# Data, Decision Trees and Ensembles

## Machine Learning in Molecular Science

Prof. Michael Shirts

July 23rd, 2024

# Reminders

- All of yesterday's material is on GitHub (slides now, too!)
- Remember to work on teams during the breaks.
- Any questions now that you have thought about things more?

# The importance of good data

- GIGO
  - Garbage in, Garbage out
- Sometimes (**often?**) data is at least a bit erroneous
- Need to be **careful** about what you include in your data set
- For LARGE data sets, especially
- Anomaly detection
  - A problem in unsupervised learning

# One Problem: Missing Data?

- How bad is the problem?
  - Let's say we have 10,000 data points, and there are 100 features per point
  - Assume a 3% chance for each feature to be missing for every point
  - The chance that any given data point is actually complete is $(1\text{-}0.03)^{100} = 0.048$
  - Only 4.8% of the points have all features, despite the data being 97% complete!
  - That's leaving a lot of data if we only use feature-complete data!

# Data imputation

- Strategies:
  - Fill in missing data with using "around" it
  - What are the choices?
    - Mean of all other choices for that feature
    - Random selection of that feature
    - The mean of "close" data
    - Regression from other inputs
- Multiple imputation generates multiple guesses for each missing data point, which can help improve the statistics
- There are tools in pandas and scikit-learn to impute missing data.

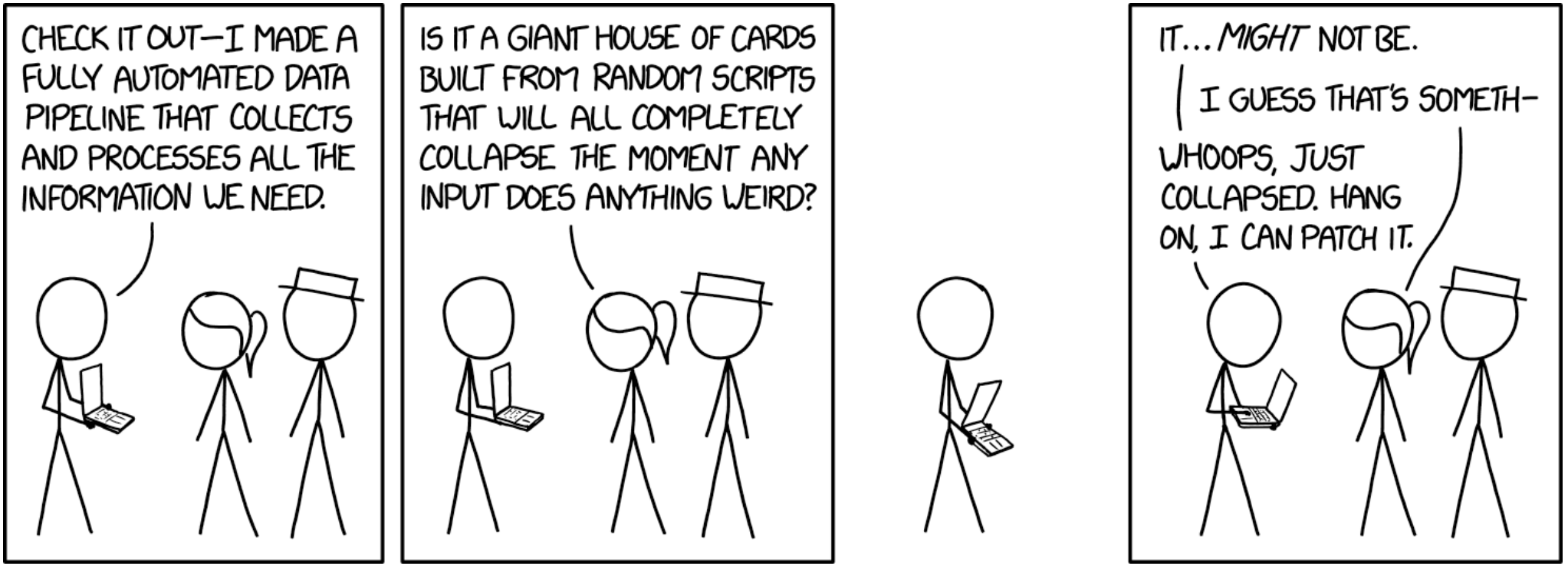
# Go to Notebook

# Another Thing to Check for: Data Leakage

- Does training data get into the testing set?
- Are some of your features actually surrogates of your labels?
- What is going to happen with ChatGPT 8 when it ends up getting trained on outputs of ChatGPT 4 through 7?

