Introduction to Machine Learning – Decision Trees Pt. 2

Dr. Ab Mosca (they/them)

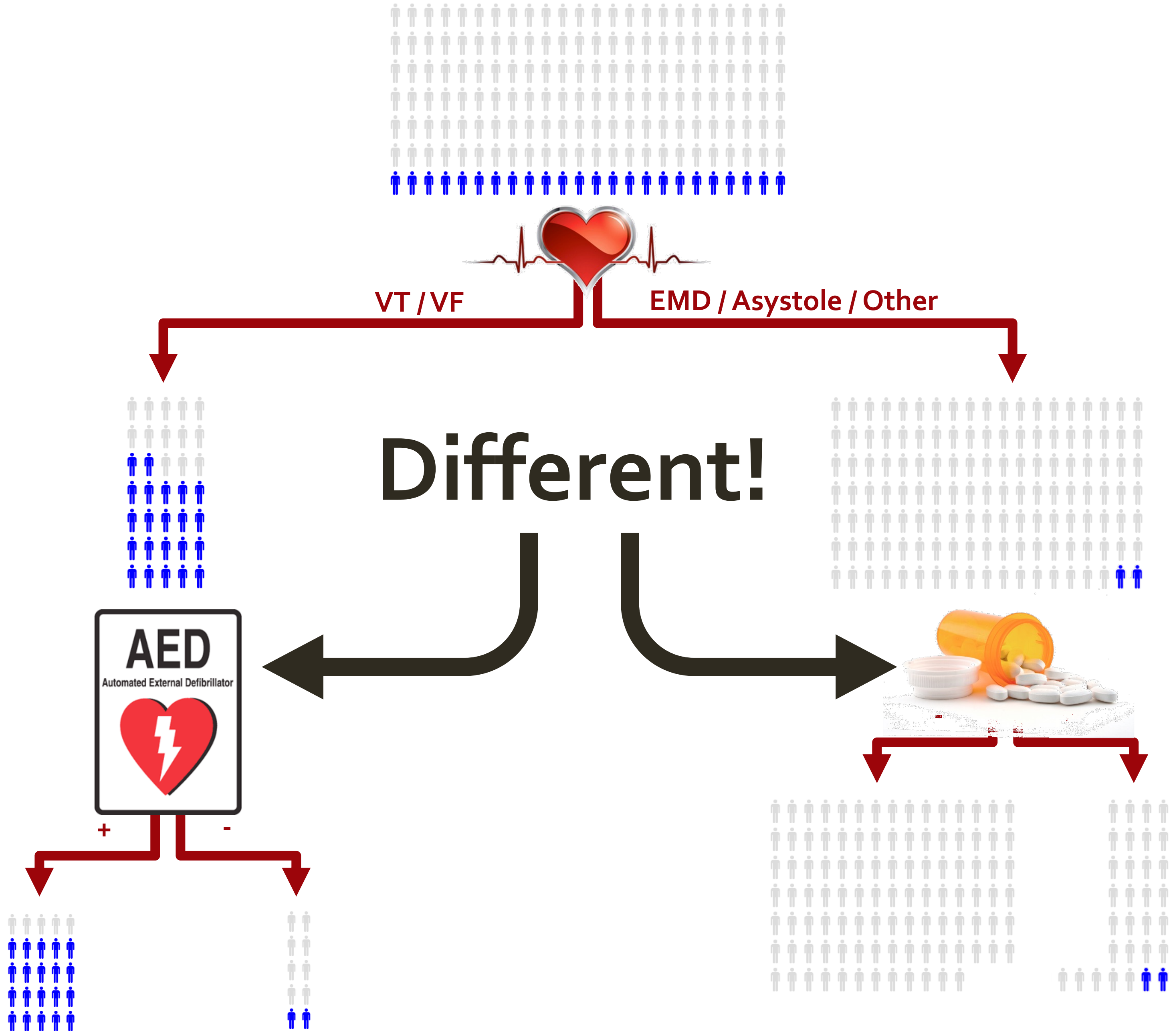
Slides based off slides courtesy of Jordan Crouser (https://jcrouser.github.io/)

