Section 2: Machine Learning

Valentine Gilbert

February 10, 2022

Overview

Motivation

- Most of what we'll talk about in this class is **causal inference** – we care about $\hat{\beta}$
- ML is typically not about causal inference, but about **prediction** – we care about $\hat{Y}_i$
- Causal relationships tend to have clear policy implications. The policy implications of predictions can be less clear.
  - Example: Educational attainment predicts income. What should we do with this information?
- Kleinberg et al. (AER P&P 2015) define prediction policy problems as policy problems in which the payoff of a policy intervention depends on predicting some outcome. For example:
  - Should I carry an umbrella with me today?
  - Should an elderly patient receive hip replacement surgery?
  - Who should receive cash assistance?
  - Should an arrestee be detained pending trial?

Out-of-Sample Predictions

We want our predictions to be accurate **out-of-sample**

- That is, we want accurate predictions for newly observed data
- So our prediction problem is that we observe **training data** D on outcome y and a vector of covariates x and want to estimate $f(x)$ to predict y for observations where we don't observe the outcome
- The more information we use, the better our **in-sample** predictions will be
- But some patterns we observe in D may just be sampling variation that doesn't generalize **out-of-sample**

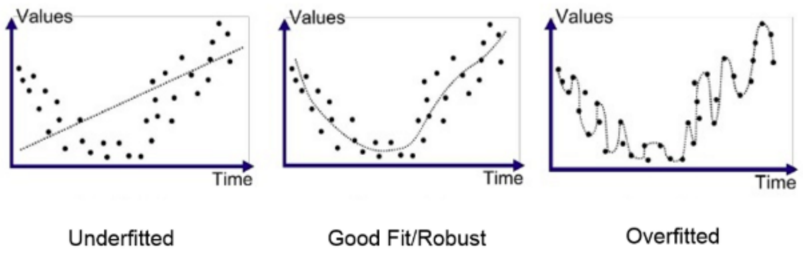
Bias-Variance Trade-off: Decomposition

- Suppose we see covariates x_0 of a new observation and want to predict y_0 using $\hat{f}(x; D)$, a prediction function estimated using data D
- Let $y = f(x) + \varepsilon$, where ε has variance σ^2
- We seek to minimize the out-of-sample mean squared prediction error, $E_D[(y_0 - \hat{f}(x_0; D))^2]$.
- This prediction error can be decomposed into the variance of the predictor and the squared bias of the predictor:

$$\begin{aligned}
E_D[(y_0 - \hat{f}(x_0; D))^2] &= E_D[(f(x_0) + \varepsilon - \hat{f}(x_0; D))^2] \\
&= E_D[(f(x_0) - \hat{f}(x_0; D))^2] + \sigma^2 \\
&= E_D[\hat{f}(x_0; D)^2] - E_D[\hat{f}(x_0; D)]^2 \\
&\quad + E_D[\hat{f}(x_0; D)]^2 + f(x_0)^2 - 2f(x_0)E_D[\hat{f}(x_0; D)] + \sigma^2 \\
&= \underbrace{E_D[\hat{f}(x_0; D)^2] - E_D[\hat{f}(x_0; D)]^2}_{\text{Variance of } \hat{f}(\cdot)} + \underbrace{E_D[(f(x) - \hat{f}(x_0; D))]^2}_{\text{Squared bias of } \hat{f}(\cdot)} + \sigma^2
\end{aligned}$$

Bias-Variance Trade-off: Intuition

- On average, more complicated models capture more of the relationship between y and x (have less bias)
- But more complicated models also change more with variation in the training data (have greater variance)
- The bias-variance trade-off is a trade-off between overfitting and underfitting your model

