Random Forests

Brad Jacobs (via Darren Reger)

Outline

- Morning
    - Intro to Ensembles
    - Review Regression Trees, Overfitting, Bootstrapping
    - Bagging
    - Moving from Bagging to RF
- Afternoon
    - Out-of-Bag Error
    - Feature Importance
    - Best Practices for RF

Intro to Ensembles

- Main Idea: more predictors can make a better model than any one predictor by itself
- Ensemble - combines many predictors
  - Average or weighted average
  - Same kind of learner or different
- Wisdom of the Crowd

Ensembles: Intuition

- Binary classifier with Majority Vote:
  - Suppose we have 5 completely independent classifiers, each with an accuracy of 70%:
    - $5\mathbf{C}3(0.7)^3(0.3)^2 + 5\mathbf{C}4(0.7)^4(0.3)^1 + 5\mathbf{C}5(0.7)^5(0.3)^0$
    - yields 83.7% accuracy with majority vote.
  - If we had 101 such classifiers:
    - 99.9% accuracy with majority vote.

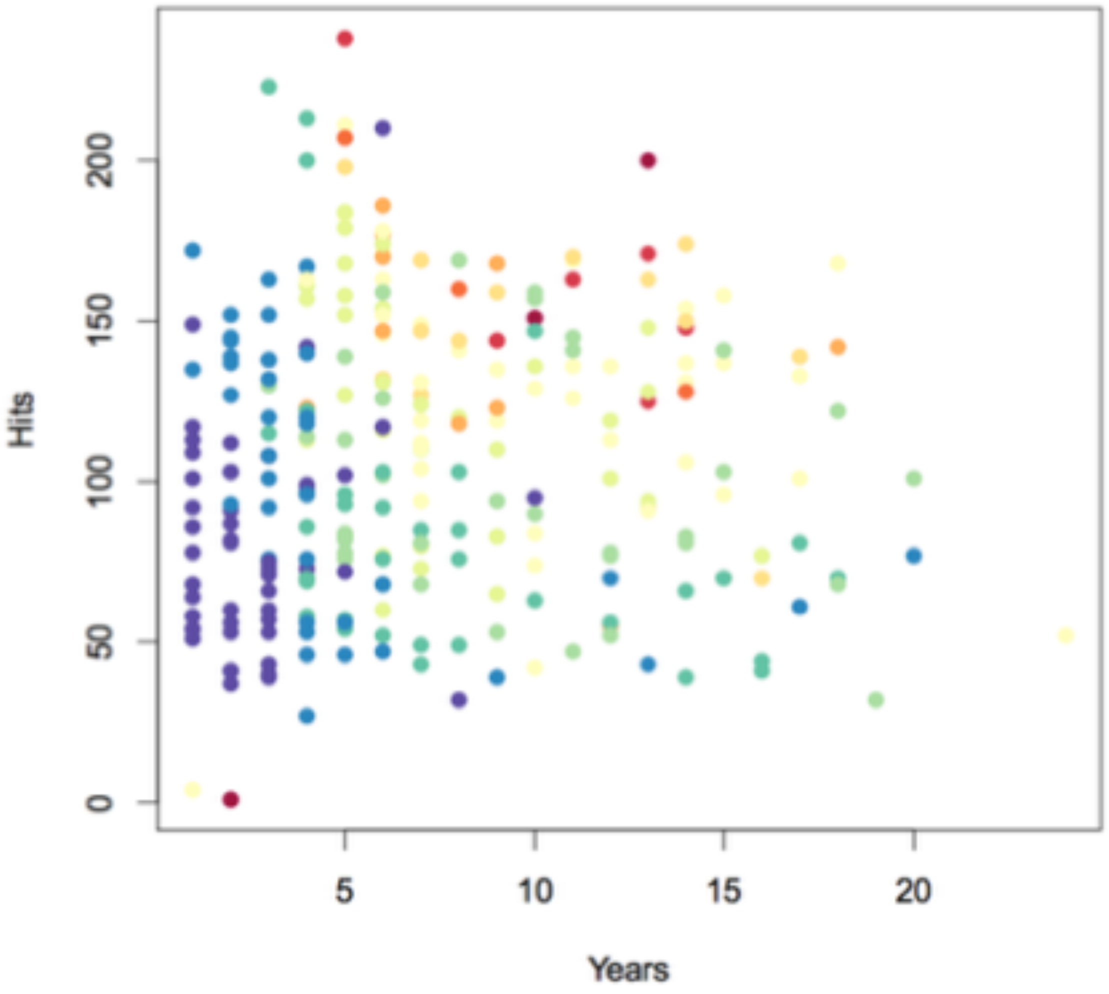
Brief Review:
Decision Tree Regression

"Some of the figures in this presentation are taken from "An Introduction to Statistical Learning, with applications in R" (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani "

Decision Trees - Regression

Salaries color-coded from low to high

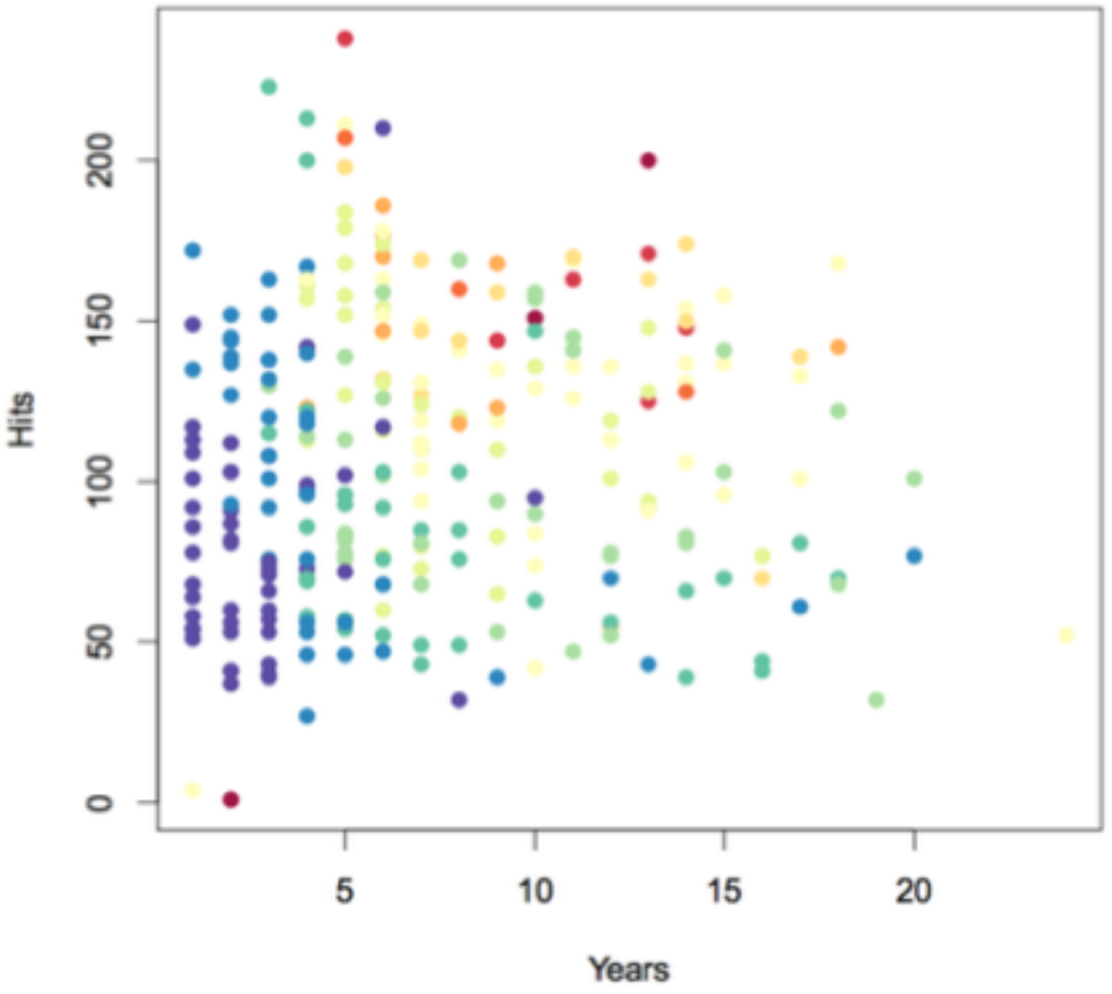
- Low salaries in blue, green
- High salaries in orange, red

