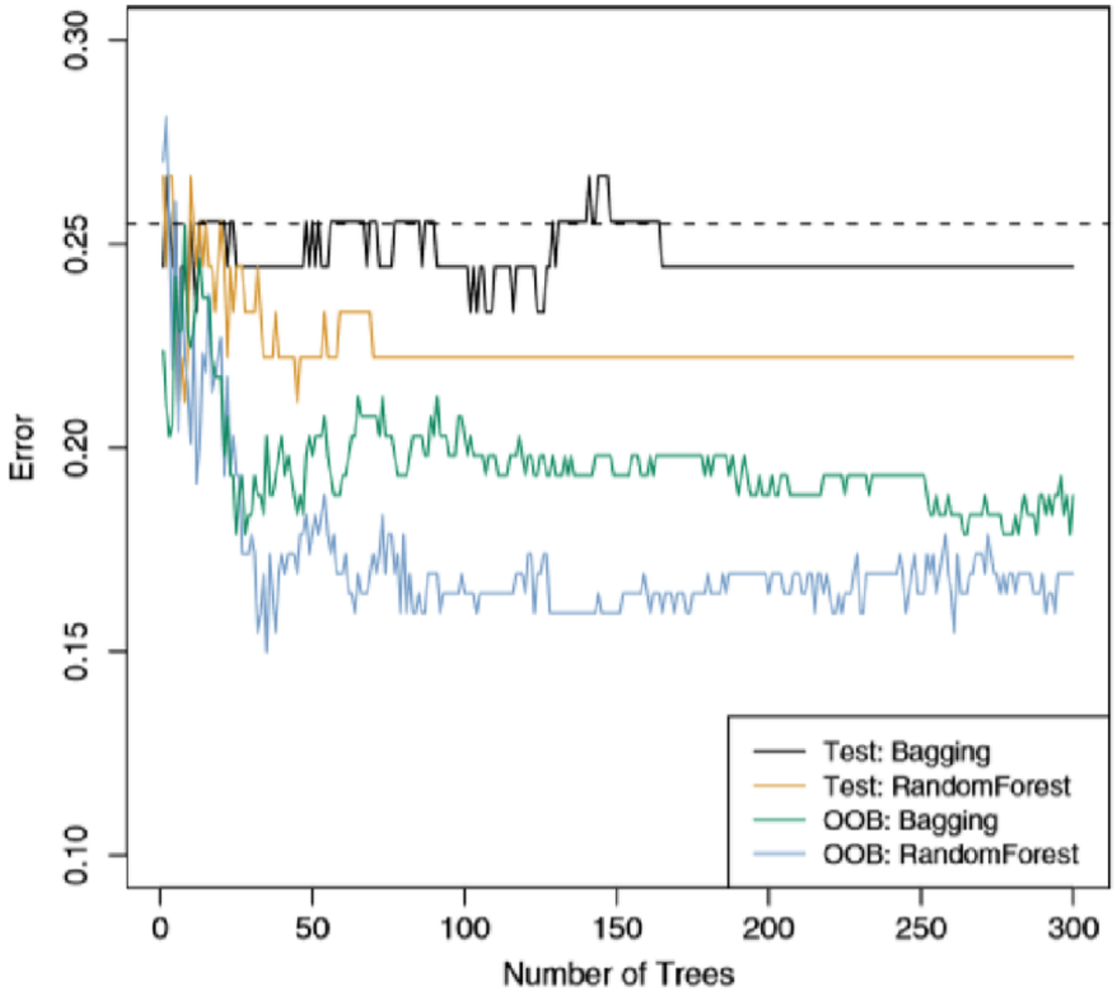
- If we repeatedly fit bootstrapped subsets (say 2/3) of our data
- Each time we will be left with a subset (say 1/3) that we can call out of bag
- We can then estimate the error for this as we train – we call this the Out of Bag Estimation