Plan for Today

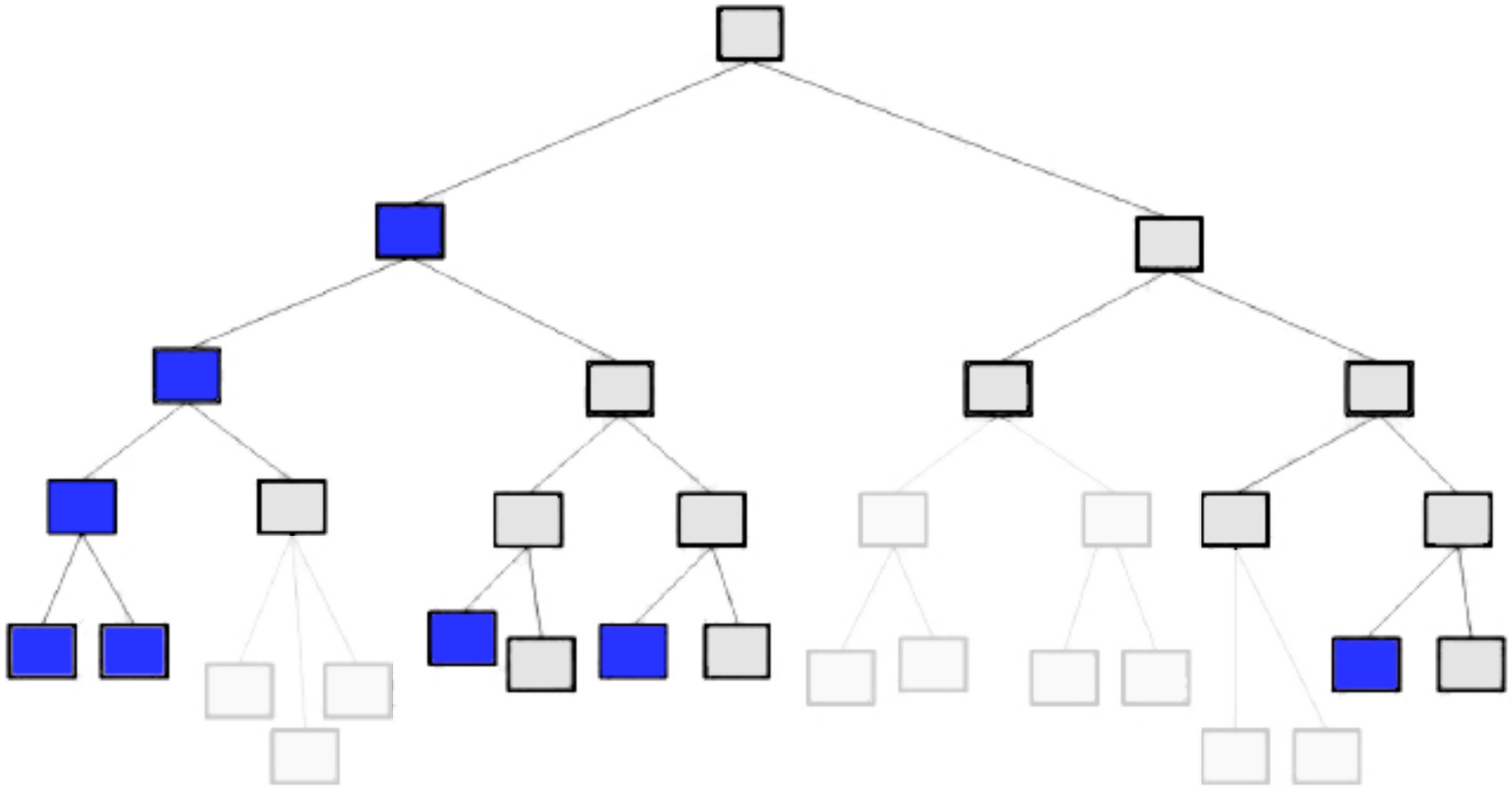
Avoiding overfitting of decision trees

- Bootstrap aggregation (“bagging”)
- Random forests
- Boosting

Flashback


VT / VF
EMD / Asystole / Other
Different!
AED
Automated External Defibrillator
+
-

Flashback



Problem: high variance

Discussion

What can we do to combat high variance?