# Be careufi

- Two needs:
  - Automating data processing to ensure consistent treatment over the entire data set.
  - Carefully human curation of the data to make sure there's nothing the automation didn't handle.

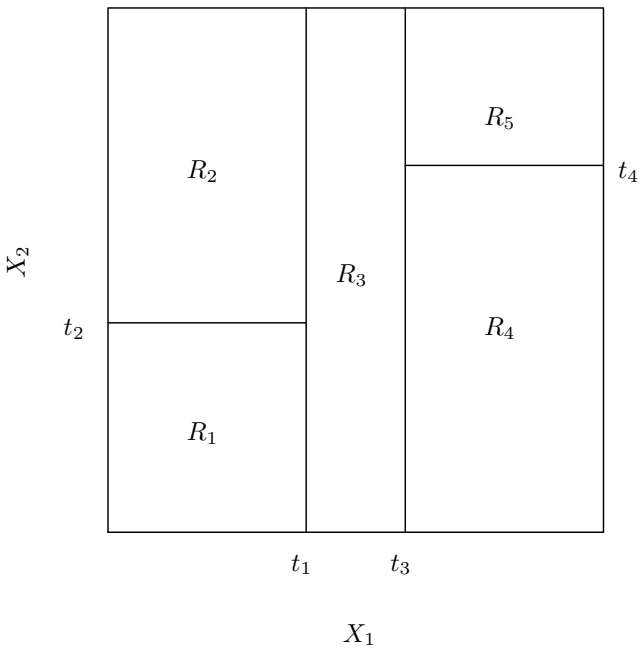



https://xkcd.com/2054/

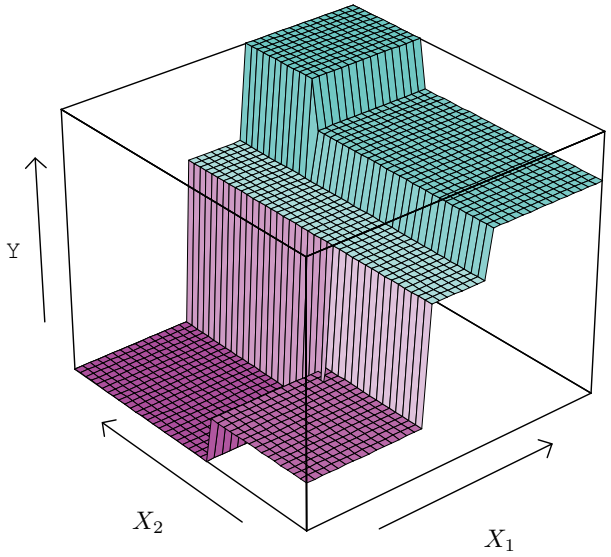
# Decision Trees and Random Forests

- Tree Methods
  - Regression trees
  - Classifier trees
- Notebook!
- Ensemble methods
  - Bagging, boosting, random forest
- Notebook!