New Example: Heart Dataset
https://archive.ics.uci.edu/ml/datasets/heart+Disease

```
> summary(data)
       X              Age             Sex                 ChestPain       RestBP           Chol            Fbs        
 Min.   :  1.0   Min.   :29.00   Min.   :0.0000   asymptomatic:144   Min.   : 94.0   Min.   :126.0   Min.   :0.0000  
 1st Qu.: 76.5   1st Qu.:48.00   1st Qu.:0.0000   nonanginal  : 86   1st Qu.:120.0   1st Qu.:211.0   1st Qu.:0.0000  
 Median :152.0   Median :56.00   Median :1.0000   nontypical  : 50   Median :130.0   Median :241.0   Median :0.0000  
 Mean   :152.0   Mean   :54.44   Mean   :0.6799   typical     : 23   Mean   :131.7   Mean   :246.7   Mean   :0.1485  
 3rd Qu.:227.5   3rd Qu.:61.00   3rd Qu.:1.0000                      3rd Qu.:140.0   3rd Qu.:275.0   3rd Qu.:0.0000  
 Max.   :303.0   Max.   :77.00   Max.   :1.0000                      Max.   :200.0   Max.   :564.0   Max.   :1.0000  

    RestECG           MaxHR           ExAng           Oldpeak         Slope             Ca                 Thal    
 Min.   :0.0000   Min.   : 71.0   Min.   :0.0000   Min.   :0.00   Min.   :1.000   Min.   :0.0000   fixed     : 18  
 1st Qu.:0.0000   1st Qu.:133.5   1st Qu.:0.0000   1st Qu.:0.00   1st Qu.:1.000   1st Qu.:0.0000   normal    :166  
 Median :1.0000   Median :153.0   Median :0.0000   Median :0.80   Median :2.000   Median :0.0000   reversable:117  
 Mean   :0.9901   Mean   :149.6   Mean   :0.3267   Mean   :1.04   Mean   :1.601   Mean   :0.6722   NA's      :  2  
 3rd Qu.:2.0000   3rd Qu.:166.0   3rd Qu.:1.0000   3rd Qu.:1.60   3rd Qu.:2.000   3rd Qu.:1.0000                    
 Max.   :2.0000   Max.   :202.0   Max.   :1.0000   Max.   :6.20   Max.   :3.000   Max.   :3.0000                    
                                                                                  NA's   :4                         
  AHD     
 No :164  
 Yes:139  
```