Bagging

Big idea: use regular old bootstrapping to generate a bunch of sample training sets, build trees for each one, and **average** their resulting predictions

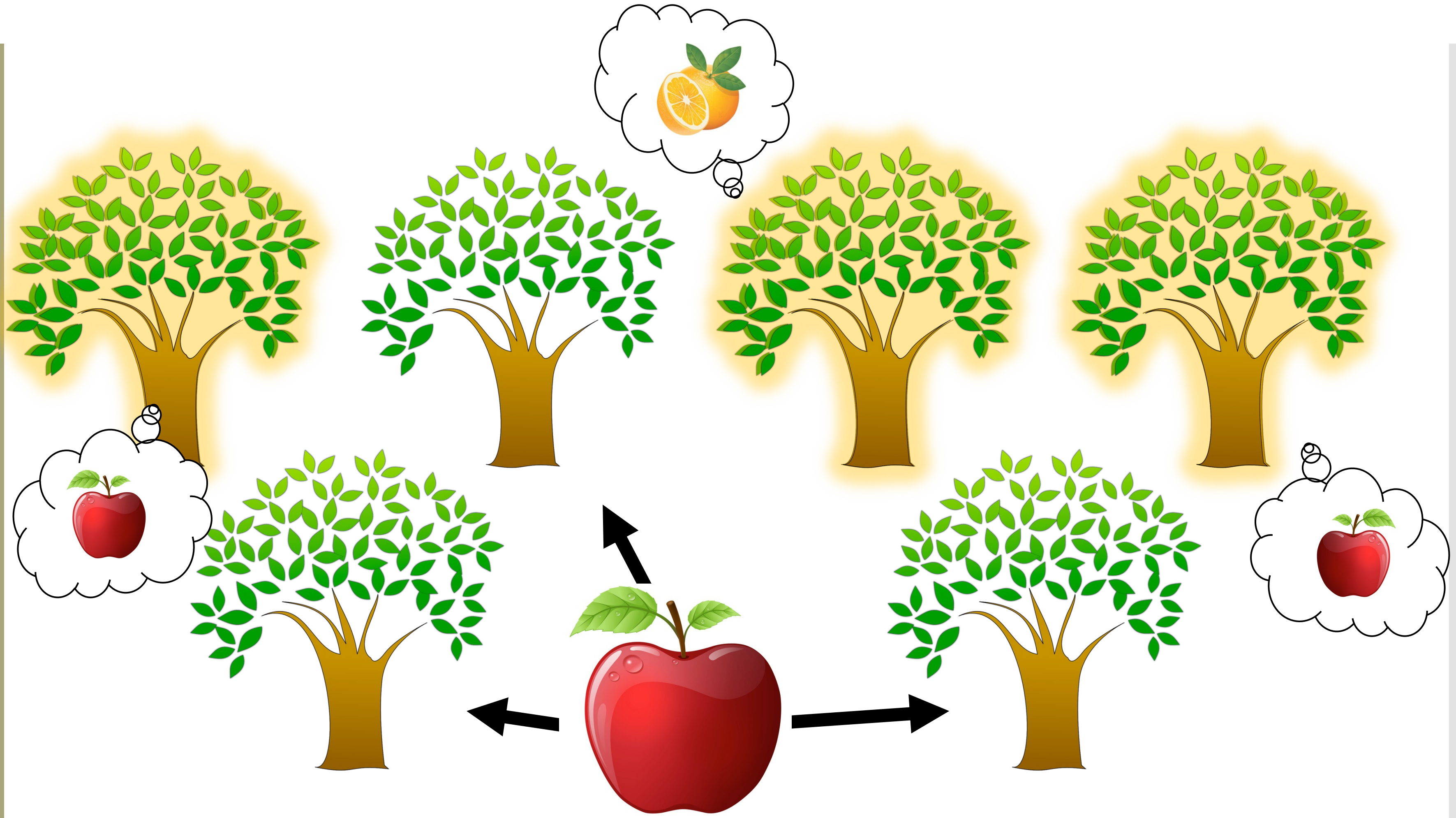
Image courtesy Rachopin77 on DeviantArt

Estimating test error

- **Fun fact**: there is a straightforward way to estimate the test error of a bagged model, without needing to use a test set or cross-validate!