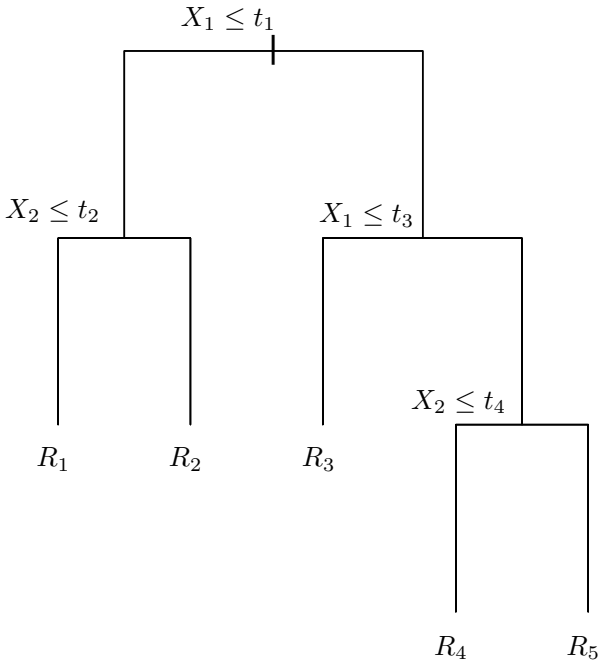
# Visualization of a decision tree




2D division of space

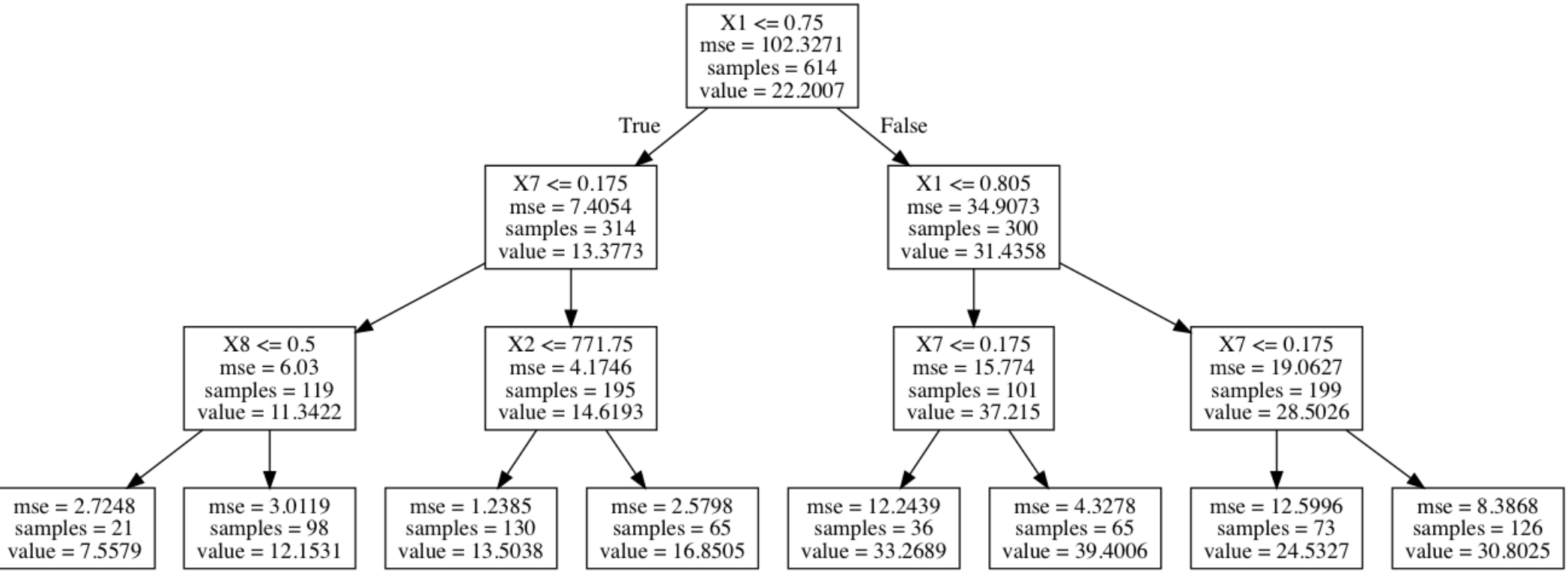



2D division of space versus prediction

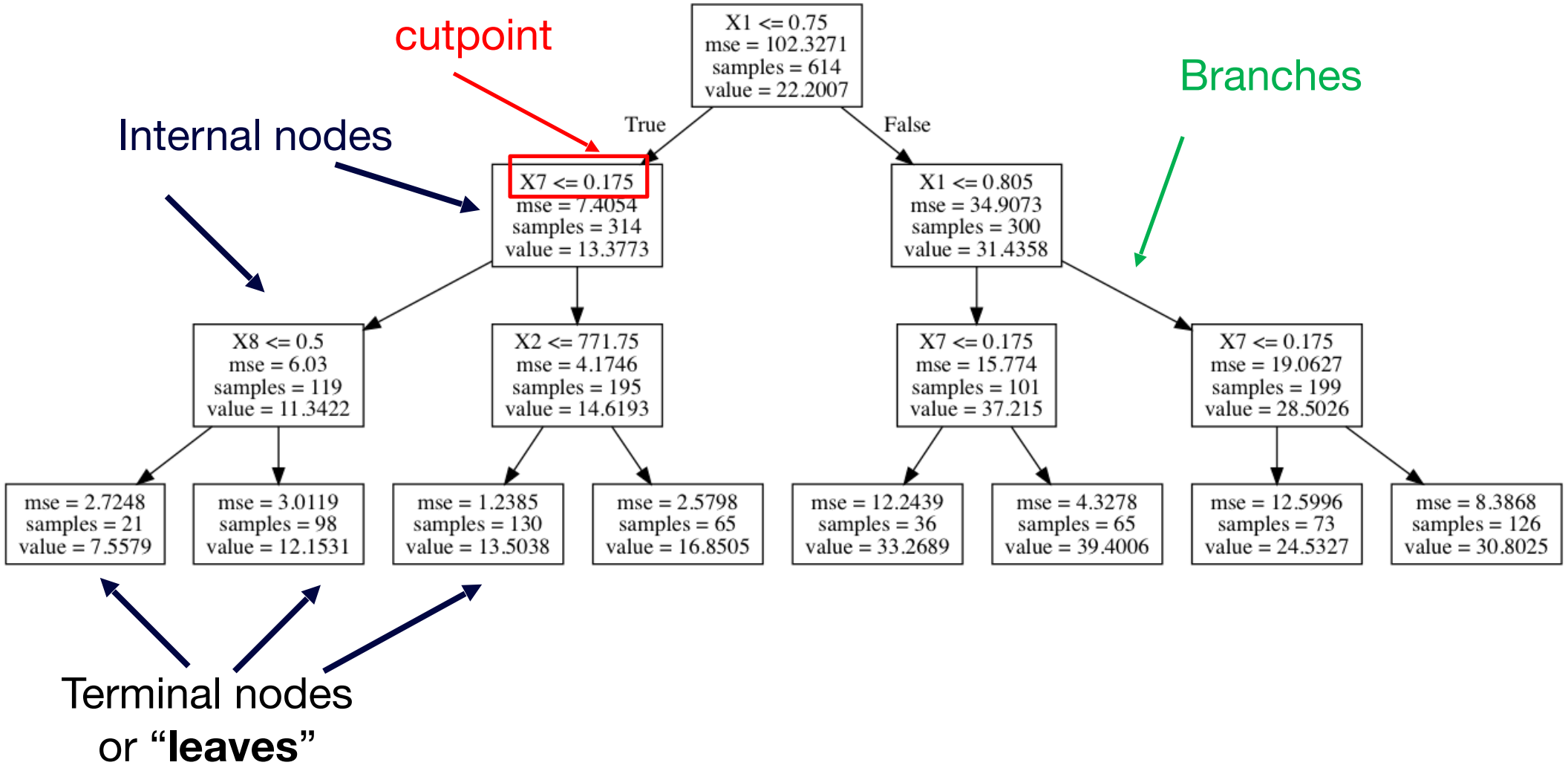



Tree diagram