Comparison of Algorithms



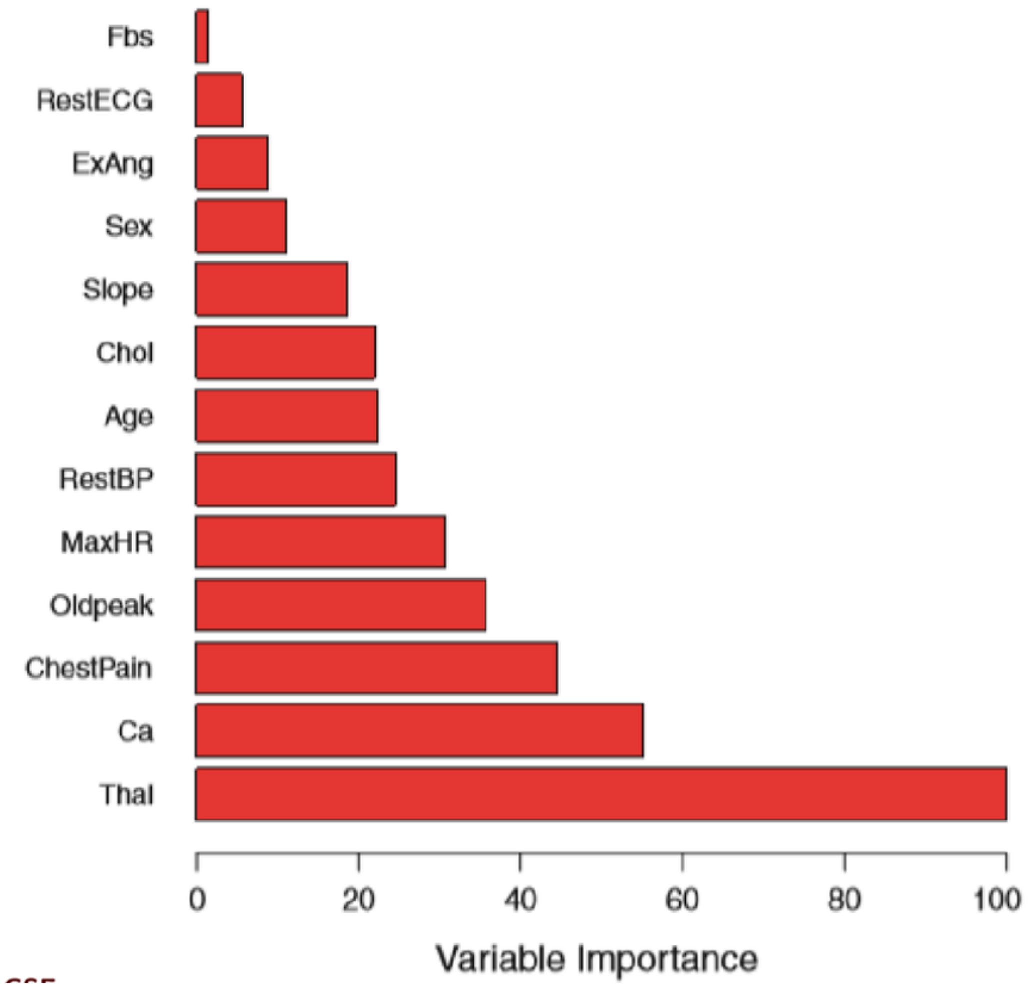
Variable Importance: Multiple Trees

Variable Importance

- Interpreting Bagging is difficult
- No longer possible to decide variable order from a single tree
- With regression trees – must understand summary reduction in RSS at each split
- With classification trees – overall summary in reduction in Gini Index or Entropy at each split
- Can create a new term: Relative Importance

$$v_j = \frac{1}{M} \sum_{m=1}^{M} I\,(j \in T_m)$$

Heart Dataset Variable Importance



Bagging Limitations: Strong Predictor

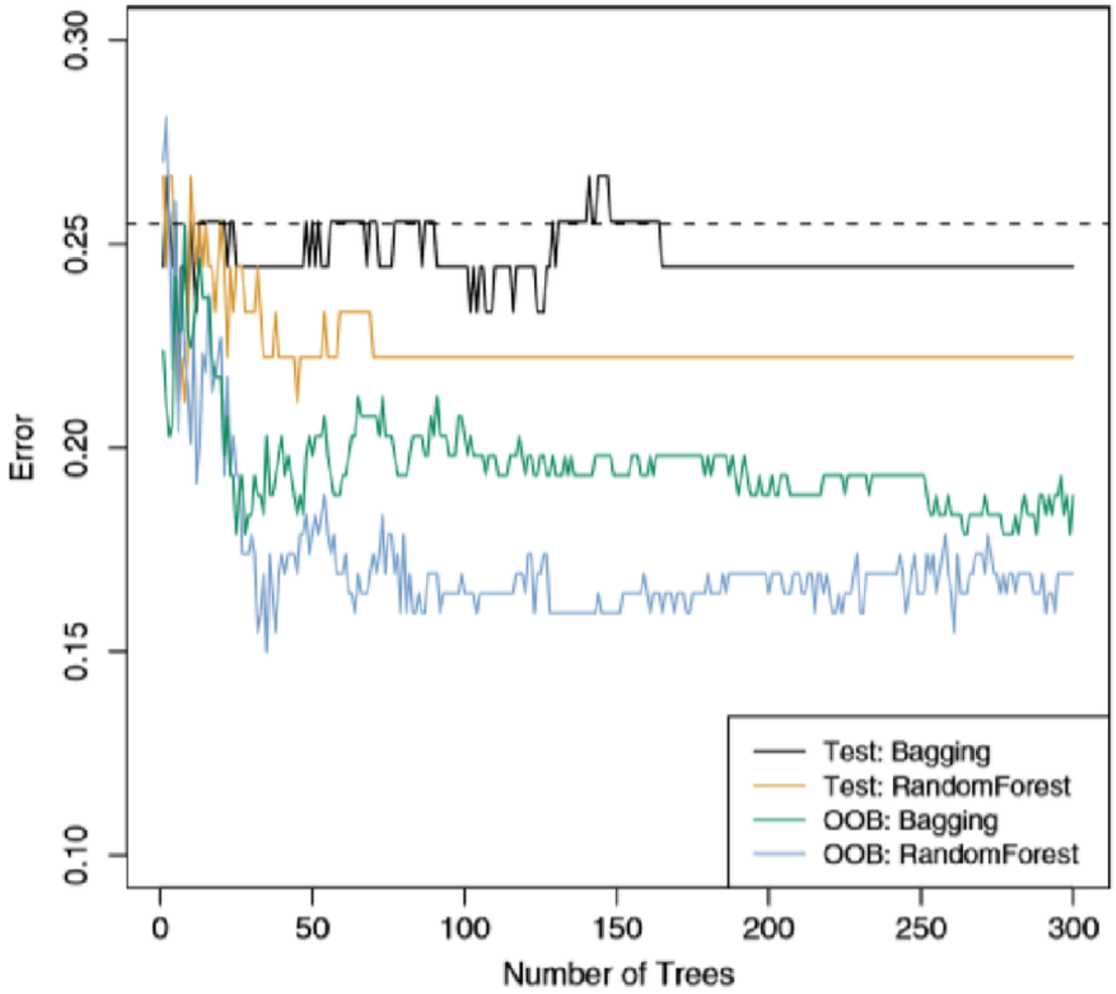
Bagging Limitations

- What if you have a strong predictor and then a bunch of moderate predictors?
- Each time, the first variable is that strong predictor!
- So, are these trees really any different? In other words, is variance really reduced?
- What if at each split of each tree we only consider a subset m of predictors p?
- In other words: What if we randomly eliminate the strong predictor when making some of the trees?