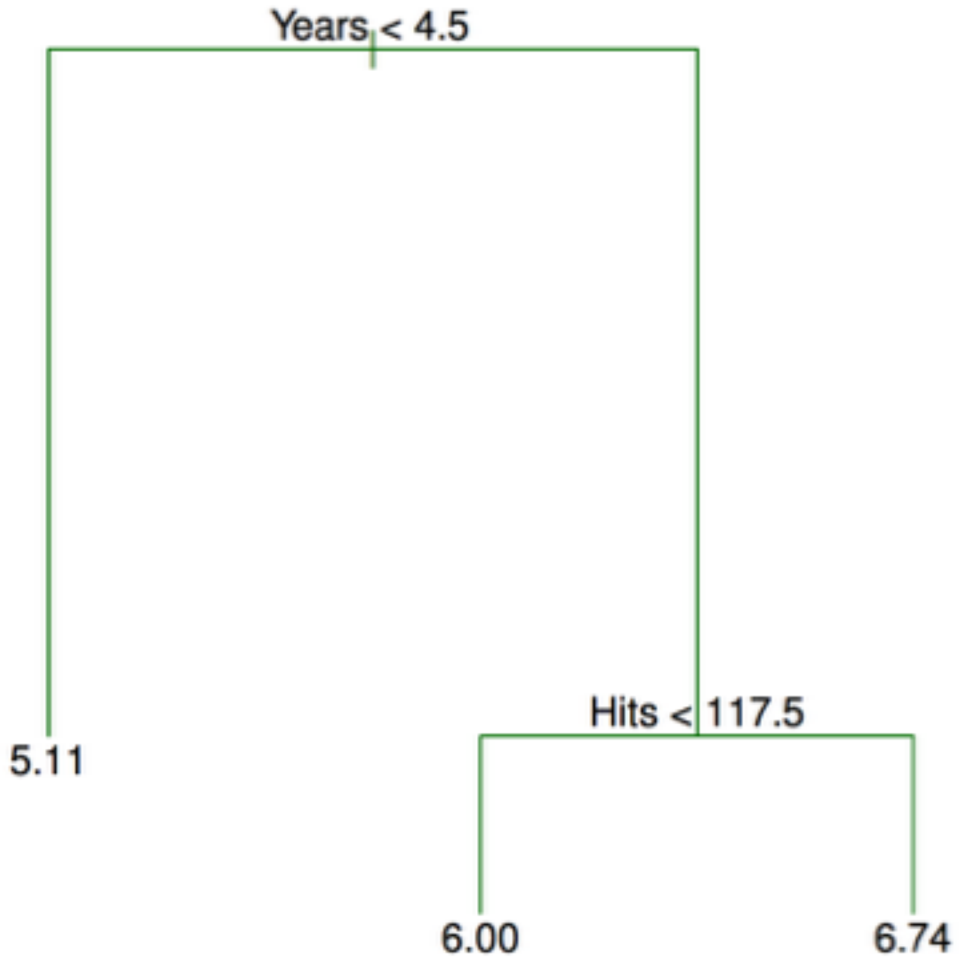


Decision Trees - Regression

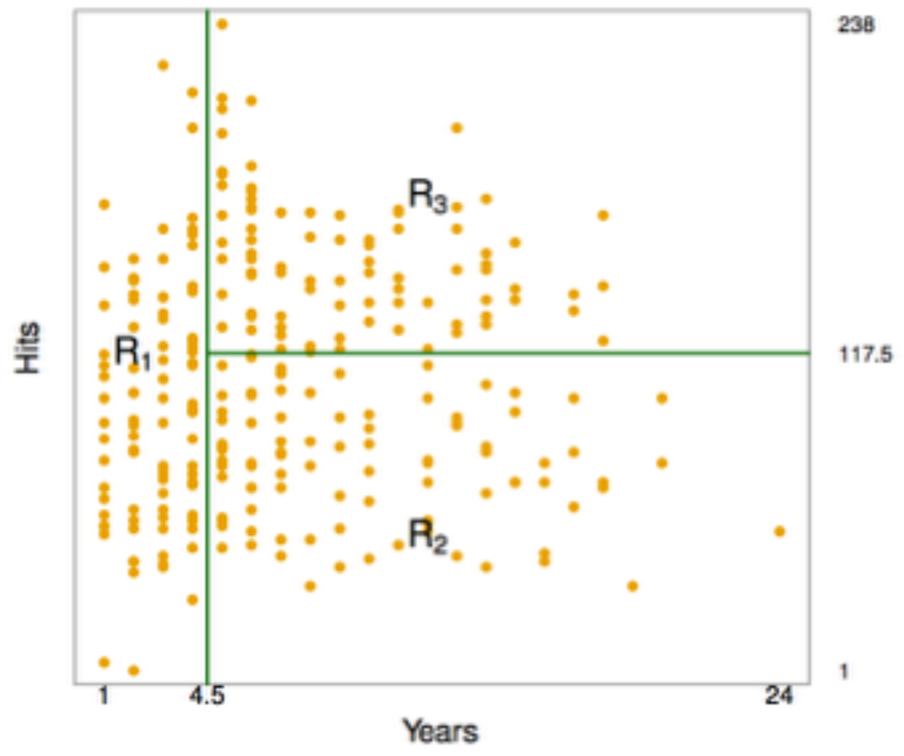
Salaries color-coded from low to high

- Low salaries in blue, green
- High salaries in orange, red





Decision Trees – Regression



Consider sequence of trees indexed by tuning parameter α.

For each α, there is a corresponding subtree, T, such that the following is minimized:

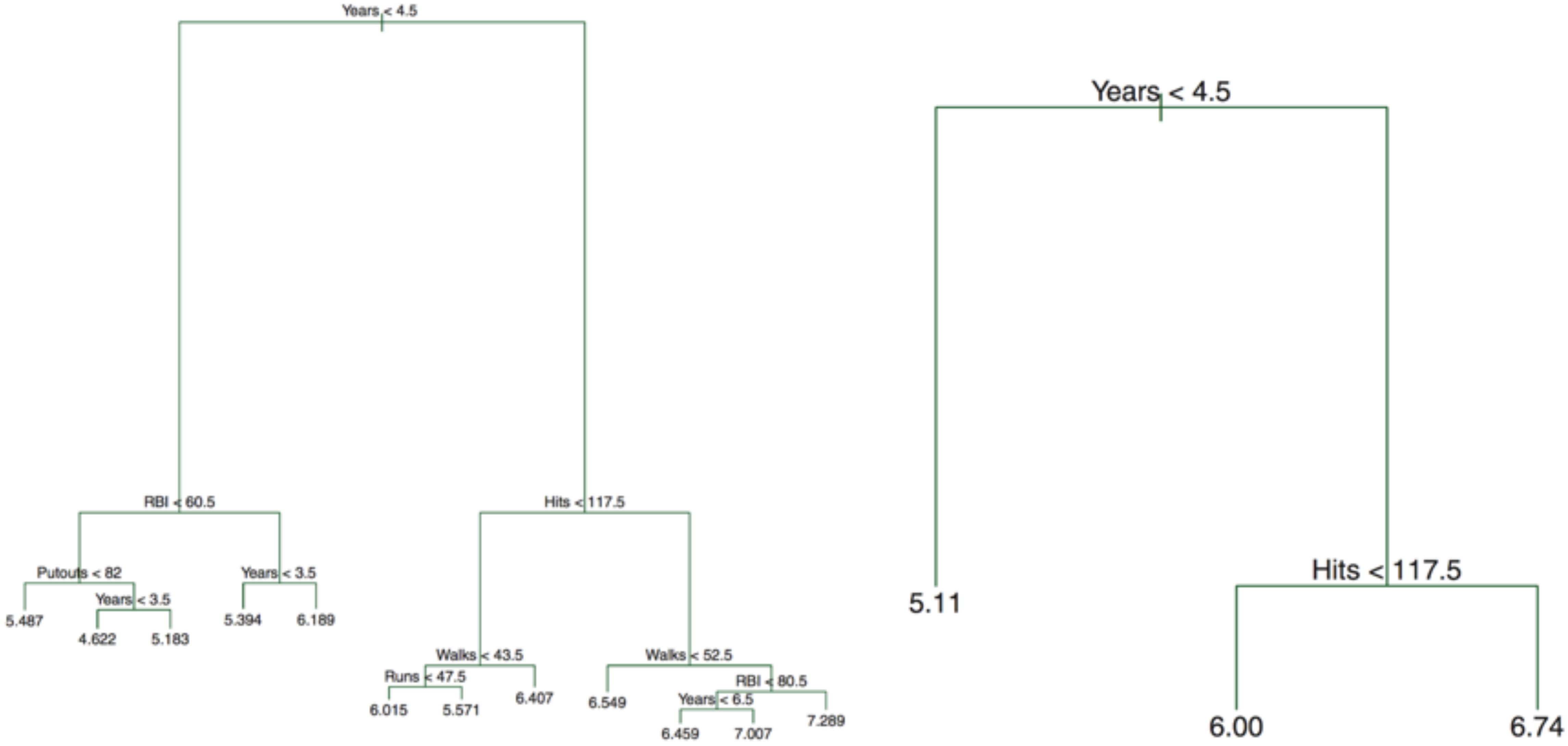
$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \overset{\downarrow}{\alpha}|T|$$

where |T| is the number of terminal nodes

Post-Pruning

Best practice for a single tree:

