Random Forests, Runtimes, Modeling Overview

Learning goals:

- Learn the random forest algorithm and how it extends decision trees.
- Review runtime notation and analyze algorithms in 100.
- Give a big-picture overview of modeling in 100.

UC Berkeley Data 100 Summer 2019
Sam Lau

Also, see this neat animated recap of decision trees.
Sadly, they never got around to animating random forests.

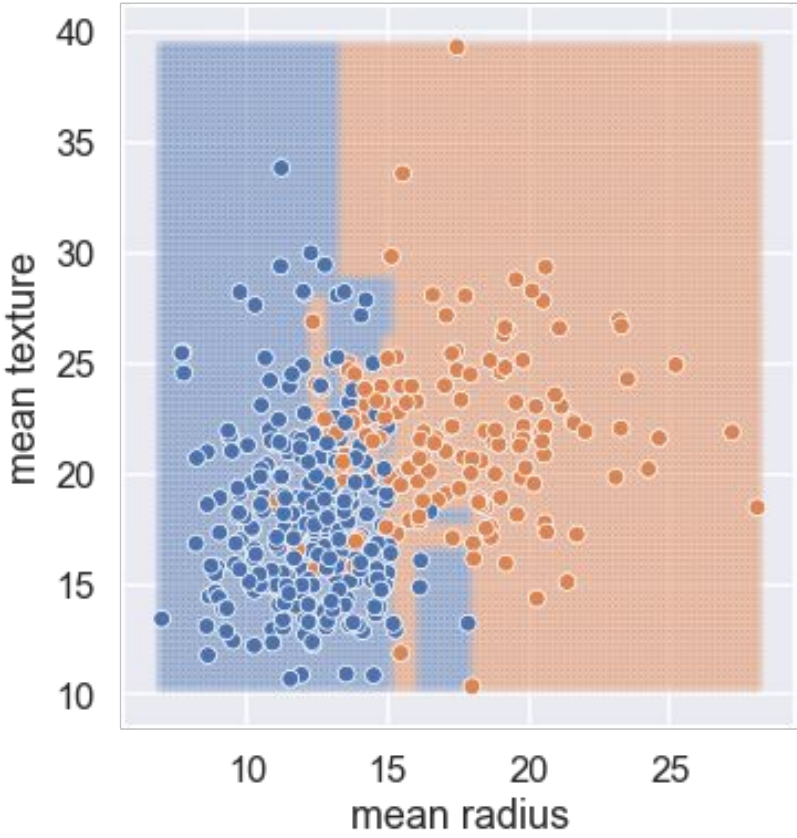
Announcements

- Project 3 out, due **Tues**
- Final next Thurs, Aug 15 9:30am-12:30pm in 10 Evans.
  - Can bring **two** handwritten double-sided cheat sheets.
  - Bring **pencil and eraser** (not pen).
  - Will provide midterm reference sheet again.
- If you have special accomodations for final, we will email you soon.
- Tomorrow is last lecture! No screencast planned.

Random Forests

Problems with Decision Trees

- Fully-grown decision trees will almost always overfit data.
  - Low model bias, high model variance
- IOW: Small changes in dataset will result in very different decision tree.
- Idea: Use the **bootstrap** to fit many trees, then take average over the trees.



Bagging

- **Bagging** is short for Bootstrap AGGregatING.
  - Generate bootstrap resamples of training data.
  - Fit one model for each resample.
  - Final model = average predictions of each small model.
- Invented by Leo Breiman in 1994 (Berkeley Statistics!)

(Demo)

Random Forests

- Bagging often isn't enough to reduce model variance!
  - Decision trees often look very similar to each other.
  - E.g. one strong feature always used for first split.
- Idea: Only use a sample of m features at each split.
  - Usually $m = \sqrt{p}$ for classification and $p/3$ for regression.
- This tries to make individual trees overfit in different ways so that overall forest has low variance.

It might feel a bit odd that we decrease variance by making individual trees more different from each other. This works because if all the trees vary in slightly different ways, the average tree will stabilize out.

Random Forests Algorithm

- Bootstrap training data T times. For each resample, fit a decision tree by doing the following:
  - Start with data in one node. Until all nodes pure:
  - Pick an impure node.
  - Pick a random subset of m features. Out of those features, find feature j and value θ that minimize loss.
  - Split into two child nodes (one for $x_j < \theta$, one for $x_j \geq \theta$).
- To predict, ask the T decision trees for their predictions and take majority vote.