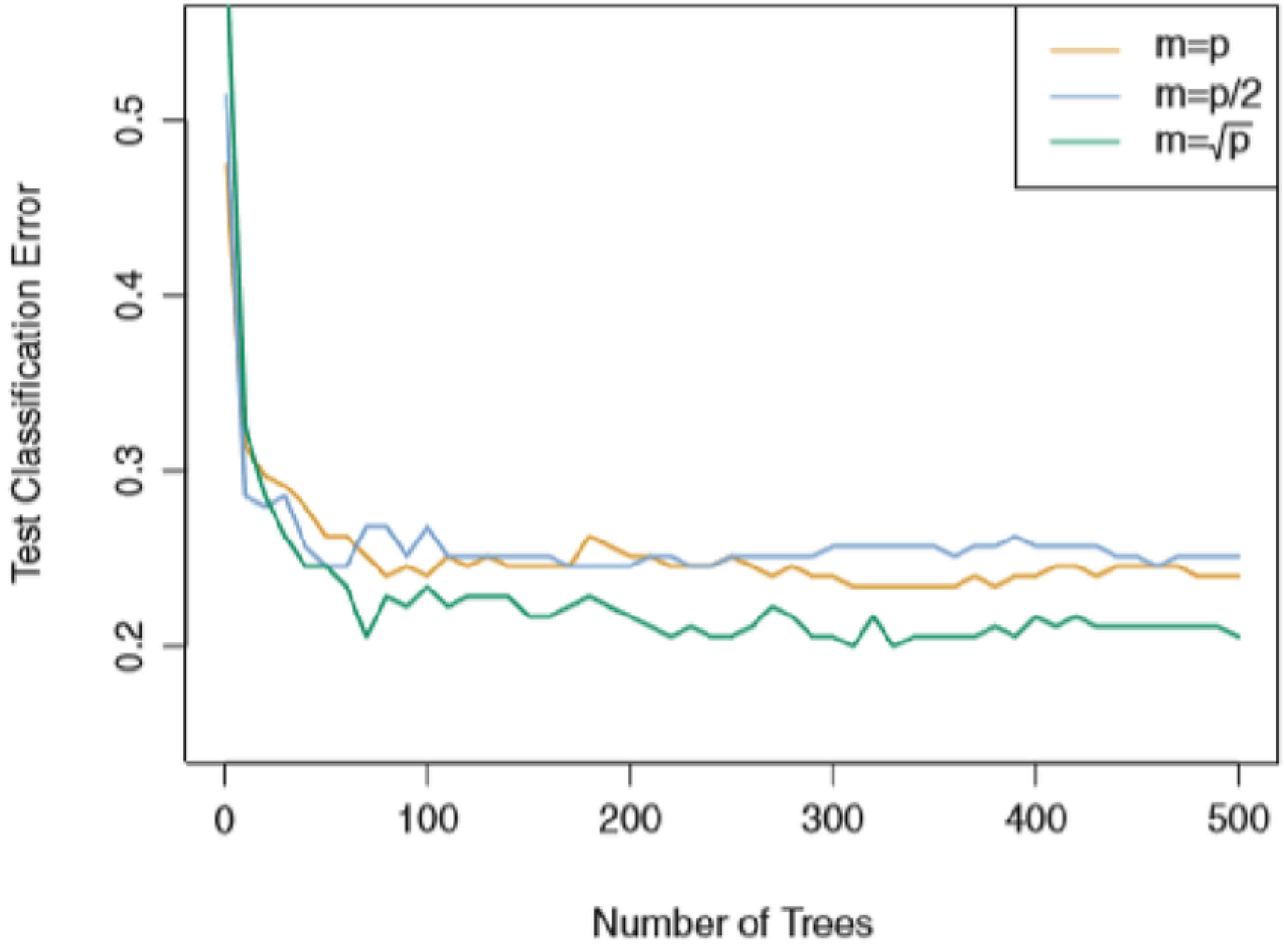
Random Forests

- Set $m \approx \sqrt{p}$
- Each time you want to build a tree: $\frac{p - m}{p}$ predictors aren't considered
- This gives other moderate predictors a chance to be important!
- The average tree becomes less variable and thus, more reliable!

Random Forest: Heart Disease



Random Forest: Adjusting m



What we have learnt so far!