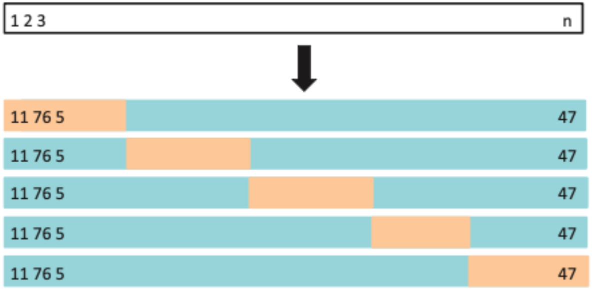


Model Assessment: Intro

- We've seen that minimizing out-of-sample prediction error requires selecting a model that strikes the right balance between simplicity and complexity.
- To choose the right model, we can assess the prediction error of various candidate models by training them on one set of data and evaluating the prediction error on another.
- The simplest way to do this would be to randomly divide your data in two, one for training the model and one for assessing its performance.
- But this method has two drawbacks:
    - It may overestimate the prediction error, since the training set has fewer observations than are available in total
    - The estimated prediction error can depend on how you happen to split the data

Model Assessment: k-fold Cross-Validation

- k-fold cross validation provides a better way to assess model performance.
- To implement k-fold cross validation:
  - Randomly divide the training data into k equal subsets (called folds)
  - For each fold, use the $k - 1$ other folds to train the model
  - Then predict outcomes on the held out fold
- In the end, you will have an out-of-sample prediction for every observation in your training data.



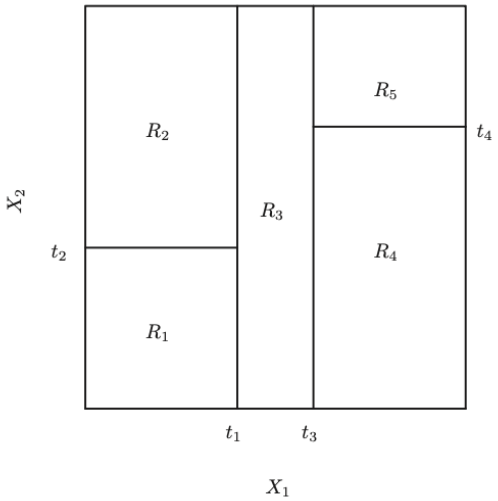
Searching over the Model Space

- Now that we know how to assess the performance of a given model, we have to decide which models to assess.
- Suppose you want to use OLS for prediction and you have k potential regressors.
- That gives you 2^k possible models to search over!
- If you transform some of your regressors (e.g. by adding polynomial terms or interactions), the model space grows even larger.
- It quickly becomes computationally infeasible to search over the entire model space.
- The problem is made tractable by setting a small number of tuning parameters that govern the complexity of the model. For example:
  - Tree depth in CART
  - The regularization parameter, λ, in LASSO: $\min_\beta RSS + \lambda \sum_j |\beta_j|$
  - The number of trees and depth of trees in Random Forest

Decision Trees

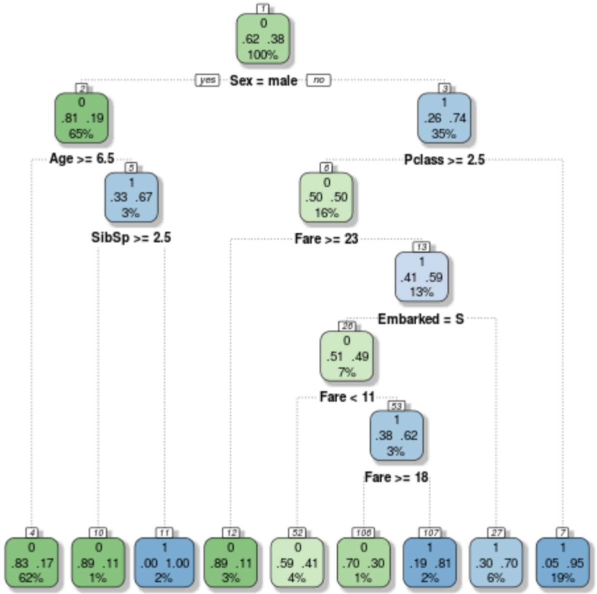
Decision trees make predictions by partitioning the covariate space into boxes

- Boxes created by a series of binary splits
- Observations in the same region get the same predictions
- More boxes $\rightarrow$ more complexity
- Example: Predicting survival of passengers on the Titanic based on age, sex, and class



So how are the splits chosen?

