450c - Random Forests and GAMs

Apoorva Lal, Zuhad Hai

April 30, 2021

Stanford

Questions?

Nonparametric Regression: Setup

- Introductory metrics classes teach you to model $\mathbb{E}[Y|X] = \mathbf{x}_i'\boldsymbol{\beta} + \varepsilon_i$
- This is almost always wrong
    - Angrist and Pischke: OLS yields best linear approximation
- for interpretable $\boldsymbol{\beta}$s, this might be worth the approximation error
- When one is simply interested in curve fitting, however, we can do better
- General problem formulation

$$y = f(\mathbf{x}) + \varepsilon$$

Nonparametric Regression: Estimators

- Given a random pair $(\mathbf{x}, y) \in \mathbb{R}^d \times \mathbb{R}$, the regression function is $m_0(\mathbf{x}) = \mathbb{E}[Y|\mathbf{X} = \mathbf{x}]$
- Approximated with following smoother

$$\widehat{m}(\mathbf{x}) = \sum_{i=1}^{n} w_i(\mathbf{x}) y_i$$

- **KNN Regression**: Fix an integer $k \geq 1$; $\widehat{m}(\mathbf{x}) = \frac{1}{k} \sum_{i \in \mathfrak{N}_k(\mathbf{x})} y_i$ where $\mathfrak{N}_k(\mathbf{x})$ is the k-neighbourhood of $\mathbf{x}$ which contains the indices of the k closest points.
- **Nadaraya-Watson/Kernel Regression** Given a kernel function $K(\cdot)$,

$$\widehat{m}(x) = \frac{\sum_{i=1}^{n} K(\frac{x - x_i}{h}) y_i}{\sum_{i=1}^{n} K(\frac{x - x_i}{h})}$$

Regression Trees: Basics

Suppose we have a training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, a test point $\mathbf{x}$, and a tree predictor

$$\widehat{m}(\boldsymbol{x}) = \sum_{i=1}^{n} w_i(\boldsymbol{x}) y_i \quad \text{where} \quad w_i(\mathbf{x}) = \frac{\mathbb{1}_{\mathbf{x}_i \in \mathcal{L}(\mathbf{x})}}{|i : \boldsymbol{x}_i \in \mathcal{L}(\mathbf{x})|}$$

where $\mathcal{X}$ is partitioned into leaves $\mathcal{L}(\mathbf{x})$, where leaves are constructed to maximise heterogeneity between nodes.

Regression Trees in a picture

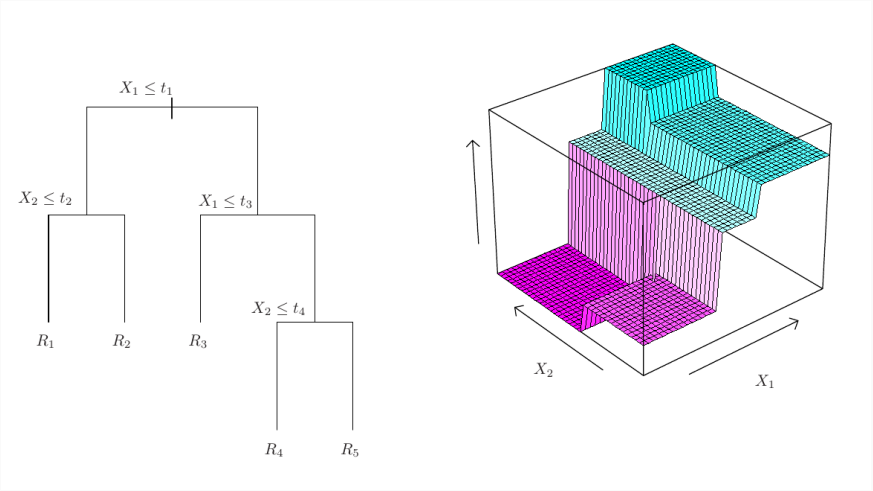


Figure 1: Chapter 8 - 'Computer Age Statistical Inference', Efron and Hastie

CART < Random Forests

- Do this until all leaves have $2\times$ `minimum leaf size` observations.
- Regression trees overfit, so we need to use cross-validation + other tricks.
- Random forests build and average many different trees T^* by
    - Bagging / subsampling training set (Breiman)
    - Selecting the splitting variable at each step from m out of p randomly drawn features (Amit and Geman)
- We want to compute the average $\bar{y}$ for every partition of the data, where the partition is a unique combination of covariates.