- Decision Trees
  - Hierarchical structure to perform modeling
  - Tree structure determined by splitting criterion
  - Pruning: prevents overfitting by limiting depth of the tree
  - Main advantage: interpretable!
- Random Forests
  - Ensemble of trees
  - Through bagging and randomization results in reduced variance
  - Great performance in practice
  - Reduced interpretability

What we have learnt so far!

- Decision Trees
  - Hierarchical structure to perform modeling
  - Tree structure determined by splitting criterion
  - Pruning: prevents overfitting by limiting depth of the tree
  - Main advantage: interpretable!
- Random Forests
  - Ensemble of trees
  - Through bagging and randomization results in reduced variance
  - Great performance in practice
  - Reduced interpretability

CAN WE DO EVEN BETTER?

Boosting

- We can potentially improve decision tree performance even further
- Develop a method here that actually works on any classifier
- Decision Tree: Build a tree based upon variable importance in training data
- Bagging: Build each tree randomly – reduces variance, improves overfitting issues
- Random Forest: build each tree randomly, with random variations in predictors, improves performance/accuracy

- So: What if we build trees in a sequential, ordered fashion?

Generalized Boosting

- Can we build a high-capacity model "one unit at a time"?

$$model\,(x, w) = w_0 + f_1(x) * w_1 + f_2(x) * w_2 + \ldots + f_M(x) * w_M$$

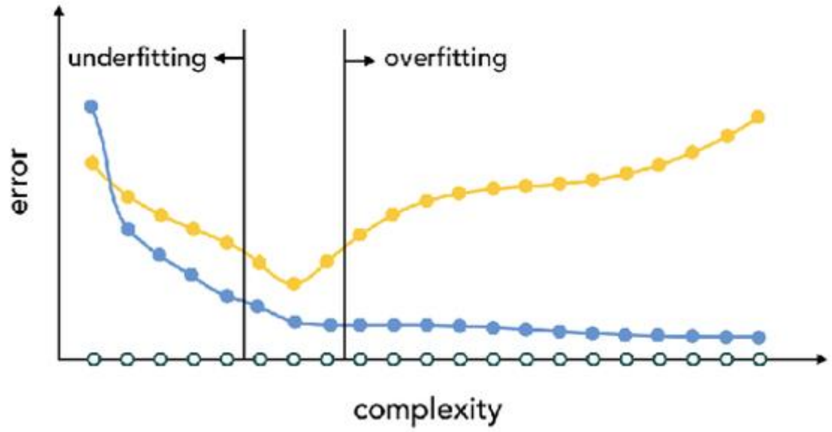
Generalized Boosting

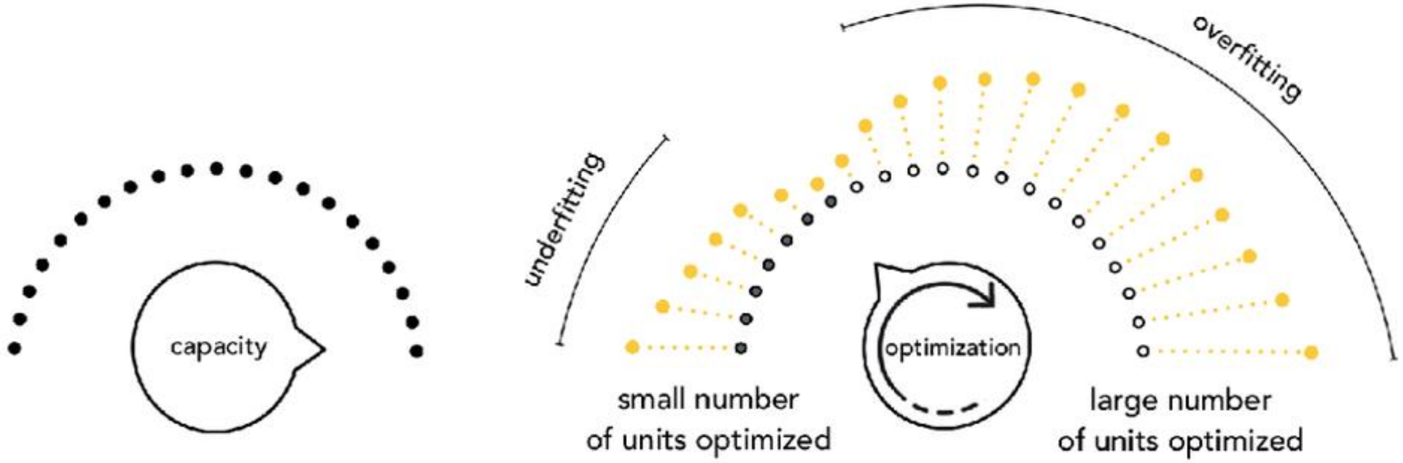
- Can we build a high-capacity model “one unit at a time”?