- **Key**: trees are repeatedly fit to bootstrapped subsets
  - Each bagged tree only trains on ~2/3 of the observations
  - The remaining ~1/3 are called the **out-of-bag (OOB) observations**

OOB error



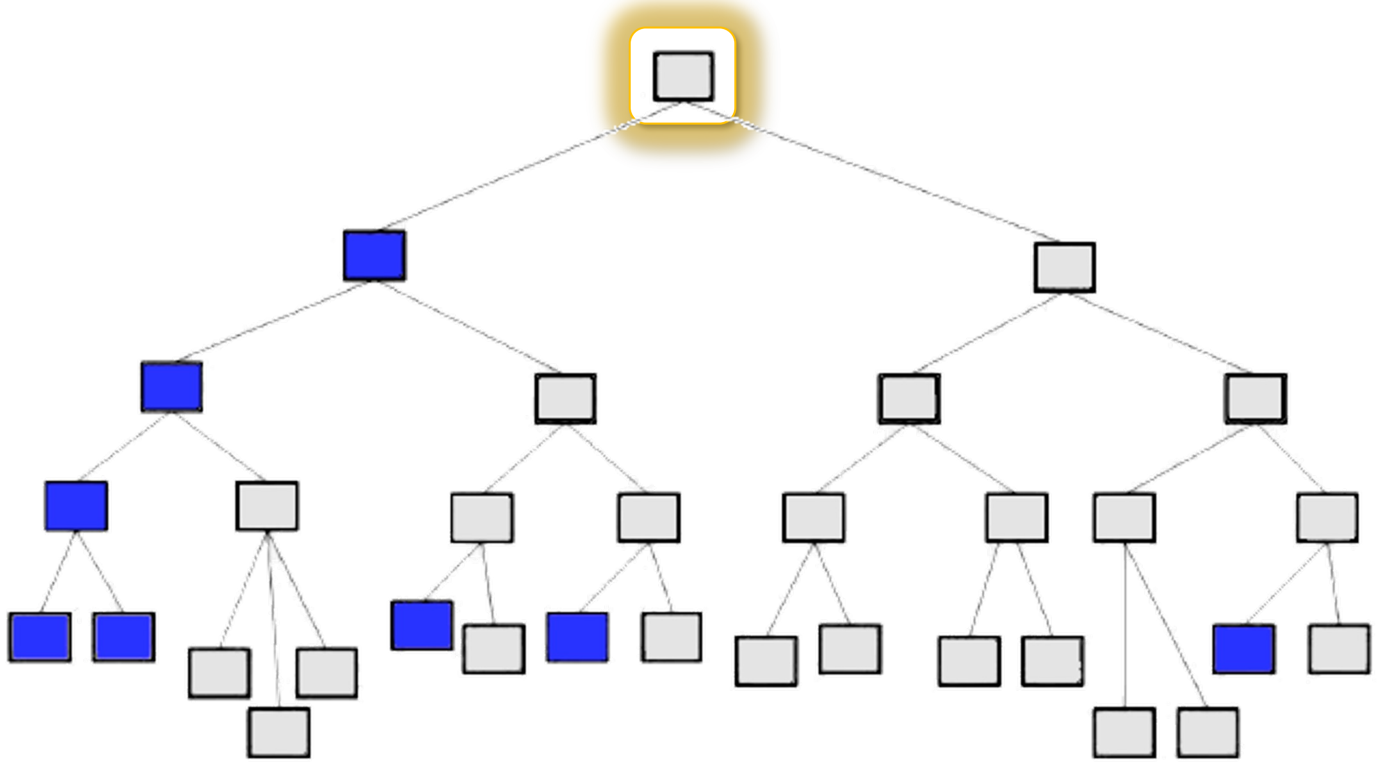
Trained on this observation?