Random Forests: Intuition

- To mitigate concern, we introduce random sampling across variables (select z of the J variables)
- When different variables are selected, we will also observe different nodes / trees!
- In general, no universal advice on how deep we should grow these trees / how many trees we want
- Tree depth comes at a bias-variance tradeoff: the less data we have in each node, the more do we run the risk of overfitting.
- As usual, pick 'optimal' set of hyperparameters using cross-validation

Random Forests: Implementation

Let’s prepare our data.

```
library(randomForest)
library(mlbench)
library(caret)

data(Sonar)
df <- Sonar
x <- df[, 1:50]
y <- df[, 51]
```

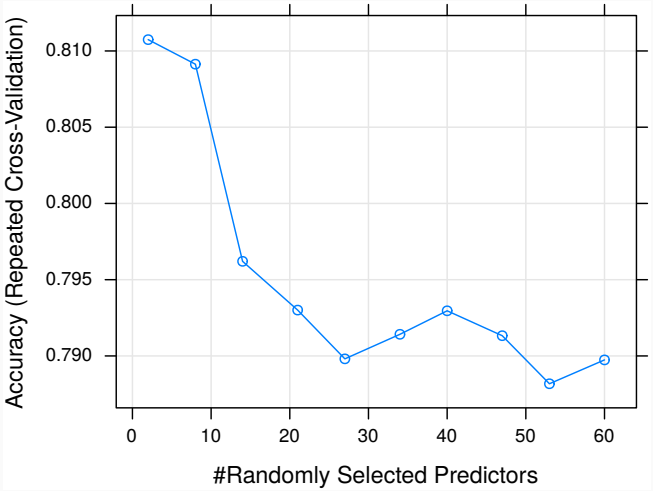
Random Forests: Implementation

Fit the model.

```
control <- trainControl(method = "repeatedcv",
                        number = 3, repeats = 3)
metric <- "Accuracy"
rf_random <- train(Class ~ ., data = df, method = "rf",
  metric = metric, tuneLength = 10, trControl = control)
```

Random Forests: Implementation

```
plot(rf_random)
```



Random Forests: Tuning

- mtry is the number of variables to split on. In applications, tune on multiple hyperparameters (`expand.grid` is your friend)

```
tg <- expand.grid(.mtry = c(10:15))
rf_grid <- train(Class ~ ., data = df, method = "rf",
  metric = metric, tuneGrid = tg, trControl = control)
```

Random Forests: Implementation

```
print(rf_grid)
```

```
## Random Forest
##
## 208 samples
##  60 predictor
##   2 classes: 'M', 'R'
##
## No pre-processing
## Resampling: Cross-Validated (3 fold, repeated 3 times)
## Summary of sample sizes: 139, 138, 139, 138, 139, 139, ...
## Resampling results across tuning parameters:
##
##   mtry  Accuracy  Kappa
##   10    0.8157    0.6249
##   11    0.8077    0.6086
##   12    0.8093    0.6120
##   13    0.8157    0.6255
##   14    0.8141    0.6220
##   15    0.8013    0.5956
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was mtry = 13.
```

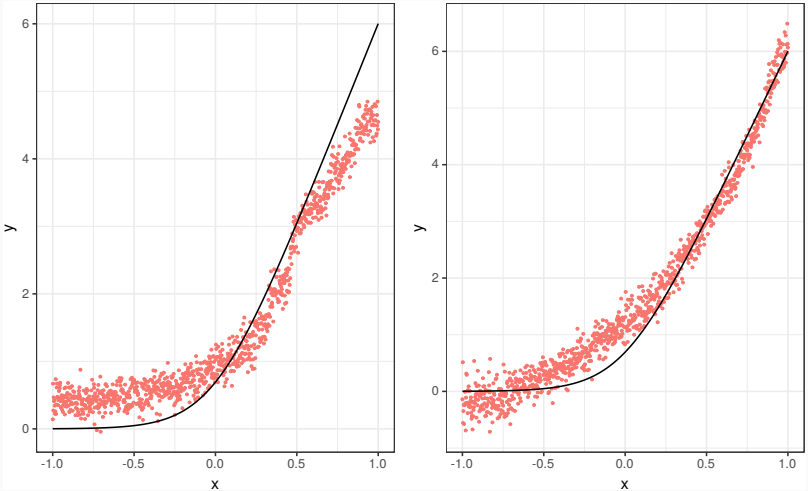
Generalised Random Forests ()

