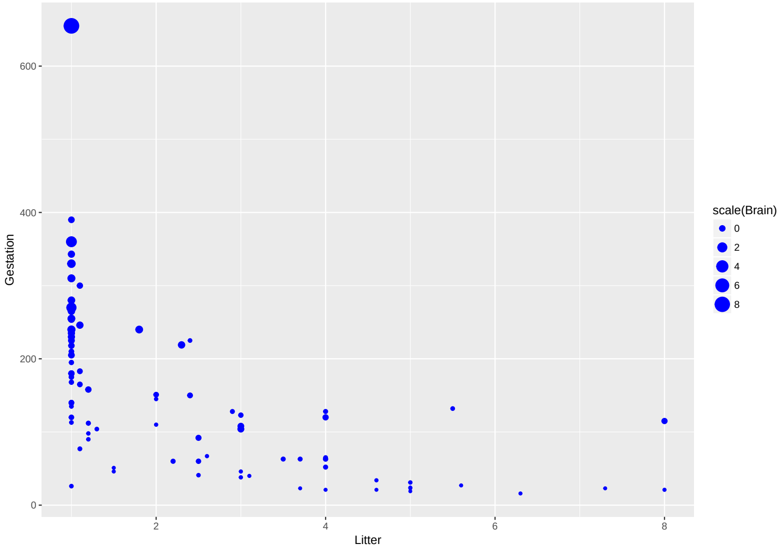
Tree - Based Method

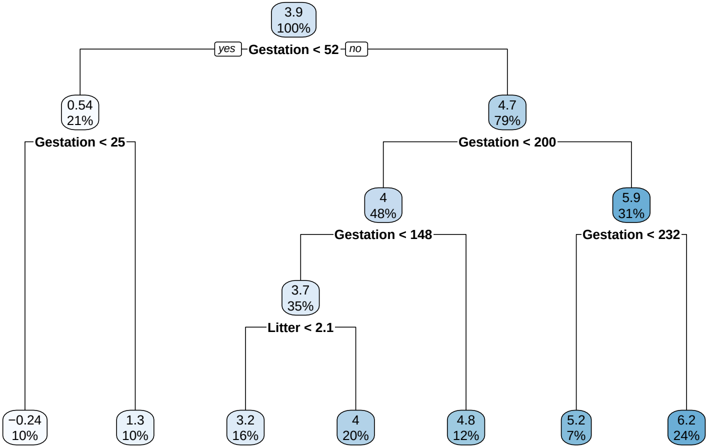
Enrique J. De La Hoz D.

Data Science - UTB

Example: Predict animal intelligence from Gestation Time and Litter Size



Decision Trees



Rules as a result of a Decision Tree

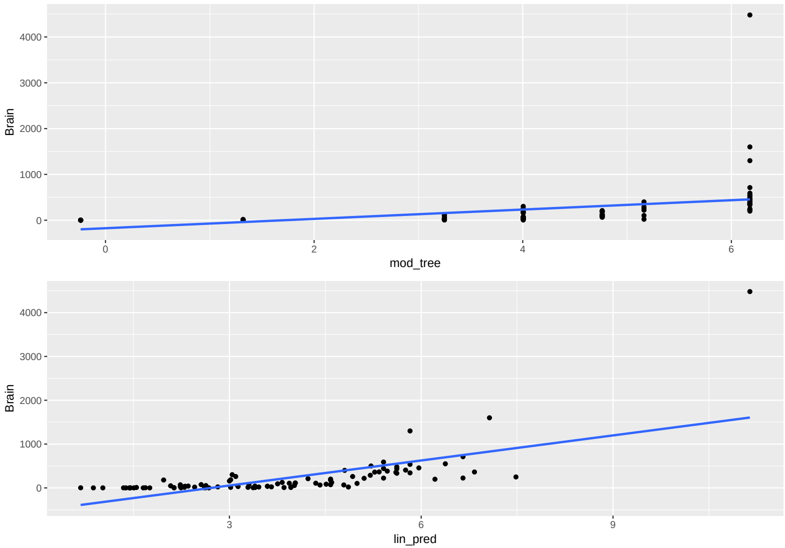
- Rules of the form:
  - if a AND b AND c THEN y
- Non-linear concepts
  - intervals
  - non-monotonic relationships
- non-additive interactions
  - AND: similar to multiplication