- Post-pruning: $\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \hat{y}_{Rm})^2 + \alpha |T|$



Review Overfitting & Bootstrap

- Overfitting:
  - “Memorizing” the training set
  - How can we see if we’ve overfit?
  - What techniques to stop it?

Bootstrap Sampling

- Method:
    - Start with your dataset of size n
    - Sample from your dataset *with replacement* to create 1 bootstrap sample of size n (many observations will be repeated)
    - Repeat *B* times
- Each bootstrap sample can then be used as a separate dataset for estimation or model fitting.

Bootstrap **AGG**regat**ING**

1. Draw a random **bootstrap** sample of size *n* (randomly choose *n* samples from the training set with replacement).

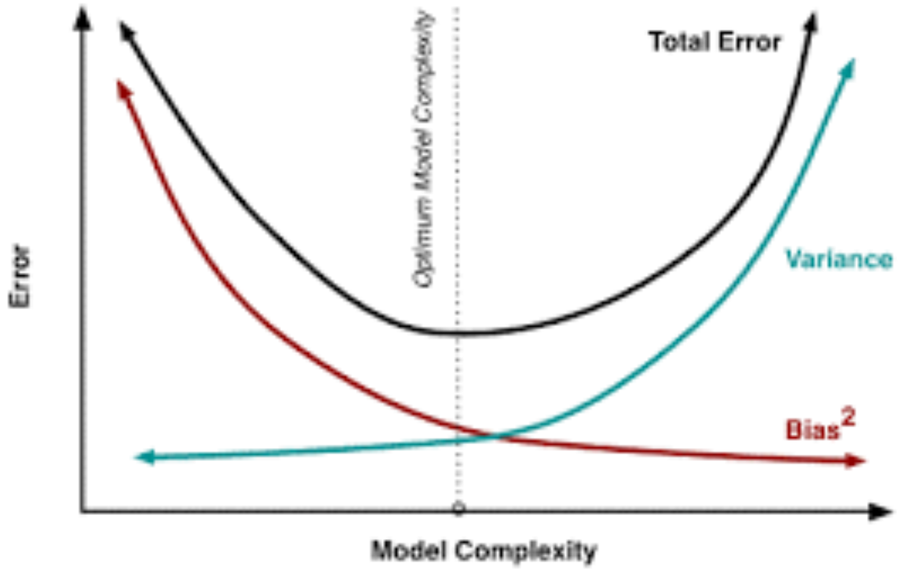
2. Grow a decision tree* from the bootstrap sample. At each node:

    1. Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.

3. Repeat the steps 1 & 2 *k* times.

4. Aggregate the prediction by each tree to assign the class label by **majority vote** (or average them for regression).

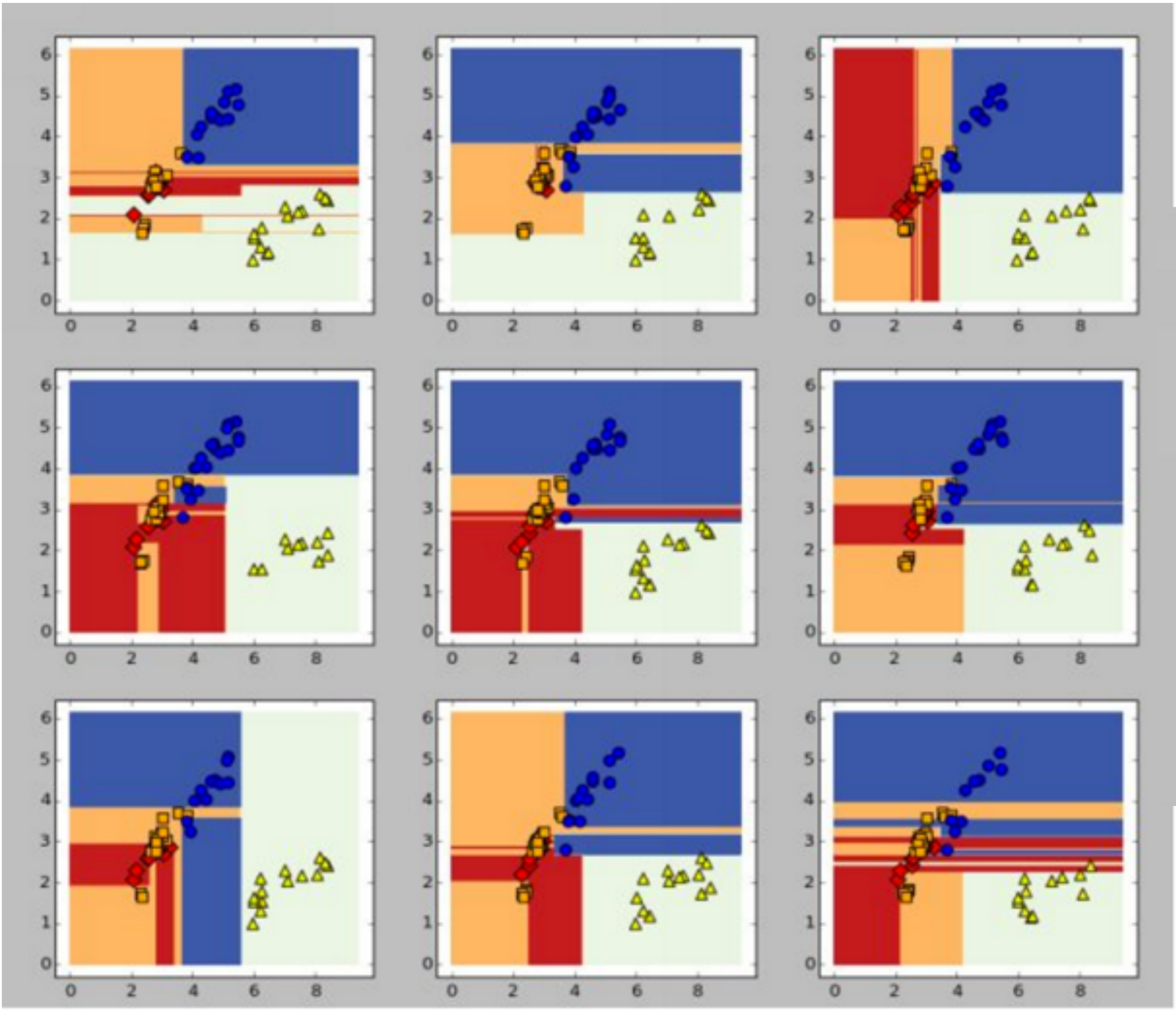
*Decision trees are the most common base model for bagging, but it can be applied to any type of predictive model.