(Demo)

You should understand: How do T and m affect the bias-variance tradeoff?

Why Random Forests?

- Versatile: does both regression and classification
- Invariant to feature scaling and translation
- Automatic feature selection
- Nonlinear decision boundaries without complicated feature engineering
- Doesn't overfit as often as other nonlinear models (e.g. polynomial features)
- Example of **ensemble method**: combine the knowledge of many simple models to make a sophisticated model.
- Example of using **bootstrap** to reduce model variance.

Break!
Fill out Attendance:
http://bit.ly/at-d100

Runtime Considerations

Know Thy Runtimes

- Modeling often limited by runtime of algorithm.
- Suppose we have an (n x p) design matrix X.
- We use **big-O notation** to describe runtimes:
  - E.g. O(np) means that doubling the number of sample points (or number of features) will double the runtime
  - $O(n^2)$ means that doubling number of sample points will increase runtime by 4x (quadratic time).
- Typically, runtimes ≥ quadratic time will be too slow!
  - n and p can both be large.

Practice with Runtimes

- Find the runtimes of:
  - Looping once through all values in dataset.
  - Looping twice through all values in dataset.
  - Taking the mean of a column.
  - Taking the means of all columns.
  - Doing a dot-product of two length-p vectors.

Practice with Runtimes

- Looping once through all points in dataset.
  - O(np)
- Looping twice through all points in dataset.
  - O(2np) = O(np)
- Taking the mean of a column.
  - O(n)
- Taking the means of all columns.
  - O(np)
- Doing a dot-product of two length-p vectors.
  - O(p)

If these were hard for you, review the material from CS61A.

Rules of Thumb

- Very roughly speaking, vanilla Python can do about 1 million calculations in a second.
- E.g. Suppose dataset has 1,000,000 rows and 10 columns.
- Estimate that an O(np) algorithm takes ~10 seconds.
- But an $O(n^2)$ algorithm takes > 11 days (!)
- Keep runtimes in mind so that you can figure out whether an algorithm will finish in a reasonable amount of time.

The 1 million calculations per second is just Sam's rule of thumb. But it gets close enough most of the time.

Gradient Runtimes

- How long does it take to compute the average gradient for linear regression + MSE loss?

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{n} \sum_{i}^{n} [-2(y_i - \boldsymbol{\theta} \cdot \boldsymbol{X_i})(\boldsymbol{X_i})]$$

- Inner sum: O(p) time since dot product is O(p).
- Calculate inner sum O(n) times, so total time is O(np).

