Overview

Regression Tree

1. How to build a tree
2. How to use tree to form prediction
3. Pros and cons of tree models

Regression Forest

1. **randomForest** based on bagging
2. **gbm** based on boosting

Case Study: Ames Housing Data

Overview

Regression Tree

1. How to build a tree
2. How to use tree to form prediction
3. Pros and cons of tree models

Regression Forest

1. **randomForest** based on bagging
2. **gbm** based on boosting

Case Study: Ames Housing Data

Tree Models

http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

Build a Tree

- **Top-down greedy** fashion: **recursively** divide data into small subgroups, and then fit a simple model (**constant**) for each subgroup.

- **Internal node**: variable/split_point
- **Leaf node**: a constant prediction

Make Predictions

Tree Models

http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

Pros

- Easy to interpret
- Automatic variable selection and interaction
- Invariant under any monotone transformations on predictors

Cons

- Unstable
- Weak performance

Overview

Regression Tree

1. How to build a tree
2. How to use tree to form prediction
3. Pros and cons of tree models

Regression Forest

1. **randomForest** based on bagging
2. **gbm** based on boosting

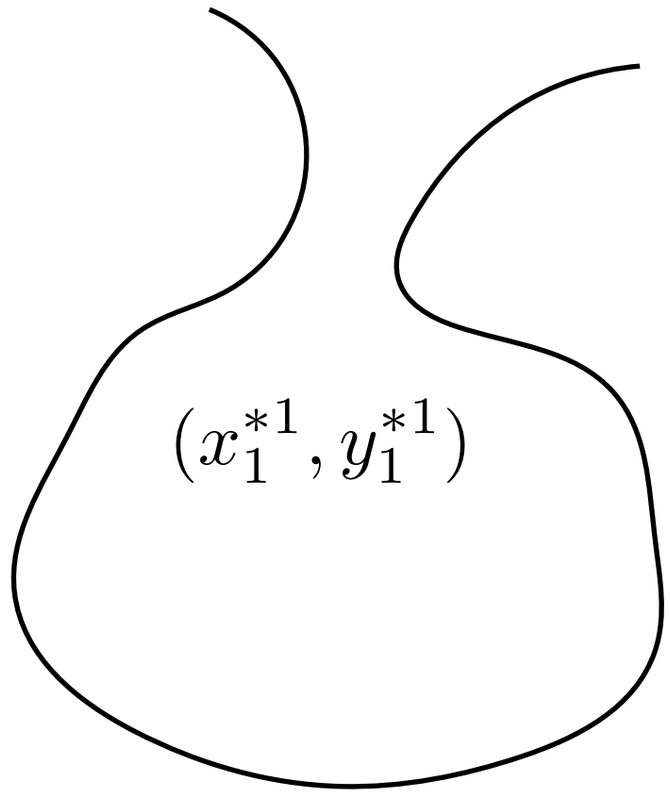
Case Study: Ames Housing Data

Bagging (Bootstrap Aggregation)

Training Data

$$\mathbf{Z} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

Bootstrap Sample 1



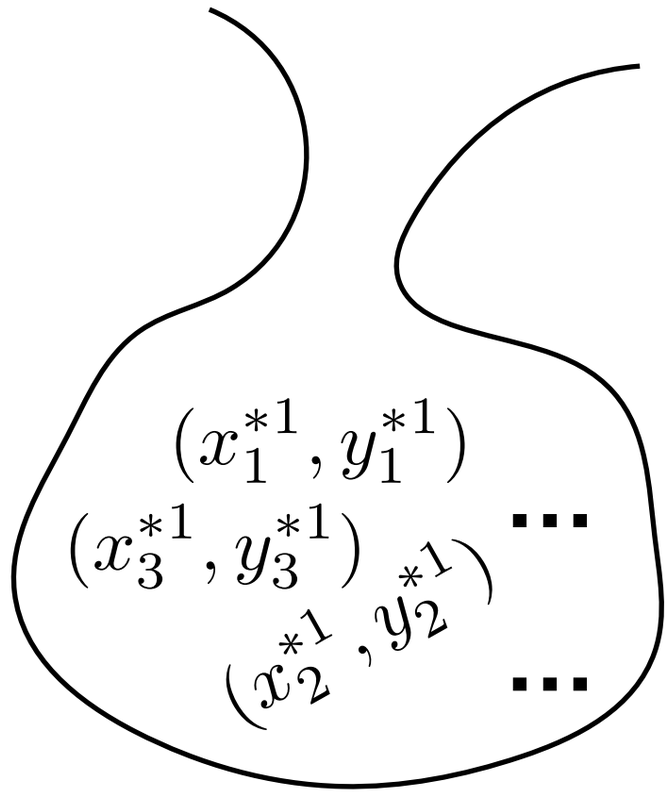
Sample with Replacement

Bagging (Bootstrap Aggregation)

Training Data

$$\mathbf{Z} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

Bootstrap Sample 1



Sample with Replacement

Bagging (Bootstrap Aggregation)

Training Data

$$\mathbf{Z} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$$

Bootstrap Sample 1

$$\mathbf{Z}^{*1} = \{x_1^{*1}, y_1^{*1}),\ (x_2^{*1}, y_2^{*1}),\quad \cdots,\quad (x_n^{*1}, y_n^{*1})\}$$

Bootstrap Sample 2

$$\mathbf{Z}^{*2} = \{x_1^{*2}, y_1^{*2}),\ (x_2^{*2}, y_2^{*2}),\quad \cdots,\quad (x_n^{*2}, y_n^{*2})\}$$