Repeat for **every observation**, average to get MSE or classification error

*with enough trees, this is essentially equivalent to LOOCV error

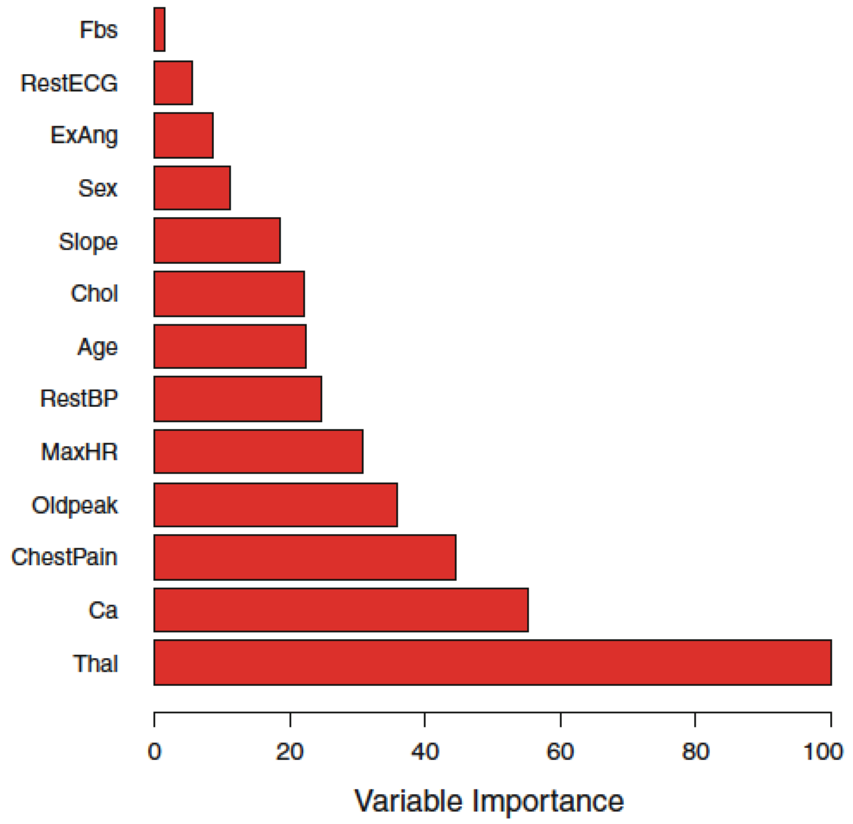
Measuring predictor importance

- With just one tree, it was easy to pick out the most important predictor (which one was it?)



Measuring predictor importance

- With lots of trees, we can't just "read from the top"
- Instead, we can look at average reduction in RSS or Gini due to splits on a given predictor



Just one problem…

- One issue with bagging is that it sometimes gives us trees that are pretty **highly correlated** (why?)

If we have one **very strong predictor** in the data set, most or all of the trees will use this predictor in the **top split**

- Averaging highly correlated values **doesn't reduce variance** as much as averaging uncorrelated quantities

Random forests

- **Strange idea**: what if each time we go to make a split, we randomly limit the choice to some *subset* of predictors?

Random forests

- **Strange idea**: what if each time we go to make a split, we randomly limit the choice to some *subset* of predictors?
    - Only consider ***m*** **out of** ***p*** predictors each time we decide on a split