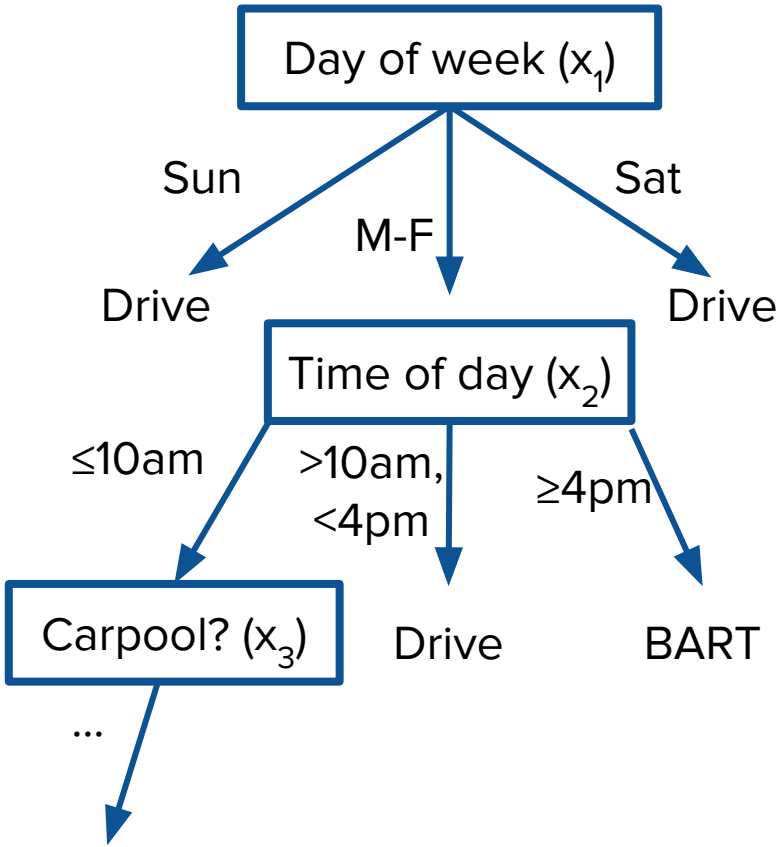
Gradient Descent Runtimes

- Let $\epsilon > 0$ be the desired error after running GD.
- Batch GD takes $O(\log(1/\epsilon))$ iterations to converge.
- SGD takes $O(1/\epsilon)$ iterations to converge.
    - If you want to get very close to the minimum loss, you will have to pay the price and iterate more.
- Batch GD takes fewer iterations than SGD, but each iterations takes a lot longer to compute!
- BGD: $O(np)$ per iteration, $O(\log(1/\epsilon))$ iterations, overall runtime = $O(np * \log(1/\epsilon))$
- SGD: $O(p)$ per iteration, overall runtime = $O(p / \epsilon)$

Running Cross-Validation

- Suppose we run K-fold CV for linear regression. What's the runtime?
- Each linear regression model takes $O(p / \epsilon)$ time to fit.
- Since we fit K of them, $O(Kp / \epsilon)$ time overall.

Decision Trees



What's the runtime of making a prediction using a decision tree?

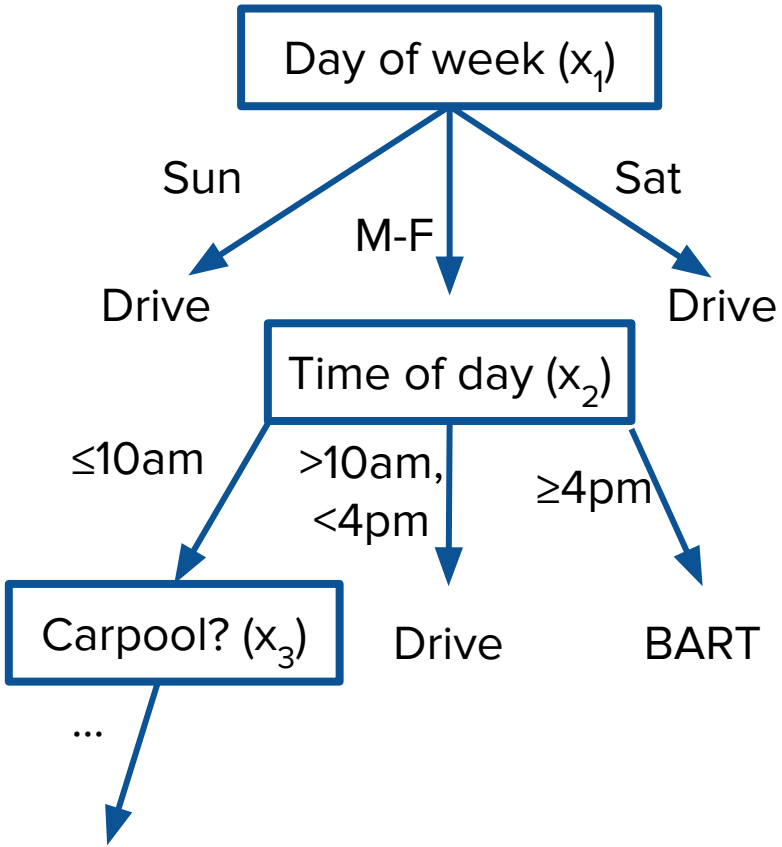
O(1) time for each rule. How many rules do we check?

Say that d = maximum depth of decision tree.

Making prediction: O(d) time.

- Usually, d ≃ log(n)
- But sometimes, d ≃ n.

Trees and Forests



Let d be depth of tree.

Fitting a decision tree takes O(npd) time.

Usually $d \approx \log(n)$, so $O(np \cdot \log(n))$ time. This is reasonable!

If we fit T trees for a random forest, this takes O(Tnpd) time.

Modeling Overview

You've learned a lot!