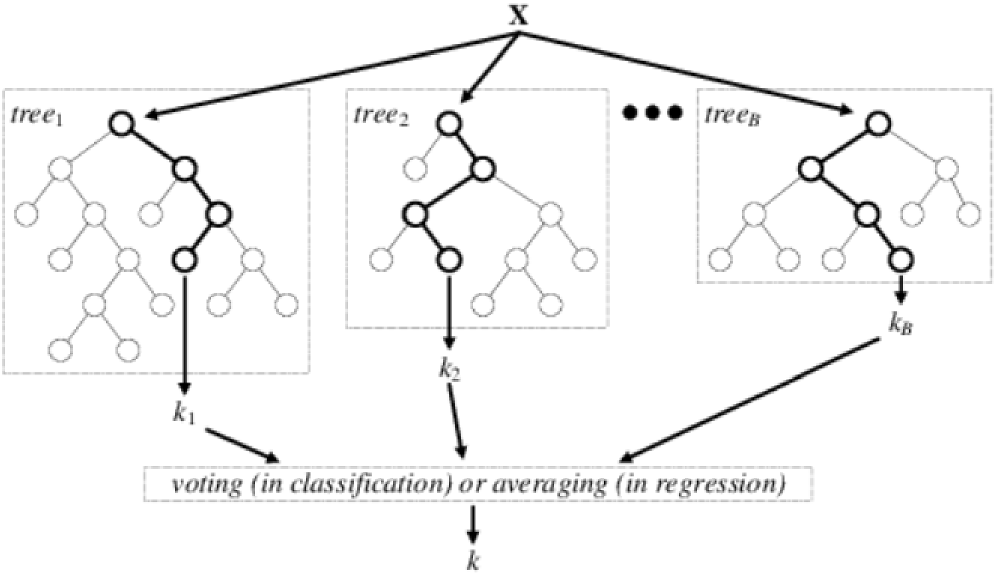
- Ultimately want to minimize $RSS = \sum_j \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$
- Each split is chosen to maximize the reduction in *RSS*
- This is a “greedy” algorithm – it chooses splits to maximize the immediate improvement in the objective function, not to minimize the final *RSS*
- Why? Computationally intractable to search over all possible trees and choose the best one



Random Forests

Random forests improve on decision trees by growing many different trees (e.g. 500) and averaging their predictions

- Each tree is grown on a bootstrapped sample
- Each time a tree splits, it only considers a random subset of covariates to split on $\rightarrow$ decorrelates trees
- Gives “smoother” and lower variance predictions
- Two parameters controlling complexity: Tree depth and the size of the subset of covariates
- Shallow trees and subset size $\sqrt{k}$ often gives good results


X
$tree_1$
$tree_2$
$tree_B$
k_1
k_2
k_B
voting (in classification) or averaging (in regression)
k

Kleinberg et al. (2018): Summary

- After the police arrest someone, a judge must decide whether that person should await trial at home or in jail.
- This decision is legally required to be based only on the judge's prediction of whether the defendant is likely to re-offend before the trial or fail to appear at the trial.
- Kleinberg et al. compare how well human bail judges in NYC do in this task compared to what could be achieved by a machine learning algorithm (gradient boosted decision trees).
- They find substantial improvements possible by using an algorithm.

Kleinberg et al. (2018): Selective Labels

- Suppose we do the following to evaluate the algorithm relative to the human judge:
  - Train the algorithm to predict re-offending rates using individuals released by the judge
  - Predict the expected re-offending rates of individuals detained by the judge
  - Predict how much re-offending would occur if the AI judge matched the human judge's detention rate by detaining the defendants with the highest predicted risk
- Any problems?
- We only observe outcomes for defendants that the judge decided to release! If judges observe and use information that the algorithm doesn't use, our comparison will be biased in favor of the algorithm.
- Kleinberg et al. call this the "selective labels" problem and it's the primary challenge they have to overcome in the paper.