$$model\ (x, w) = w_0 + f_1(x) * w_1 + f_2(x) * w_2 + \ldots + f_M(x) * w_M$$

- The basic principle of boosting is to progressively build a high capacity model one unit at a time

Boosting Capacity





Sequential Building

- Set of M nonlinear features or units from a single family of universal approximators

$$F = \{f_1(x), f_2(x), \dots, f_M(x)\}$$

- Add units sequentially (one at a time) to build a set of M models that increase in complexity with respect to the trained data, ending with a generic nonlinear model composed of M units

Boosted Model

$$model\,(x, w) = w_0 + f_{s_1}(x) * w_1 + f_{s_2}(x) * w_2 + \ldots + f_{s_M}(x) * w_M$$

- Re-indexed the individual units to f_{s_m} to denote the unit from the entire collection F added in the mth round
- The linear combination of weights w are collectively represented sometimes as θ

Boosting Procedure

- Process of boosting is performed for a total of M rounds
- At each round, we determine which unit, when added to the running model, best lowers its training error
- We then measure the corresponding validation error
- For the sake of simplicity – let's stick with regression for now
- Classification remains exactly the same!

Initial Round (Round 0)

- Start with model:

$$model_0(x, \theta) = w_0$$

- Whose weight set $\theta_0 = \{w_0\}$, which contains a single bias weight which minimizes least squares (notation from Watt, Borhani, and Kastaggelos – P is people):

$$\frac{1}{p}\sum_{p=1}^{P}\left(model_0\left(x_p, \theta_0\right) - y_p\right)^2 = \frac{1}{p}\sum_{p=1}^{P}\left(w_0 - y_p\right)^2$$

- This optimal w_0 remains fixed forever forward

Round 1 of Boosting

- Having tuned the only parameter, we now boost its complexity by adding weighted unit $f_{s_1}(x)\,w_1$:

$$model_1(x, \theta_1) = model_0(x, \theta_0) + f_{s_1}(x)\,w_1$$

- To determine which unit in our set F best lowers the training error, we pick the $f_s \in F$ that minimizes the cost:

$$\frac{1}{p}\sum_{p=1}^{P}(model_0(x_p, \theta_0) + f_{s_1}(x_p)\,w_1 - y_p)^2 =$$

$$\frac{1}{p}\sum_{p=1}^{P}(w_0 + f_{s_1}(x_p)\,w_1 - y_p)^2$$

Round m > 1 of Boosting

$$model_{m-1}(x, \theta_{m-1}) = w_0 + f_{s_1}(x) * w_1 + f_{s_2}(x) * w_2 + \ldots + f_{s_{m-1}}(x) * w_{m-1}$$

- We then seek out the best next unit to add

$$model_m(x, \theta_m) = model_{m-1}(x, \theta_{m-1}) + f_{s_m}(x)\, w_m$$

- By minimizing

$$\frac{1}{p} \sum_{p=1}^{P} \left(model_{m-1}(x_p, \theta_{m-1}) + f_{s_m}(x_p)\, w_m - y_p \right)^2 =$$