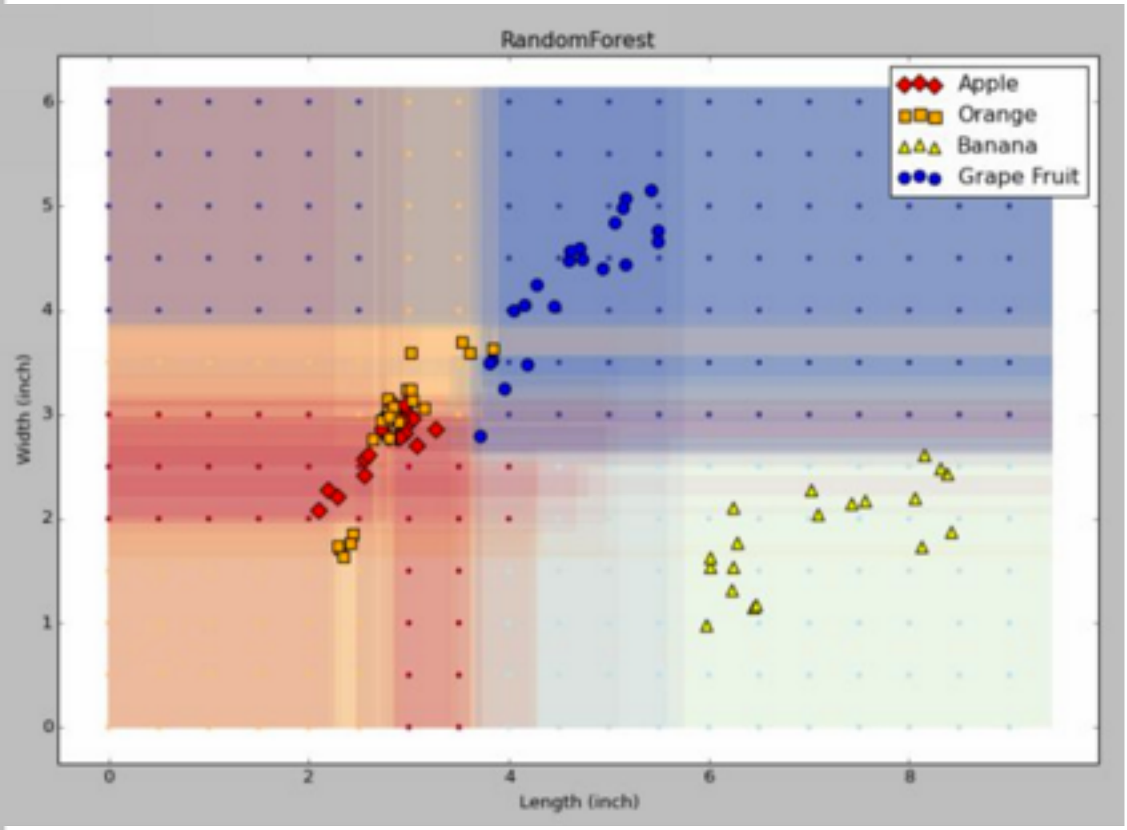


Bias Variance Tradeoff

- Bias
  - Can our model represent the true best predictor?
  - Caused by choosing a function that is too simple or choosing a regularization parameter that is too strong.
- Variance
  - Random noise due to the specifics of our training data
  - Gets better as we get more data

Comparing DT’s and RF





Random Forests vs Bagging Alone

- Random forests provide an improvement over bagged trees by way of a small tweak that further de-correlates the trees
  - Both methods utilize bootstrap samples
  - RF also split on a feature taken from a random subset of all features
- Intuition:
  - In bagged trees, a very strongly predictive feature will often/always show up at the top node, making trees look similar and making their predictions more correlated
  - In RF, the predictive feature will only be available a portion of the time, allowing moderately strong predictors a chance

Random Forest

1. Draw a random **bootstrap** sample of size *n* (randomly choose *n* samples from the training set with replacement).

2. Grow a decision tree from the bootstrap sample. At each node:

    1. **Randomly select *d* features without replacement.**

    2. Split the node using the feature that provides the best split according to the objective function, for instance, by maximizing the information gain.