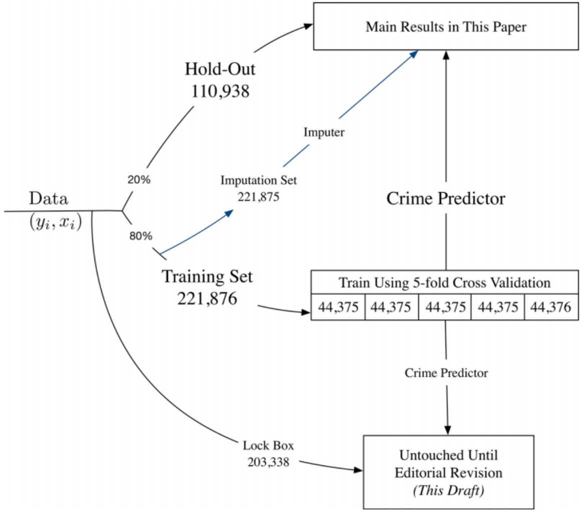


FIGURE I

Partition of New York City Data (2008–13) into Data Sets Used for Prediction and Evaluation

Kleinberg et al. (2018): Contraction

- The authors use the fact that cases are randomly assigned to judges of different leniency to overcome the selective labels problem.
- They predict what would happen if the pool of defendants released by a more lenient judge was “contracted” by the algorithm to match the detention rate of a less lenient judge.
- For example, if judge A releases 90% of defendants, we can observe amount of re-arrest that would occur if the algorithm detained enough additional defendants to match judge B’s release rate of 80%.
- We can then compare the re-offending rates of defendants released by judge B to those released by judge A and the algorithm.

- Figure compares performance of judges at different quintiles of leniency to predicted performance of algorithm
- Show that algorithm dominates stricter judges: Can either hold detention rate constant and decrease crime or hold crime constant and decrease detention rate (or a little of both)
- E.g. Can maintain crime rate of the least lenient judges while decreasing detention rate by ~ 12 percentage points

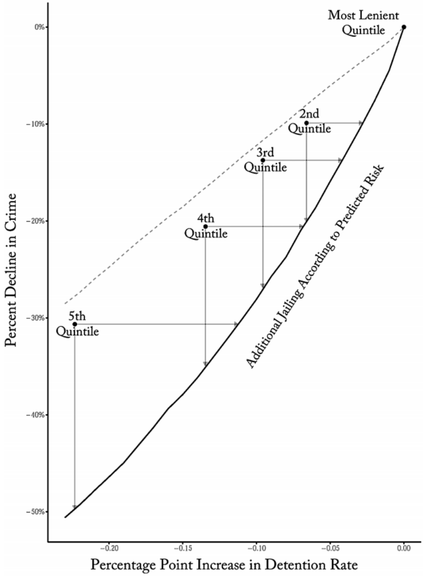


FIGURE V

Does Jailing Additional Defendants by Predicted Risk Improve on Judges? Contraction of the Most Lenient Judges' Released Set

- Might worry that more lenient judges are more skilled at using unobservables
- If so, then defendants released by lenient judges will be less likely to be re-arrested than similar defendants released by stricter judges
- Figure shows that algorithm trained on defendants released by most lenient judges accurately predicts re-arrest of defendants released by stricter judges

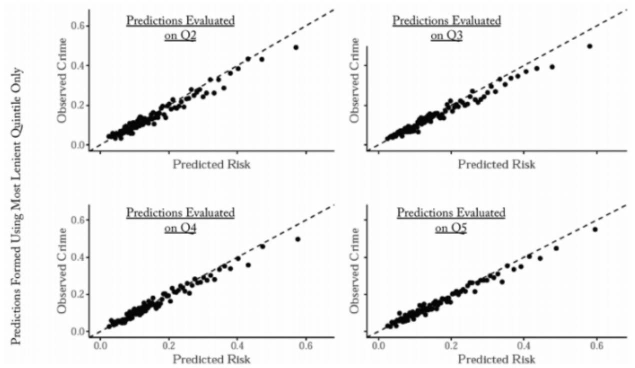


FIGURE IV

Do Judges of Different Leniency Screen Differently on Unobservables? Evaluated Predictors Formed Using Most Lenient Quintile on Other Quintiles