# An example of a decision tree



# Decision tree prediction



# Building a regression tree

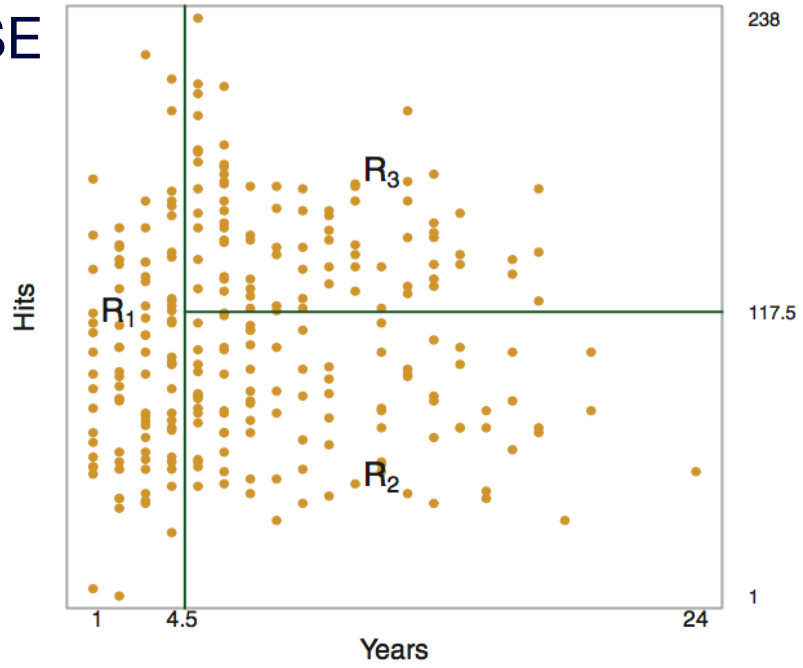
- We need a way to assign a value to our prediction
- The value we report for a prediction is the average of all of the training points placed in the same "container"