3. Repeat the steps 1 & 2 *k* times.

4. Aggregate the prediction by each tree to assign the class label by **majority vote** (or average them for regression).

Feature Subsetting

- Suggested subset sizes, for *p* features:
    - Classification: $d = \sqrt{p}$
    - Regression: $d = p/3$

- In practice this is a hyper-parameter that we tune

Afternoon Lecture

Out-of-bag Error

- Recall that we are taking bootstrap samples
- $1-(1-1/n)^n$ chance of being in the bootstrap set
- Only about 63.2% of the data is in each set
- What do we do with the remaining 36.8%?
  - Use it for generalization!
- Every observation will be OOB ~1/3 of the time
- We predict on our OOB samples and see how well we do
- oob_score = True in sklearn

Random Forest Error

- Overall forest error rate depends on two things:
  - The correlation between trees in the forest
  - The strength of each individual tree in the forest
- Reducing the ***d*** (num. features considered at each split) reduces both the correlation and strength, while increasing it increases both.
  - Use OOB error to find the optimal value of ***d***.

How many trees?

- Variance decreases with more trees
  - With diminishing returns
- Run time scales linearly with more trees
- n_estimators in sklearn
- More is still better, but wait until the end to run thousands of trees

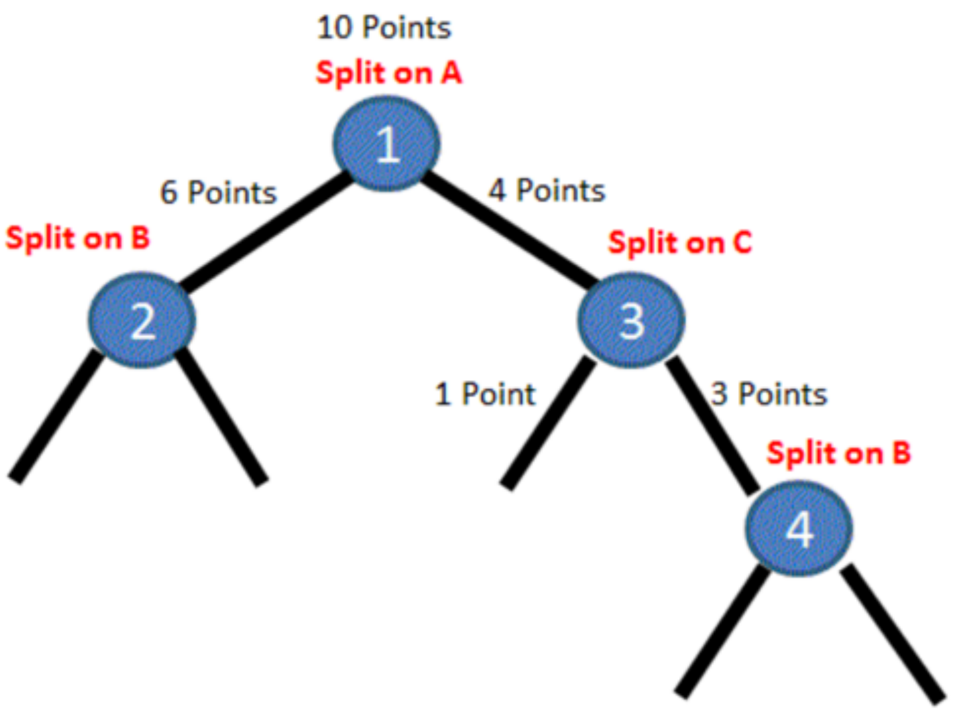
Feature Importances

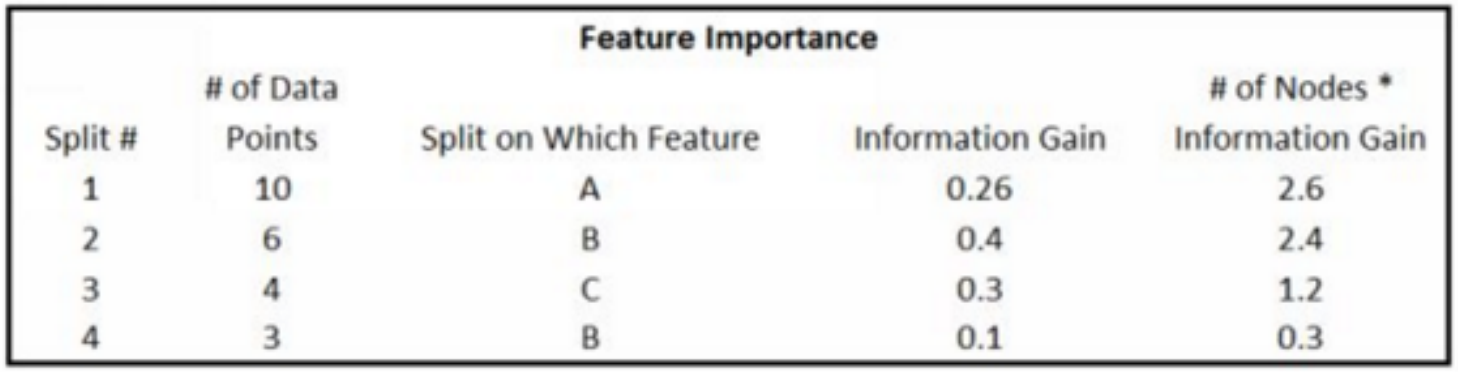
- Relative importance of the variables in your data
- When you have many features you want to know which are the most important
  - For interpretation
  - For model building