IF Gestation < 148 AND Litter $>= 2.1 \rightarrow$ Brain $= 4$

Linear Reg vs Decision Tree

```
   Model  RMSE
1 Linear 1.232
2   Tree 0.818
```

Visual Analysis



It's difficult for trees to express Linear Relationships

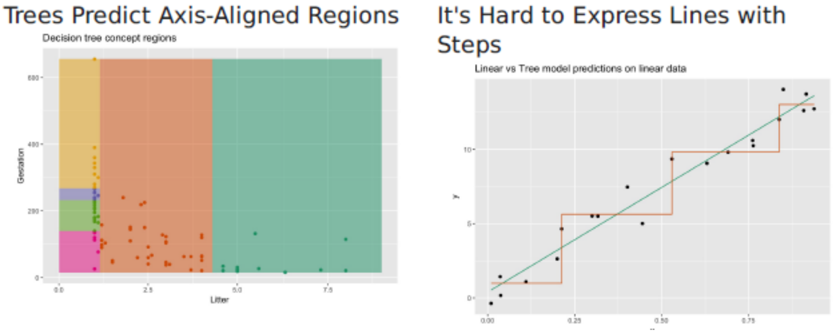


Figure 1:

- Each Color is a different predicted value

Issues with Trees

- Tree with too many splits (deep tree):
  - Too complex - danger of overfit
- Tree with too few splits (shallow tree):
  - Predictions too coarse-grained

Other Issues with Trees

- Tree with too many splits (deep tree):
  - Too complex - danger of overfit
- Tree with too few splits (shallow tree):
  - Predictions too coarse-grained

Ensembles of Trees

- Ensemble Model Fits Animal Intelligence Data Better than Single Tree

	Model	RMSE
1	Linear	1.232
2	Tree	0.818
3	Random Forest	1.171

Random Forests

- Multiple diverse decision trees averaged together
  - Reduces overfit
  - Increases model expressiveness
  - Finer grain predictions

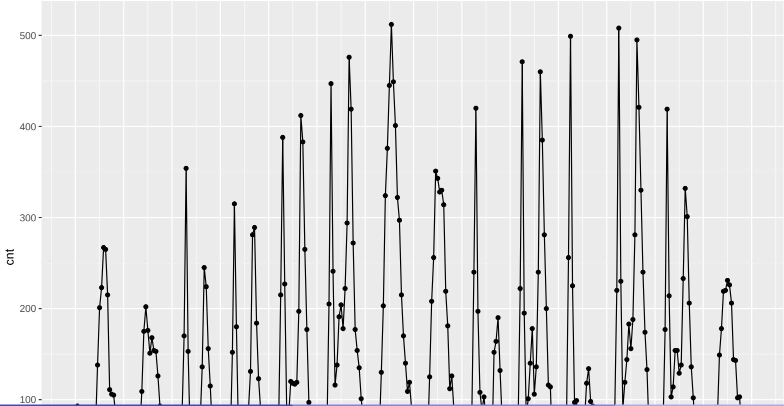
Building a Random Forest Model

1. Draw bootstrapped sample from training data
2. For each sample grow a tree
   - At each node, pick best variable to split on (from a random subset of all variables)
   - Continue until tree is grown
3. To score a datum, evaluate it with all the trees and average the results.