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2, \quad < \quad \sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2$$

Pick divisions that minimize the total RSS/MSE

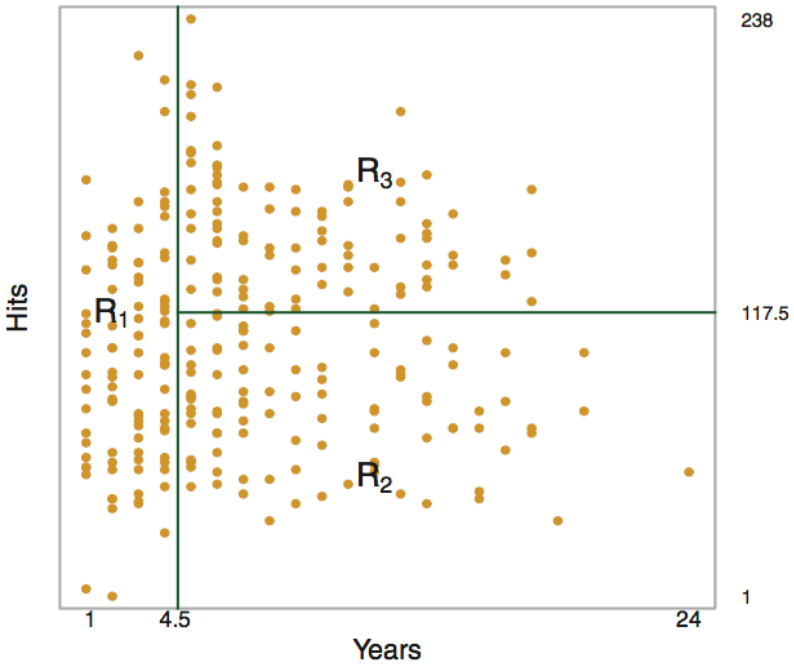
$$R_1(j,s) = \{X | X_j < s\} \text{ and } R_2(j,s) = \{X | X_j \geq s\},$$