Random forests

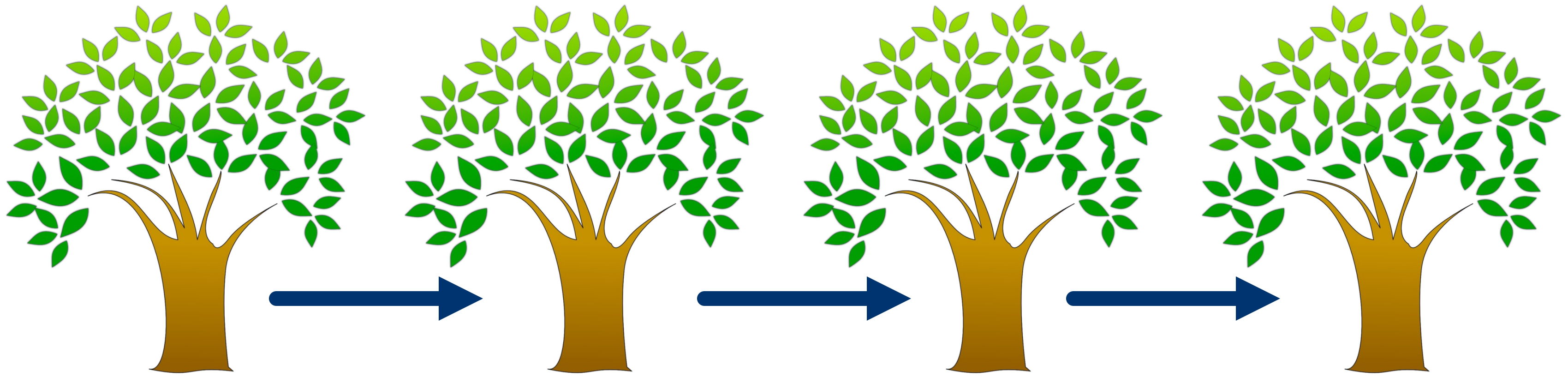
- **Strange idea**: what if each time we go to make a split, we randomly limit the choice to some *subset* of predictors?
  - Only consider ***m*** **out of** ***p*** predictors each time we decide on a split
  - Roughly *(p-m)/p* splits **won't even consider** that bully predictor, so other predictors will have a chance

Random forests

- **Strange idea**: what if each time we go to make a split, we randomly limit the choice to some *subset* of predictors?
  - Only consider $\boldsymbol{m}$ **out of** $\boldsymbol{p}$ predictors each time we decide on a split
  - Roughly $(p-m)/p$ splits **won't even consider** that bully predictor, so other predictors will have a chance
  - **Note:** when $m = p$, this is just bagging!

Boosting

- **Previous methods:** generate a bunch of training sets, fit a tree on each one independently, and aggregate results



- **Boosting** works in a similar way, except that each tree is grown using information from **previous trees**

Boosting

- **Big idea:** fit each new tree using the **residuals** from the previous tree as the response

- A shrinkage parameter, λ, slows the process even more, allowing different-shaped trees to try to deal w/ residuals

- By fitting small trees to the residuals (i.e. variance we haven't yet explained), we **slowly*** improve the model in areas where it makes mistakes

* in general, statistical learning approaches that learn **slowly** tend to perform **well**

Boosting: algorithm

1. Initialize $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set
2. For each $b = 1, 2, \dots, B$:
   a) Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to $\boxed{(X, r)}$

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

Takeaways

- Tree-based methods **partition the predictors** into a number of simple regions, and use the average value of each region to make predictions
- While easy to interpret, trees **typically won't outperform** other methods we've seen in terms of prediction accuracy
- Bagging, random forests, and boosting all try to fix this by **growing multiple trees** and using "consensus prediction"
- In lab, we will see that **combining a large number of trees** can result in dramatic improvements in prediction accuracy (at the expense of some *loss in interpretability*)

Activity

- Consider the data below. By hand, use a random forest (if you are in group 1) or boosting (if you are in group 2) to generate a model that predicts Dose based on Age, Sex, BP, Cholesterol, and Na_to_K
- Show all of the steps to your algorithm on the board

Age	Sex	BP	Cholesterol	Na_to_K	Dose
23	F	HIGH	HIGH	25.355	20
47	M	LOW	HIGH	13.093	15
47	M	LOW	HIGH	10.114	15
28	F	NORMAL	HIGH	7.798	20
49	F	NORMAL	HIGH	16.275	20
41	M	LOW	HIGH	11.037	15
60	M	NORMAL	HIGH	15.171	20
43	M	LOW	NORMAL	19.368	20
47	F	LOW	HIGH	11.767	15
23	M	LOW	HIGH	7.298	15
74	M	HIGH	HIGH	9.567	10
58	F	HIGH	NORMAL	14.239	10