1st Feature Importance

1. Start with an array that is the same size as the number of features in your model, initialize it to 0
2. Begin to traverse the tree, with the data that was used to build the tree
3. Whenever you reach a branch in the tree, determine what feature the branch operated on (every branch splits on only one feature). Additionally, count how many data points reached this branch, as opposed to going down a different path on the tree
4. Calculate the information gain after branching as opposed to before this branch. (Criterion doesn’t matter, can be MSE, Entropy, Gini, etc…)
5. Multiply the information gain by the number of data points that reached this branch and add that product to the array at whatever feature is being split
6. Once all the information gain for all the branches has been summed, normalize it.
7. Repeat for all the trees in the forest and average the values. (best to do this at the time of tree construction)

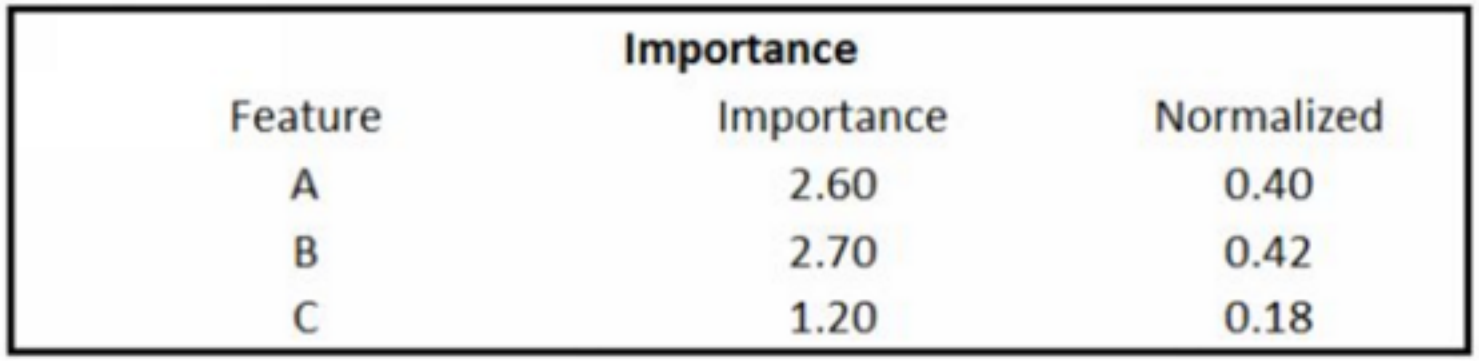
Feature Importance Example1





Feature Importance

Split #	# of Data Points	Split on Which Feature	Information Gain	# of Nodes * Information Gain
1	10	A	0.26	2.6
2	6	B	0.4	2.4
3	4	C	0.3	1.2
4	3	B	0.1	0.3



Importance

Feature	Importance	Normalized
A	2.60	0.40
B	2.70	0.42
C	1.20	0.18

2nd Feature Importance

1. Generate the OOB error for the base tree, without making any changes. This is the baseline OOB error
2. For every feature in the data used to create the tree, determine the range of values that those features can take. i.e. find the min and the max
3. For each feature, one at a time, randomly permute the values of that feature in all the data points between the max and min possible for that feature.
4. Then, after only permuting one feature, find the OOB error
5. Calculate the difference between the new OOB error and the baseline OOB error. The new OOB error will almost certainly be worse.
6. Reset the permuted feature values to their original values and repeat the process with a different feature.
7. The features which have the greatest increase in OOB error when they are permuted are the most important. If you can set a value to basically anything and not change the OOB error, it’s not important.

Feature Importance Overall

- Feature importances are almost always put forth as normalized values. What is important is that we can compare features to other features.
- The original authors of RF state that you should be interested in rank only and not magnitude
- Typically, the more features you have in a random forest, the less important any individual feature will be.
- Highly correlated features tend to split importance

Things We Can Tune

- max_features
  - classification: start with square root of p
  - regression: start with p/3 or full features
- min_sample_leaf
  - start with None and try others
- n_jobs
  - -1 will make it run on maximum # of processors