A cutpoint breaks it into two regions



# Building a regression tree

- Example: The regression tree is divided into three regions ($R_1$-$R_3$) based on sub-divisions of the feature space
- This is done iteratively by
  - finding at each the biggest **decreases** in total RSS by appropriately selecting which predictor (j) and cutpoint give the biggest decrease in RSS…

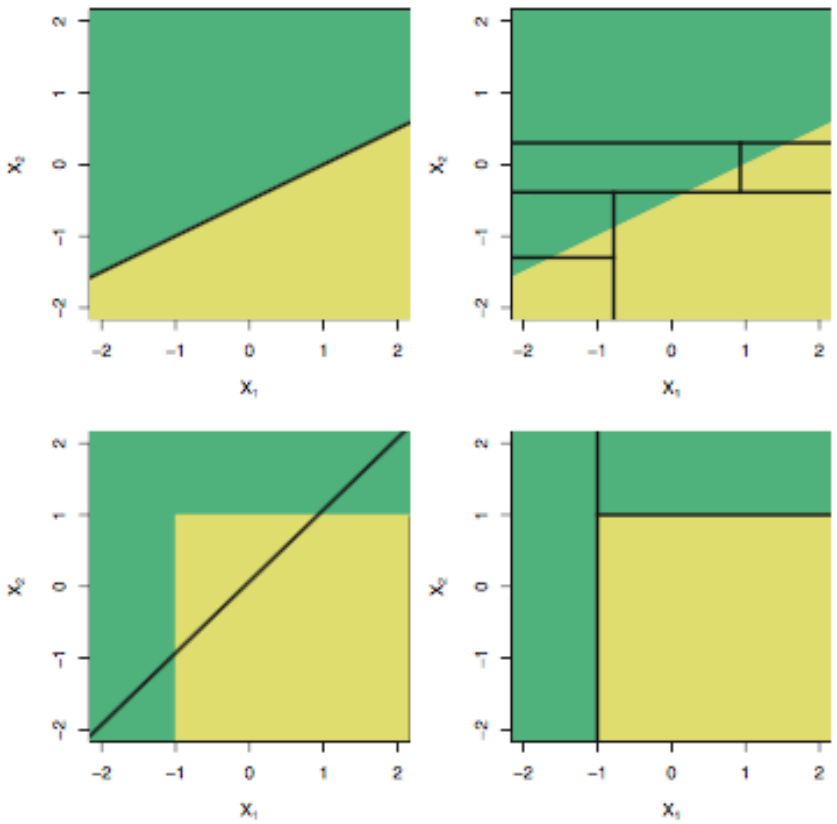


# Options for tree building

- Approach is known as “top down” or “greedy”: goes after the biggest reductions in RSS first (recursive binary splitting)
- Does not simulate **all** possible trees, so it is possible you are not finding the minimum RSS
- Also possible that trees designed in this approach can lead to overfitting (i.e. too much bias)
- The procedure of “tree pruning” can address this (we won't cover today)
- We **will** explore options/choices (hyperparameters!) in building our decision trees

# Big picture concepts in building a tree

- Decision trees are constructed as an alternative to regression models we have seen
- ISL Fig 8.7 shows some extreme examples of "regression vs. classification" and comparisons




- If relationships are inherently linear, decision trees won't be great
- If relationships have "cliffs" and other abrupt features, it will perform better than linear regression

# Big picture concepts in building a tree

- PROS:
  - Trees are easy to explain to non-experts
  - Trees can be displayed graphically (if small)
  - Very easy to interpret

- CONS:
  - Trees are highly sensitive to training data
  - Deep trees are very likely to overfit
    - Lots of cutpoints leads to high effective numbers of parameters

# To the notebook!

# Building better trees with ensembles

- Three common ensemble methods
  - Bagging
  - Random forests
  - Boosting
- Lots of jargon here, let's unpack!