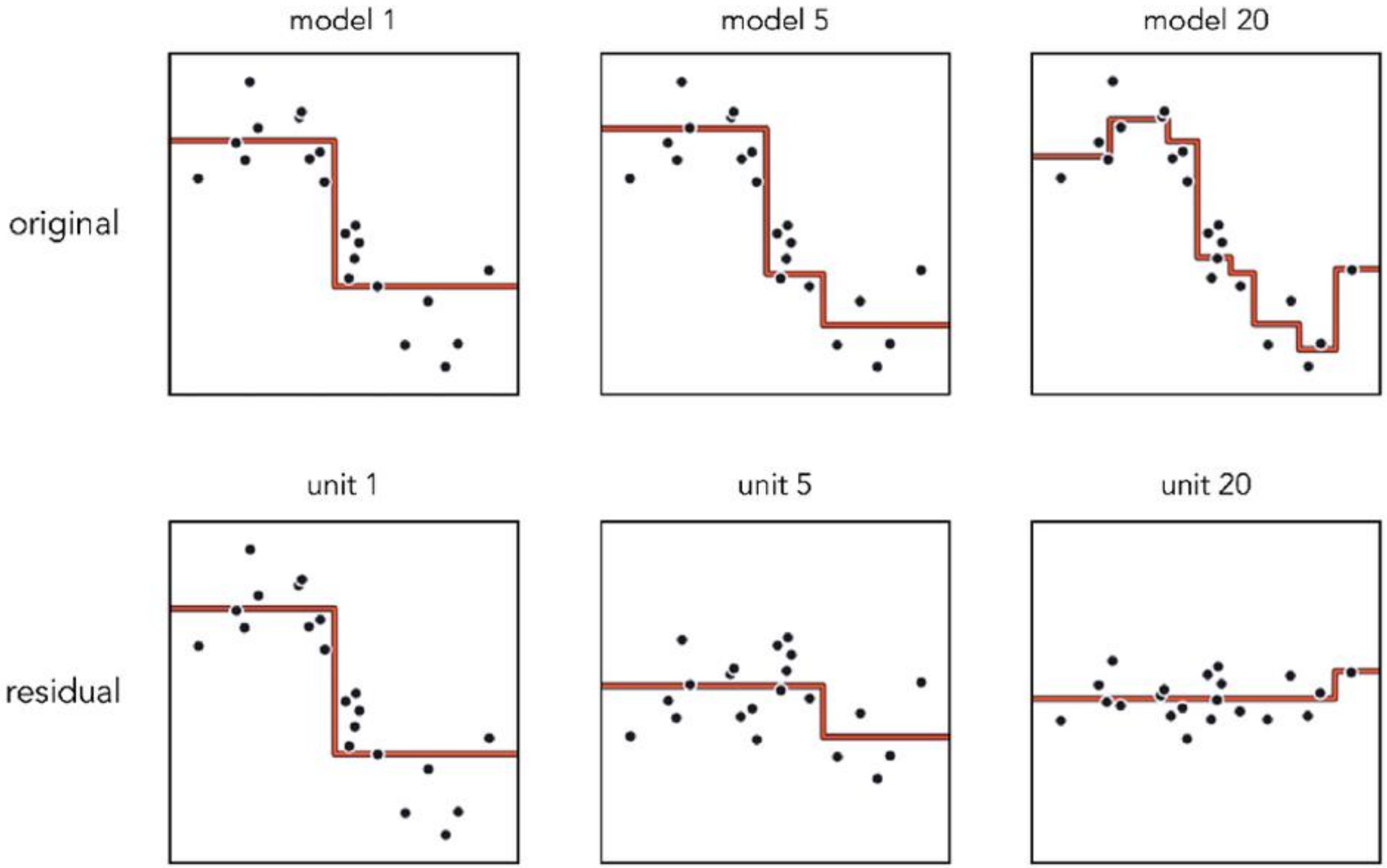
$$\frac{1}{p} \sum_{p=1}^{P} \left(w_0 + f_{s_1}(x_p)\, w_1 + \ldots + f_{s_m}(x_p)\, w_m - y_p \right)^2$$

Boosting Loss

$$\frac{1}{p}\sum_{p=1}^{P}\left(w_0 + f_{s_1}(x_p)\, w_1 + \ \ldots + f_{s_m}(x_p)\, w_m - y_p\right)^2$$

- If we use a fixed-shape approximator, say a decision stump or decision tree, this entails solving M (or M – m + 1, if we decide to check only those units not used in previous rounds) optimization problems
- If we use neural networks, each unit takes the same form, we only need to solve one such optimization problem (for future reference)
- Once boosting is complete we select from our set of models the one that provides the lowest validation error
- Alternatively, can halt if the validation error increases (overfits) – *early stopping*
- Be careful, validation error can oscillate

Visualization using trees



Boosting for Regression Trees: Algorithm

1. Set $\hat{f}(x) = 0$ and error $r_i = y_i$

Boosting for Regression Trees: Algorithm

1. Set $\hat{f}(x) = 0$ and error $r_i = y_i$
2. For b = 1, 2, … , B repeat:
   a. Fit a tree $\widehat{f^b}$ 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

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

Boosting for Regression Trees: Algorithm

1. Set $\hat{f}(x) = 0$ and error $r_i = y_i$
2. For b = 1, 2, … , B repeat:
   a. Fit a tree $\widehat{f^b}$ 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