Example: Bike Rental Data

```
cnt ~ hr + holiday + workingday +
+ weathersit + temp + atemp + hum + windspeed
```

```
cnt ~ hr + holiday + workingday + +weathersit + temp + atemp +
    hum + windspeed
```



Random Forests with ranger()

```
fmla<- cnt ~ hr + holiday + workingday +
+ weathersit + temp + atemp + hum + windspeed

model_rf2 <- ranger(fmla, bikesJan, num.trees = 500,
respect.unordered.factors = "order")
```

- formula, data
- num.trees (default 500) - use at least 200
- mtry - number of variables to try at each node
  - default: square root of the total number of variables
- respect.unordered.factors - recommend set to “order”
  - “safe” hashing of categorical variables

Random Forest results

```
Ranger result

Call:
 ranger(fmla, bikesJan, num.trees = 500, respect.unordered.fac

Type:                             Regression
Number of trees:                  500
Sample size:                      741
Number of independent variables:  8
Mtry:                             2
Target node size:                 5
Variable importance mode:         none
Splitrule:                        variance
OOB prediction error (MSE):       3717.769
R squared (OOB):                  0.7409446
```

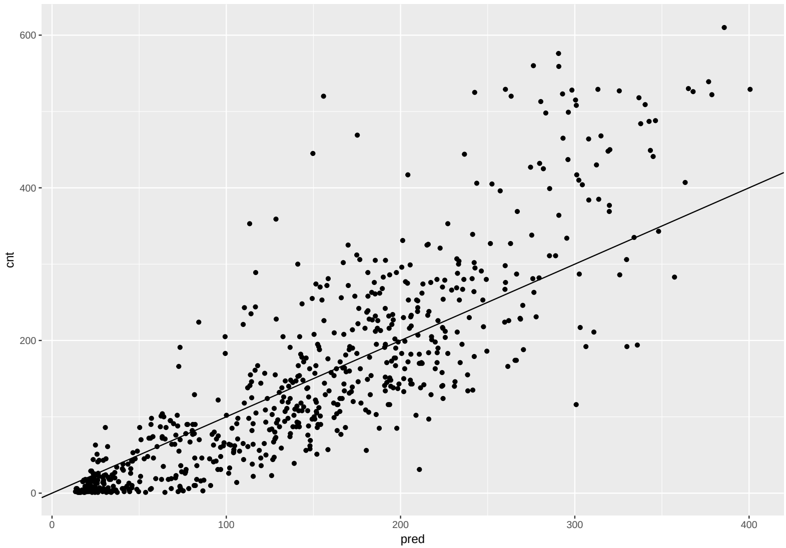
Predicting with a ranger() model

- predict() inputs:
  - model
  - data
- Predictions can be accessed in the element predictions.

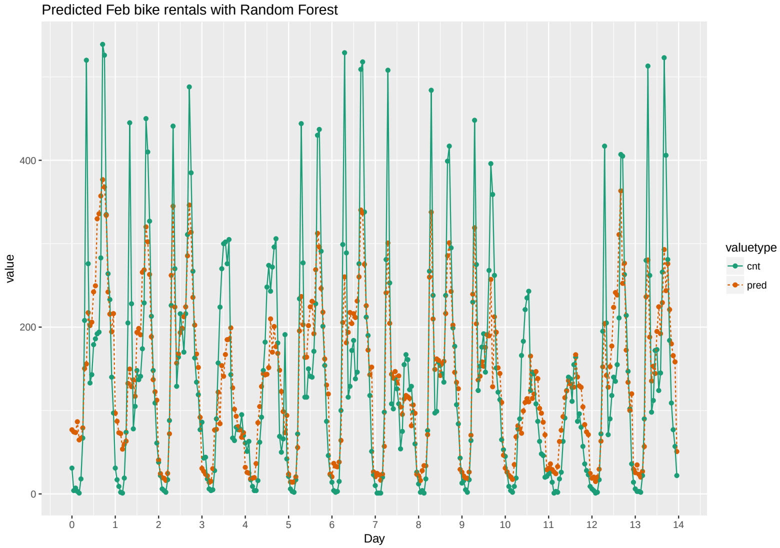
Evaluating the model

```
          Model    RMSE
1   Quasipoisson 118.964
2 Random Forest  74.292
```