# Ensemble methods reduce variance

- The error of a DT is highly dependent on the training set used – sometimes much more than in regression
- **Ensemble methods** are a way to avoid this
- Involves **bootstrapping** to create semi-new draws from the underlying distribution
- Original purpose of bootstrapping: error estimation
- **This time**, we go beyond error estimation to use ensemble methods to reduce variance by AVERAGING bootstrap draws
- **Important**: these methods get introduced in ISL in the context of decision trees, but can also be applied to **almost any other** high variance ML techniques

# Applying ensemble methods to decision trees: Bagging

- Bootstrap Aggregation = Bagging
- Bagging algorithm
  - Build a model based on B individual bootstrap data sets
  - Regression trees: Make predictions f(x) for each model and average the predicted response

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

  - Classifier trees: use “majority vote” (most common occurring response) to decide for each data point

# Estimating test error in bagging

- When you generate bootstrap samples, you can show that about 1/e ≈ 36% of the data points in the sample won't be used.
- Use THESE unused samples to compute test error for each bagging trial.
- These are called "out of bag" samples

# Applying ensemble methods to decision trees: Random Forest

- Random forest: random selection of predictors in bunch of trees
- Bagging methods work well, but the bootstrap sets can still be highly correlated,
  - Predictors with a lot of information gain (i.e. useful in predictions) will be at the top of most trees, making the final trees similar
- To reduce correlation, the random forest method uses a *random subset of predictors* at each split (node) in the tree
  - Heuristic is that $m = \sqrt{p}$ predictors are randomly chosen to use in each split, other predictors are ignored
- Do it a bunch of times like bagging

# Applying ensemble methods to decision trees: Random Forest

- Random forest scoring
  - Regression random forest: Make predictions f(x) for each model and **average** the predicted response

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(x).$$

  - Classifier random forest:
    - “majority vote” (most common occurring response) wins
    - Or average predicted probabilities and take those above/below 0.50

# To the notebook!

# Boosting

- Fit the data slowly instead of all at once.
- Fit a tree to the data;
- Fit a new tree to the difference between the data and the old tree.
- Fit a new tree to the difference between the data and the i+2 tree.
- Fit a new tree to the difference between the data and the i+3 tree.
- etc.

- Fitting data sequentially with an ensemble, rather than in parallel with an ensemble

# AdaBoost

- Standard tree booster in sklearn; there are lots of different variants

$$F_{t-1}(x_i) + \alpha_t h(x_i)$$

$$E_t = \sum_i E[F_{t-1}(x_i) + \alpha_t h(x_i)]$$

We select α each time we add a new 'learner' that minimizes the error with that new learner.

Final answer is a weighted ensemble of learners, trained sequentially on all the data, instead of in parallel on all of the data

# Gradient Boosting (like XGBoost)

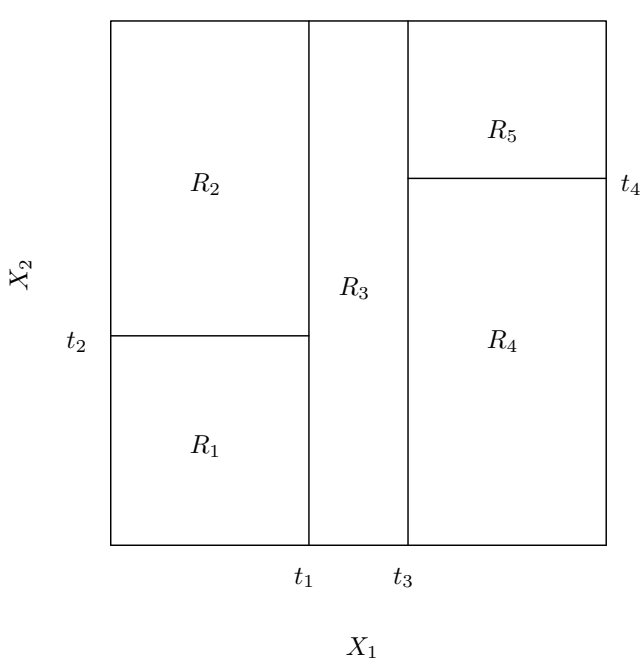
- Similar to AdaBoost
- Minimize the residuals with respect to the LAST boost.
- Very popular and fast