- RFs can suffer in the presence of smoothness
- Instead of fitting a local average, Friedberg et al propose fitting a linear regression within each leaf (with ridge regularisation)

$$\begin{pmatrix} \hat{\mu}(x_0) \\ \hat{\theta}(x_0) \end{pmatrix} = \text{argmin}_{\mu,\theta}\{\sum_{i=1}^{n} \alpha_i(x_0)\left(Y_i - \mu(x_0) - (x_i - x_0)\,\theta(x_0)\right)^2 + \lambda \left\|\theta(x_0)\right\|_2^2\}$$

Conventional RF vs Local Linear Forest

```
(g1 | g2)
```



Generalised Additive Models: Intuition

- GAMs are a family of semi-parametric regressions that introduce non-linearity in the form of **basis expansions**:

$$\mathbb{E}\left[Y|\mathbf{X}=\mathbf{x}\right]=\alpha_0+\sum_{j=1}^{p}m_j(\mathbf{x}_j)$$

Includes the family of smoothing splines, which solve

$$\hat{\beta}_{\text{SS}}=\underset{\boldsymbol{\beta}}{\operatorname{argmin}}\|\mathbf{Y}-f(\mathbf{X})\|^2+\lambda\int\left\{f''(t)\right\}^2dt$$

- GAMs allow us to interpret the relationship between any variable and the outcome in a bivariate plot

Generalised Additive Models: Implementation

- Let’s compare OLS and GAM results.
- Data and example taken from Peisakhin and Rozenas (2018) [AJPS]
- How does exposure to Russian propaganda media sources affect political behaviour in Ukraine?

Peisakhin and Rozenas (2018) Data

```
d <- read.csv("data.csv")
d <- na.omit(d)
d %>% glimpse
```

```
## Rows: 3,567
## Columns: 36
## $ precinct          <int> 590884, 590885, 590886, 590887, 590888, 590889, 59089~
## $ oblast            <chr> "Сумська", "Сумська", "Сумська", "Сумська", "Сумська"~
## $ places            <chr> " СУМИ", " СУМИ", " СУМИ", " СУМИ", " СУМИ", " СУМИ",~
## $ noplaces          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,~
## $ type              <chr> "city", "city", "city", "city", "city", "city", "city~
## $ size              <int> 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 3, 3, 3, 3, 3,~
## $ ukrainian         <dbl> 77.44, 77.44, 77.44, 77.44, 77.44, 77.44, 77.44, 77.4~
## $ district14par     <int> 157, 157, 157, 157, 157, 157, 157, 157, 157, 157~
## $ registered14par   <int> 1552, 2368, 1564, 2152, 2396, 2324, 1812, 1544, 1972,~
## $ voted14parl       <int> 844, 1370, 887, 1252, 1386, 1216, 950, 755, 1178, 652~
## $ oppblock14par     <dbl> 0.04976, 0.04088, 0.03720, 0.05192, 0.05267, 0.05099,~
## $ porosh14par       <dbl> 0.2500, 0.2504, 0.2559, 0.2740, 0.2648, 0.2558, 0.266~
## $ r14parl           <dbl> 13.981, 11.241, 11.499, 11.502, 12.049, 14.474, 10.42~
## $ turnout14parl     <dbl> 0.5438, 0.5785, 0.5671, 0.5818, 0.5785, 0.5232, 0.524~
## $ registered14pres  <int> 1543, 2318, 1559, 2167, 2369, 2313, 1788, 1440, 1959,~
## $ voted14pres       <int> 1005, 1593, 1080, 1494, 1618, 1450, 1191, 885, 1345, ~
## $ turnout14pres     <dbl> 0.6513, 0.6872, 0.6928, 0.6894, 0.6830, 0.6269, 0.666~
## $ r14pres           <dbl> 9.353, 8.914, 8.889, 7.831, 8.529, 9.517, 7.557, 13.6~
## $ Poroshenko        <int> 645, 978, 669, 913, 1043, 904, 708, 478, 848, 397, 99~
## $ district          <int> 157, 157, 157, 157, 157, 157, 157, 157, 157, 157~
## $ registered12      <int> 1595, 2408, 1588, 2200, 2405, 2377, 1808, 1543, 1926,~
```