Evaluating the model Visually Pred vs Actual



Predicted and Actual Hourly Bike Rentals



Categorical Variables in Random Forest

- ** One hot enconding variables **
- Most R functions manage the conversion for you (e.g glm())
  - model.matrix()
- xgboost() does not
  - Must convert categorical variables to numeric representation
- Conversion to indicators: one-hot encoding

One-hot-encoding and data cleaning with vtreat

Basic idea: - designTreatmentsZ() to design a treatment plan from the training data, then - prepare() to created "clean" data + All numerical + No missing values + Use prepare() with treatment plan for all future data

vtreat Example

** Training Data**

	x	u	y
1	one	5	2.669
2	three	12	1.501
3	one	56	0.199
4	two	28	1.278

Test Data

	x	u	y
1	one	44	0.486
2	two	24	1.368
3	three	66	2.036
4	two	22	1.640

Create the Treatment Plan

```r
library(vtreat)
varsx <- c("x", "u")
 treatplan <- designTreatmentsZ(dframe2,
                                varsx, verbose = FALSE)
```

- Inputs to designTreatmentsZ()
  - dframe: training data
  - varlist: list of input variable names
  - set verbose = FALSE to suppress progress messages

Get the new variables

- The scoreFrame describes the varia ble mapping and types

```
        varName origName  code
1        x_catP        x  catP
2       u_clean        u clean
3   x_lev_x.one        x   lev
4 x_lev_x.three        x   lev
5   x_lev_x.two        x   lev
```

- Get the names of the new lev and clean variables

```
[1] "u_clean"       "x_lev_x.one"   "x_lev_x.three" "x_lev_x.t
```

Prepare the Training Data for Modeling

- Inputs to prepare():
  - treatmentplan: treatment plan
  - dframe: data frame
  - varRestriction: list of variables to prepare (optional)
  - default: prepare all variables

Before and After Data Treatment

** Training Data**

```
================
    x   u    y
----------------
1  one  5  2.669
2 three 12 1.501
3  one  56 0.199
4  two  28 1.278
----------------
```

Treated training data

```
  u_clean x_lev_x.one x_lev_x
1       5           1
2      12           0
3      56           1
4      28           0
```

Prepare the Test Data Before Model Application

```
  u_clean x_lev_x.one x_lev_x.three x_lev_x.two
1      44           1             0           0
2      24           0             0           1
3      66           0             1           0
4      22           0             0           1
```

Robustness of vtreat treatment

- Previously unseen x level: four; Treated training data

```
================
    x   u    y
----------------
1  one   4  0.233
2  two   14 1.933
3 three 66 3.125
4 four  25 4.023
----------------

  u_clean x_lev_x.one x_lev_x.three x_lev_x.two
1       4           1             0           0
2      14           0             0           1
3      66           0             1           0
4      25           0             0           0
```

Gradient Boosting Machines

- boosting can be interpreted as an optimization algorithm on a suitable cost function
- optimize a cost function over function space by iteratively choosing a function (weak hypothesis) that points in the negative gradient direction
- This functional gradient view of boosting has led to the development of boosting algorithms in many areas of machine learning and statistics beyond regression and classification.

How Gradient Boosting Works

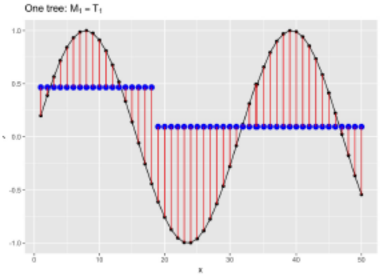


Figure 2:

1. Fit a shallow tree T_1 to the data: $M_1 = T_1$

How Gradient Boosting Works

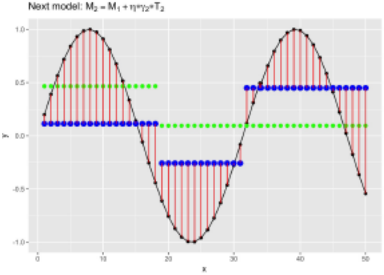


Figure 3:

1. Fit a shallow tree T_1 to the data: $M_1 = T_1$
2. Fit a tree T_2 to the residuals.
   - Find γ such that $M_2 = M_1 + \gamma T_2$ is the best fit to data

How Gradient Boosting Works

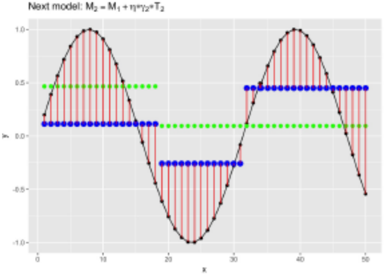


Figure 4:

- Regularization: learning rate $\eta \in (0,1)$

$$M_2 = M_1 + \eta\gamma T_2$$

- Larger η: faster learning

- Smaller η : less risk of overfit

How Gradient Boosting Works

- 100 iterations:

$$M_{100} = M_0 + \eta * \sum_{i=1}^{i=100} \gamma_i * T_i$$

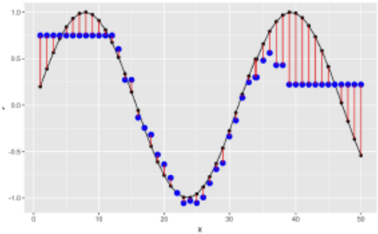


Figure 5:

1. Fit a shallow tree T_1 to the data: $M_1 = T_1$
2. Fit a tree T_2 to the residuals.
   - $M_2 = M_1 + \gamma T_2$
3. Repeat (2) until stopping condition met

Final Model

$$M = M_1 + \eta \sum \gamma_i T_i$$

Cross-validation to Guard Against Overfit

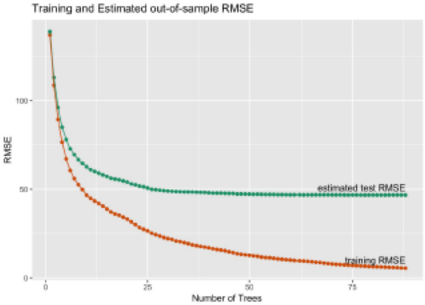


Figure 6:

- Training error keeps decreasing, but test error doesn't

Best Practice (with xgboost())

1. Run xgb.cv() with a large number of rounds (trees).
2. xgb.cv()$evaluation_log: records estimated RMSE for each round.
    - Find the number of trees that minimizes estimated RMSE: n_best
3. Run xgboost(), setting nrounds $= n_best$

Let's Continue with Bike Rental Model

- First, prepare the data

```
vars<- c('hr' , 'holiday' , 'workingday' ,
'weathersit' , 'temp' , 'atemp' , 'hum' , 'windspeed')

 treatplan <- designTreatmentsZ(bikesJan, vars,
                                 verbose = FALSE)
 newvars <- treatplan$scoreFrame %>%
 filter(code %in% c("clean", "lev")) %>%
 use_series(varName)
 bikesJan.treat <- prepare(treatplan,
              bikesJan, varRestriction = newvars)
```

- For xgboost():
  - Input data: as.matrix(bikesJan.treat)
  - Outcome: bikesJan$cnt