# To the notebook!

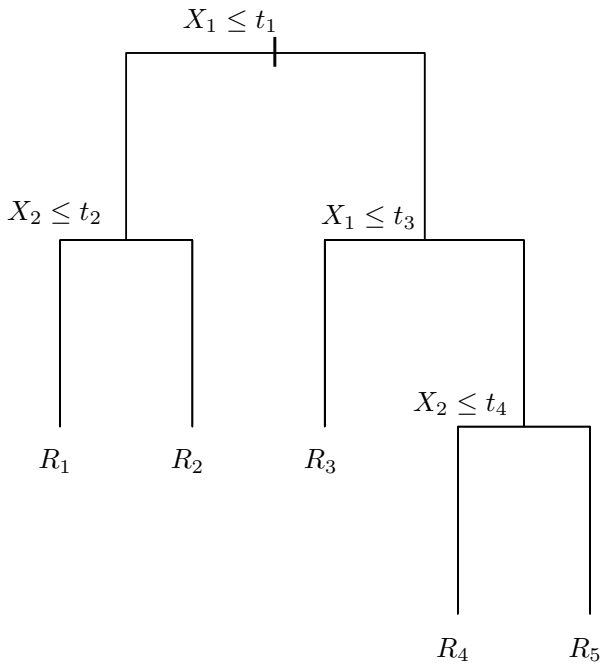
# Regression trees vs Classification trees

- Concepts are very similar
- Rather than **averaging** the values, you classify by a majority vote of what training set is in your bin
- Error metric is based on the classification error rate, not RSS, just as in K-nearest neighbors and related methods
- The decision tree leaves have your qualitative class assignment, not just a quantitative prediction

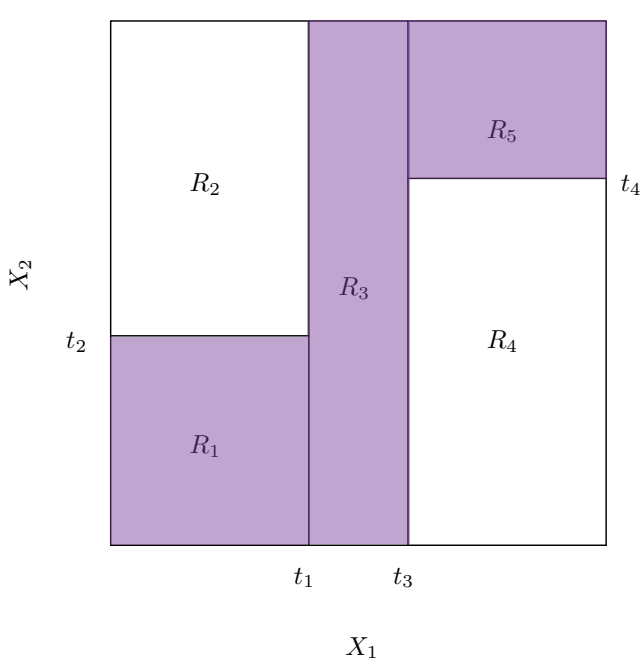
# Visualization of a classification decision tree




2D division of space




Tree diagram




2D division of space with assignment

# What is the 'best' split for classification?

- How do you decide on the best way to split a node?
  - We want nodes that are more 'pure'.

- What makes the splits the most 'pure'?

- Two common choices:

  - minimize Gini $\sum_{K} p_k(1 - p_k)$

  - minimize entropy $\sum_{K} - p_k \log p_k$

$P_k$ = percent of class k in the leaf

For two classes:
If 50/50, then Gini = 0.5
If 90/10, Gini = 0.18
If 100/0, Gini = 0.0

For two classes:
If 50/50, then entropy = 0.30
If 90/10, entropy = 0.195
If 100/0, entropy = 0.0

# Other Similar Approaches

- Support vector machines: Divide with linear partitions instead of straight lines up and down.
- Or even with curved lines!
- Otherwise quite similar to decision trees.