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

   c. Update the residuals

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

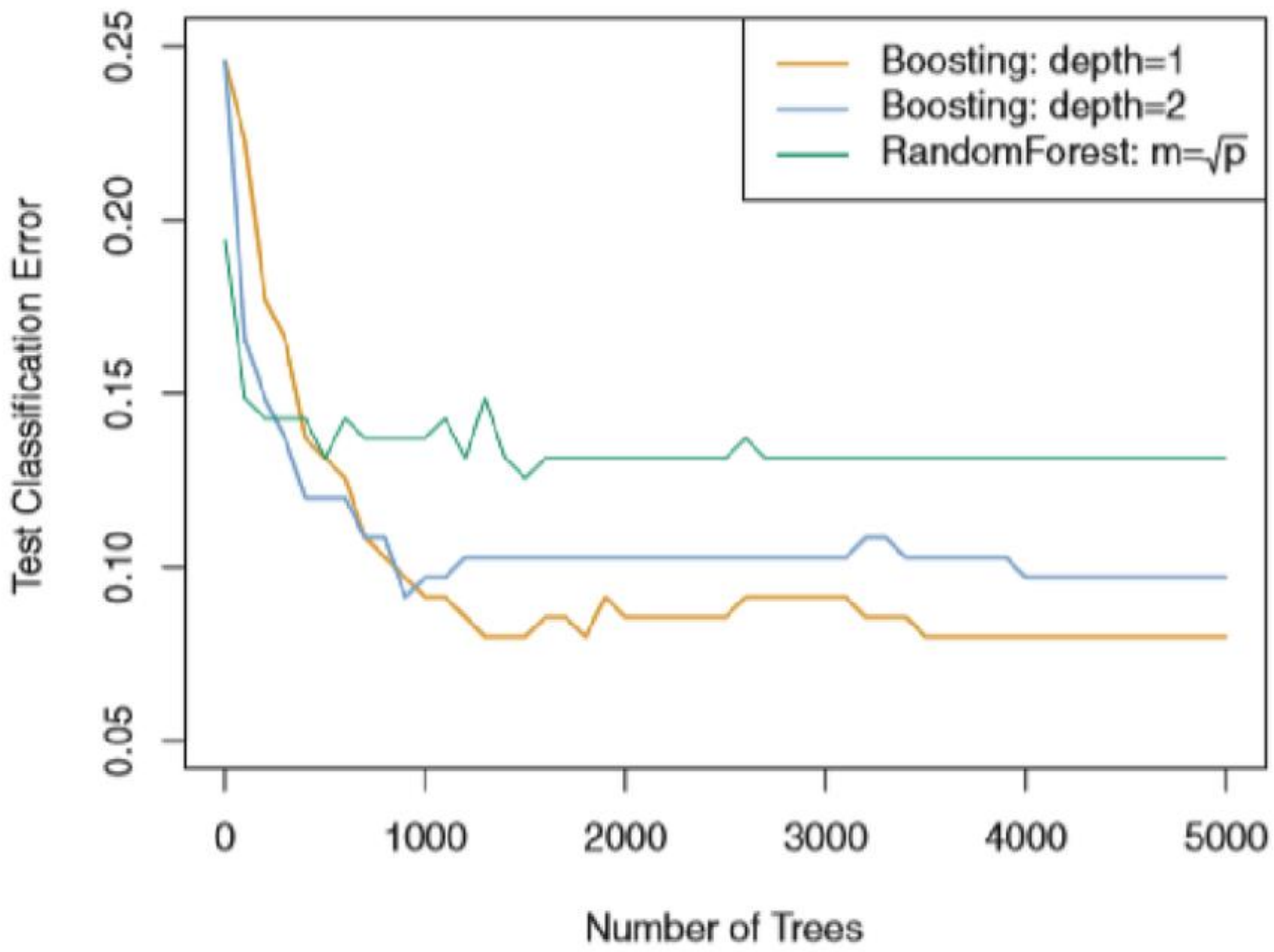
3. Output the boosted model

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

Boosted Decision Trees

- Learn slowly from shallow trees
- Given a current model – calculate residuals
- Build next tree to improve on the remaining residuals
- Slowly improve where the model does not currently perform well
- Boosted classification becomes a bit trickier in how it updates (next time)

Boosting vs. Random Forest



What we have learnt so far!

- Decision Trees
  - Hierarchical structure to perform modeling
  - Tree structure determined by splitting criterion
  - Pruning: prevents overfitting by limiting depth of the tree
  - Main advantage: interpretable!
- Random Forests
  - Ensemble of trees
  - Through bagging and randomization results in reduced variance
  - Great performance in practice
  - Reduced interpretability
- Boosted Trees
  - Ensemble of trees (boosts a bunch of weak classifiers into a strong classifier)
  - Through sequential building, improves model performance
  - If B is too large, this model Does overfit
  - λ is small, but greater than 0
  - Depth d of trees is often small (d = 1, decision stumps, are very interpretable)

Next Time

- More Boosting
- Gradient Descent Boosting for Classification
- Loss Functions
- Python Coding