Linear Benchmark

```r
form0 <- formula("r14pres ~ qualityq + distrussia +
    ukrainian + r12 + I(100*turnout12) + I(type == 'village') +
    log(registered14pres) + log(roads + 1) | Raion")

m0 <- feols(form0, data = d, cluster= ~Raion)
summary(m0)
```

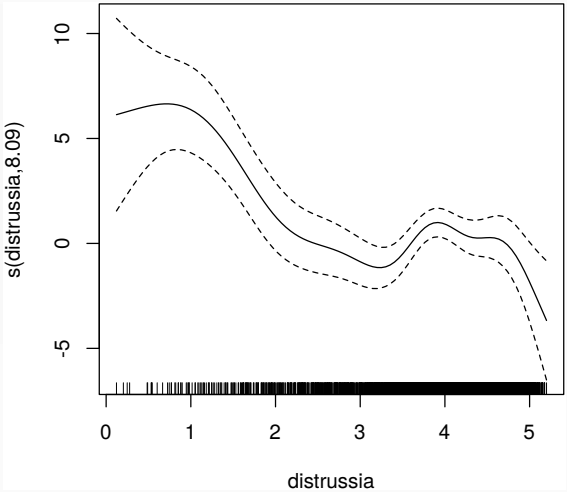
```
## OLS estimation, Dep. Var.: r14pres
## Observations: 3,567
## Fixed-effects: Raion: 66
## Standard-errors: Clustered (Raion)
##                           Estimate Std. Error t value  Pr(>|t|)
## qualityq                   6.43110    2.59240   2.481 1.571e-02 *
## distrussia                -2.01300    0.71904  -2.800 6.730e-03 **
## ukrainian                 -0.04428    0.01740  -2.545 1.331e-02 *
## r12                        0.43593    0.05824   7.484 2.400e-10 ***
## I(100 * turnout12)        -0.02799    0.02360  -1.186 2.400e-01
## I(type == "village")TRUE -0.53269    0.44201  -1.205 2.325e-01
## log(registered14pres)      0.76517    0.34874   2.194 3.181e-02 *
## log(roads + 1)            -0.46902    0.29152  -1.609 1.125e-01
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## RMSE: 5.1077     Adj. R2: 0.914868
##                 Within R2: 0.407785
```

Generalised Additive Models: Implementation

```
form1 <- formula("r14pres ~
    qualityq +
    s(distrussia) +
    factor(Raion) +
    ukrainian + r12 +
    I(100*turnout12) +
    I(type == 'village') +
    log(registered14pres) +
    log(roads + 1)")
m1 <- gam(form1, data = d)
```

- More examples

```
plot(m1)
```



GAMs for spatial smoothing (and covariate adjustment - WiP)

- Continuous - Gaussian Processes - `s(long, lat, bs = 'gp')`
- Discrete - Gaussian Markov Random Fields - `s(polygon, xt = neighbour_list, bs = 'mrf')`

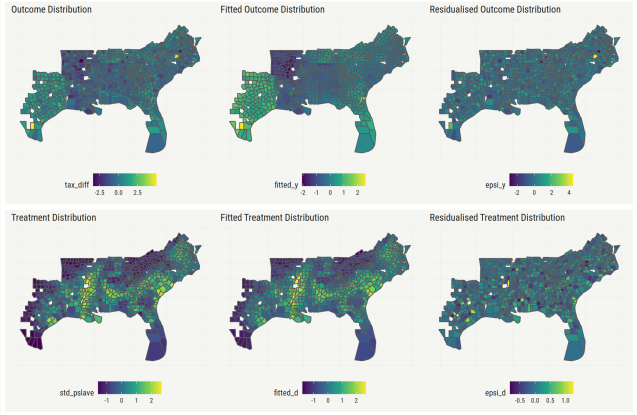


Figure 2: GMRFs for Suri and White (2020) treatment and outcome