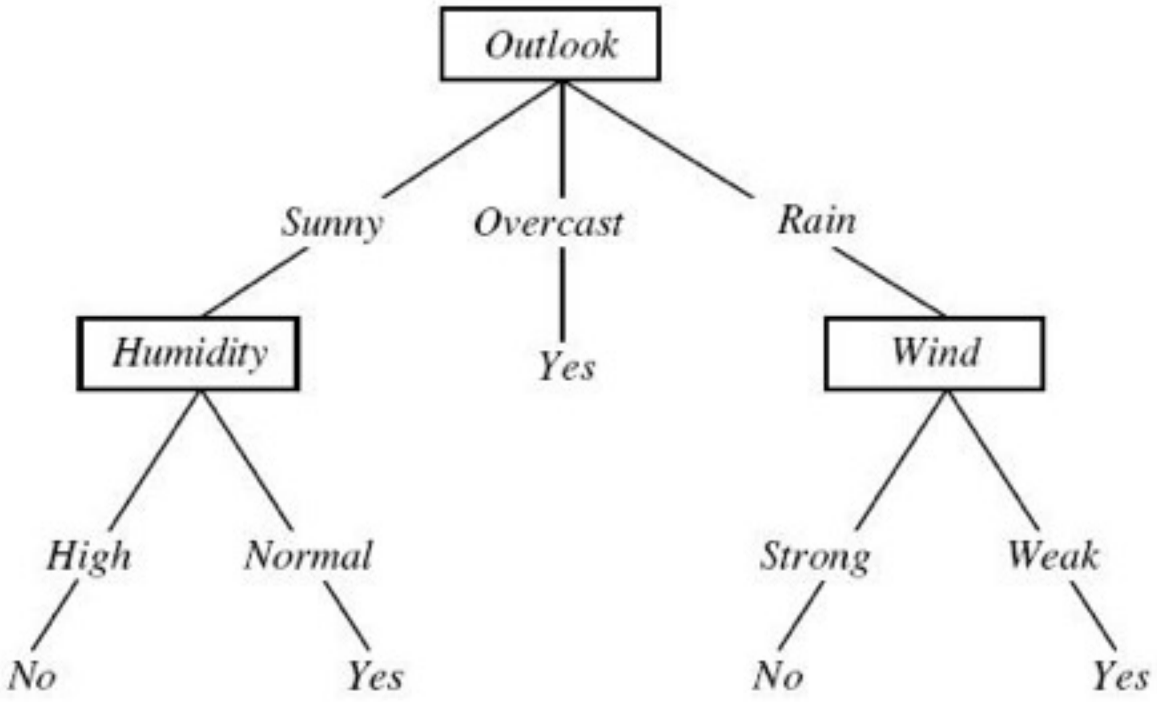
How to Handle Categorical Data

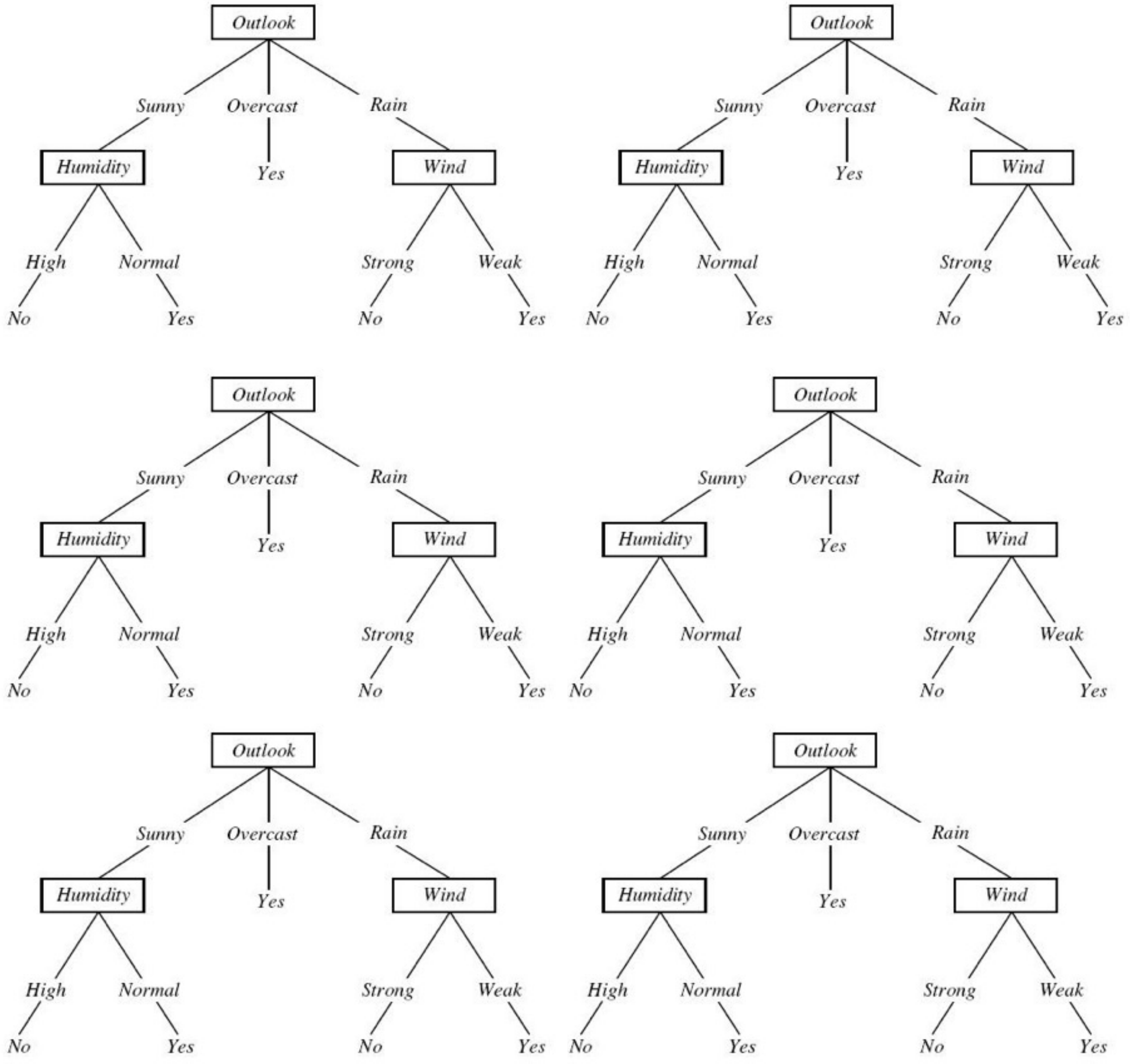
- String values need to be converted into numeric
- If possible, convert to a continuous variable
  - e.g. S, M, L to size in actual weight or height
- Sklearn doesn't support splitting on multiple features (uses the <= for all variables)
- DO NOT DROP ONE OF THE CATEGORIES!!!

Random Forest Pros

- For an out of the box model, it has very good accuracy
- Trees can be trained in parallel to make computations faster
- OOB estimates allow for an estimate of generalization error without needing CV
- Can handle thousands of features and be used for feature reduction

When are DT's Better?





Reminders

- Cannot extrapolate well for regression trees
- Just because we have interactions, doesn’t mean you’ll never want interaction variables
- Just because we have OOB error, doesn’t mean you’ll **never** use CV
- Start with a small number of trees at first, then increase
- Pickling makes a giant file (~GB)

Missing Values

- Typical implementation will use the median value
- Can first use the median values and then use proximities to calculate a more accurate missing value later