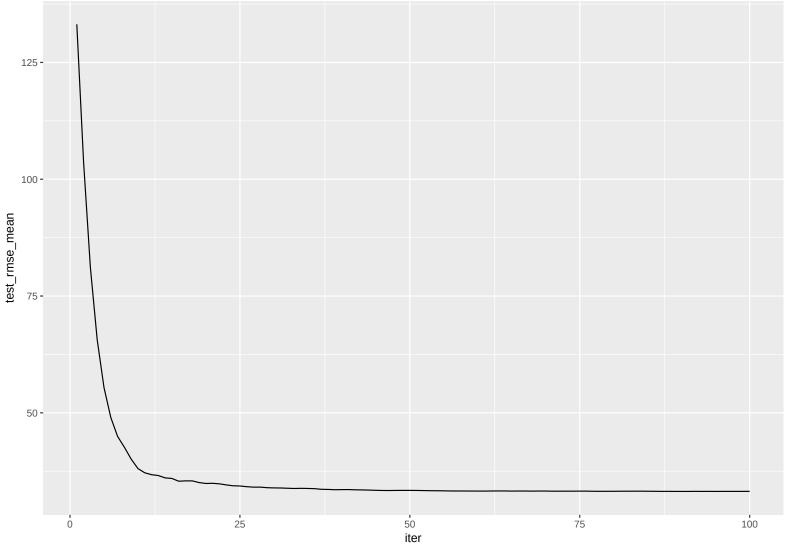
Training a model with xgboost() / xgb.cv()

```
library(xgboost)

cv <- xgb.cv(data = as.matrix(bikesJan.treat),
 label = bikesJan$cnt,
 objective = "reg:linear",
 nrounds = 100, nfold = 5, eta = 0.3,
 depth = 6, verbose = FALSE)
```

- Key inputs to xgb.cv() and xgboost()
  - data: input data as matrix ; label: outcome
  - objective: for regression - “reg:linear”
  - nrounds: maximum number of trees to fit
  - eta: learning rate
  - depth: maximum depth of individual trees
  - nfold (xgb.cv() only): number of folds for cross validation

Find the Right Number of Trees




```
[1] 91
```

Find the Right Number of Trees

```r
elog <- as.data.frame(cv$evaluation_log)

ggplot(elog, aes(x= iter, y = test_rmse_mean))+ geom_line()

(nrounds <- which.min(elog$test_rmse_mean))
```

Run xgboost() for final model

```
nrounds <- 56
model <- xgboost(data = as.matrix(bikesJan.treat),
                 label = bikesJan$cnt,
nrounds = nrounds,
objective = "reg:linear",
eta = 0.3,depth = 6, verbose = FALSE)
```

Predict with an xgboost() model

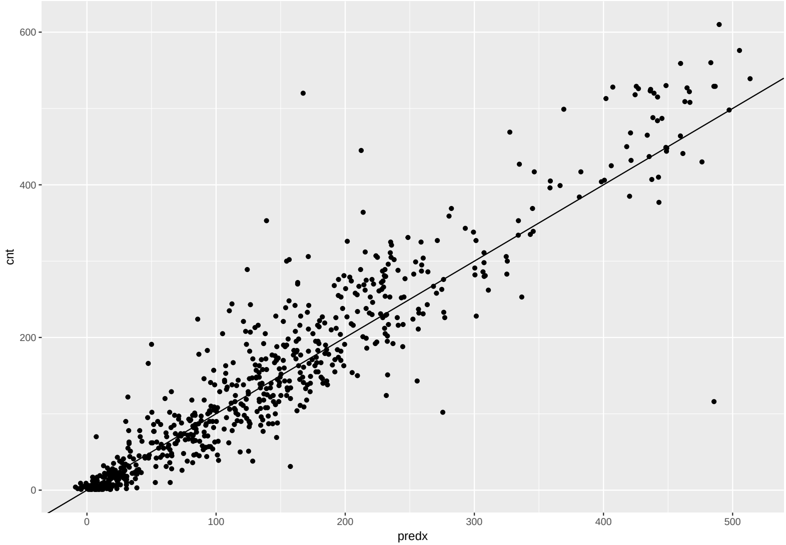
- Prepare February data, and predict

Model performances on Febrary Data

```
          Model    RMSE
1  Quasipoisson 118.964
2 Random Forest  74.292
3           XGB  47.611
```

- **XGBOOST outperforms the other models**

Evaluating the model Visually Pred vs Actual XGB



Visual Results