- How do you estimate a population parameter?
  - Create an estimator and analyze its bias and variance.
  - Create a bootstrap CI for the parameter
- How do you predict numeric values?
  - Linear regression
  - Decision trees / Random forests
- How do you predict categorical values?
  - Logistic regression
  - Decision trees / Random forests

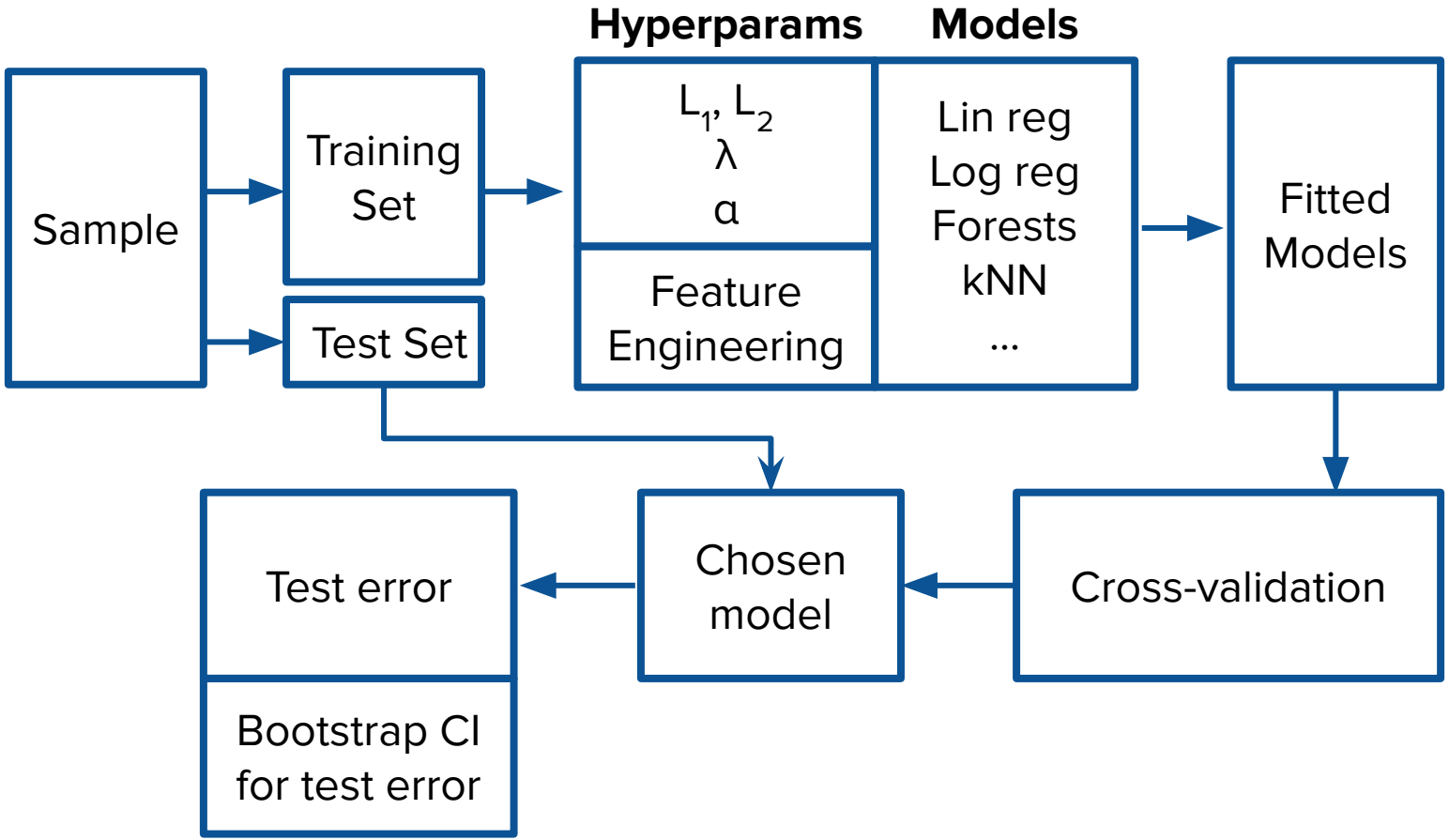
You've learned a lot!

- How do you fit a model?
  - Find gradient, run stochastic gradient descent.
- How do you know whether your model is doing well?
  - Cross-validation
  - Plot validation / learning curves
  - Plot confusion matrix and ROC curve (for classification)
- How do you improve a model?
  - Feature engineering
  - Adjusting regularization
  - Adjusting hyperparameters (e.g. number of trees in RF)

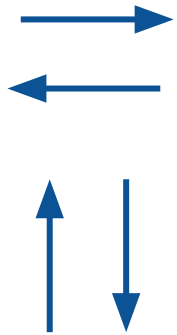
You've learned a lot!

- Why do some models do better than others?
  - Bias-variance tradeoff
- Why do some models run faster than others?
  - Runtime analysis
- What if our dataset is very large?
  - Distributed storage and computing

Putting it all together:



Validation Set



Validation Set

(Demo)