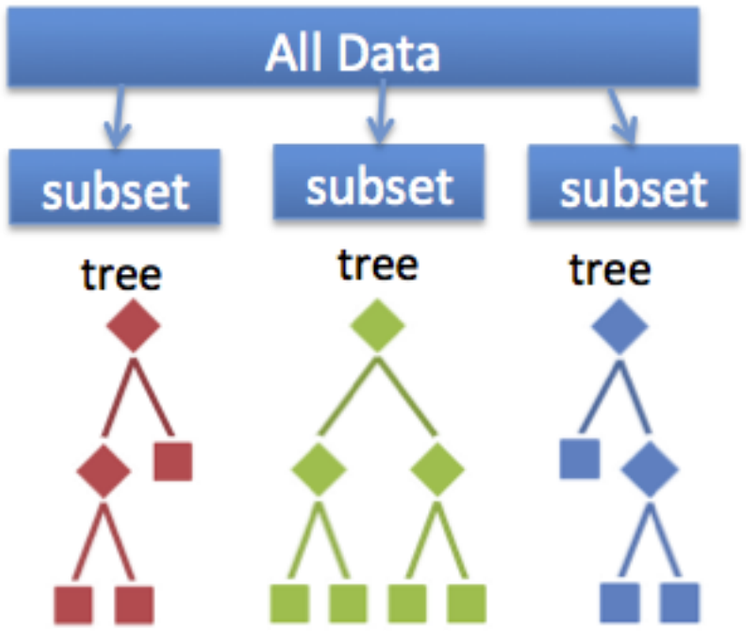
.....

Bootstrap Sample B

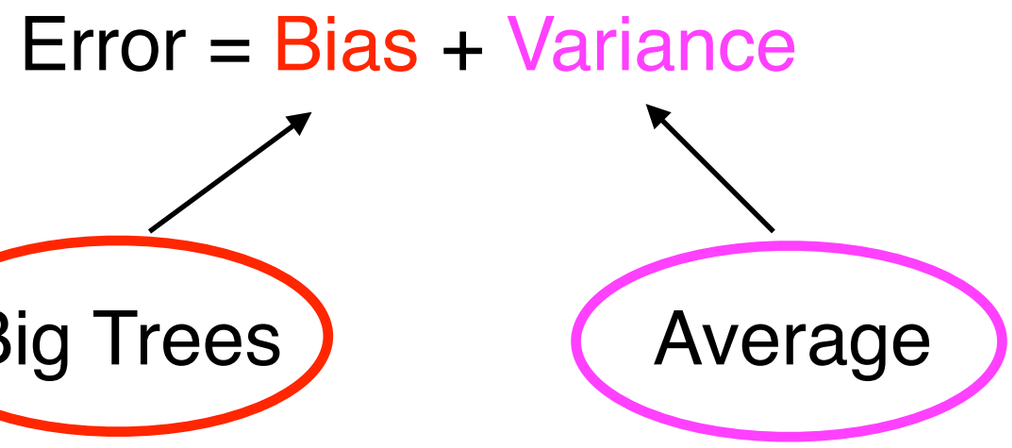
$$\mathbf{Z}^{*B} = \{x_1^{*B}, y_1^{*B}),\ (x_2^{*B}, y_2^{*B}),\quad \cdots,\quad (x_n^{*B}, y_n^{*B})\}$$

Bagging (Bootstrap Aggregation)

We can fit **B** trees, each based on one bootstrap sample, then when making predictions, we aggregate over the **B** trees.



$$\frac{1}{3}\Big(f_1(x) + f_2(x) + f_3(x)\Big)$$



Bagging (Bootstrap Aggregation)

We can fit **B** trees, each based on one bootstrap sample, then when making predictions, we aggregate over the **B** trees.

$$\frac{1}{B}\sum_{b=1}^{B} f_b(x)$$

Error = Bias + Variance

Big Trees

?

Average

Averaging won't reduce variance if outcomes are (positively) correlated.

Random Forest

a modification of Bagging; de-correlates trees by randomizing the choice for split-variables; minimal tuning; my favorite out-of-the-box learning algorithm.

```
Input: Training Data, B (# of trees)
for i = 1 to B do
|   Generate a bootstrap sample of the original data
|   Grow a big regression tree to the bootstrapped data
|   for each split do
|     |   Select m variables at random from all p variable
|     |   Pick the best variable/split_point among the m
|     |   Split the node into two
|     |   end
|   end
```

Overview

Regression Tree

1. How to build a tree
2. How to use tree to form prediction
3. Pros and cons of tree models

Regression Forest

1. **randomForest** based on bagging
2. **gbm** based on boosting

Case Study: Ames Housing Data

Boosting Trees

Boosting

Algorithms that can boost the performance of a set of weak regression (or classification) trees via combing them.

Boosting Trees

Boosting

Algorithms that can boost the performance of a set of weak regression (or classification) trees via combing them.

Forward stage-wise additive model

$$F(x) = f_1(x) + f_2(x) + \cdots + f_{T-1}(x) + f_T(x)$$

It is difficult to solve for all f_t's. Instead we solve them sequentially using a forward stage-wise greedy algorithm.

Forward Stage-wise Optimization

$$F(x) = f_1(x) + f_2(x) + \cdots + f_{T-1}(x) + f_T(x)$$

```
Input: Training Data, T (# of iterations)
Initialization: F(x) = 0 and current residuals r_i = y
for t = 1 to T do
|   Fit a regression tree f_t to the current residuals r_i
|   Add f_t to F:
|                 F = F + f_t
|   Update the current residual r_i <- r_i - f_t(x_i)
|   end
```

Forward Stage-wise Optimization

$$F(x) = f_1(x) + f_2(x) + \cdots + f_{T-1}(x) + f_T(x)$$

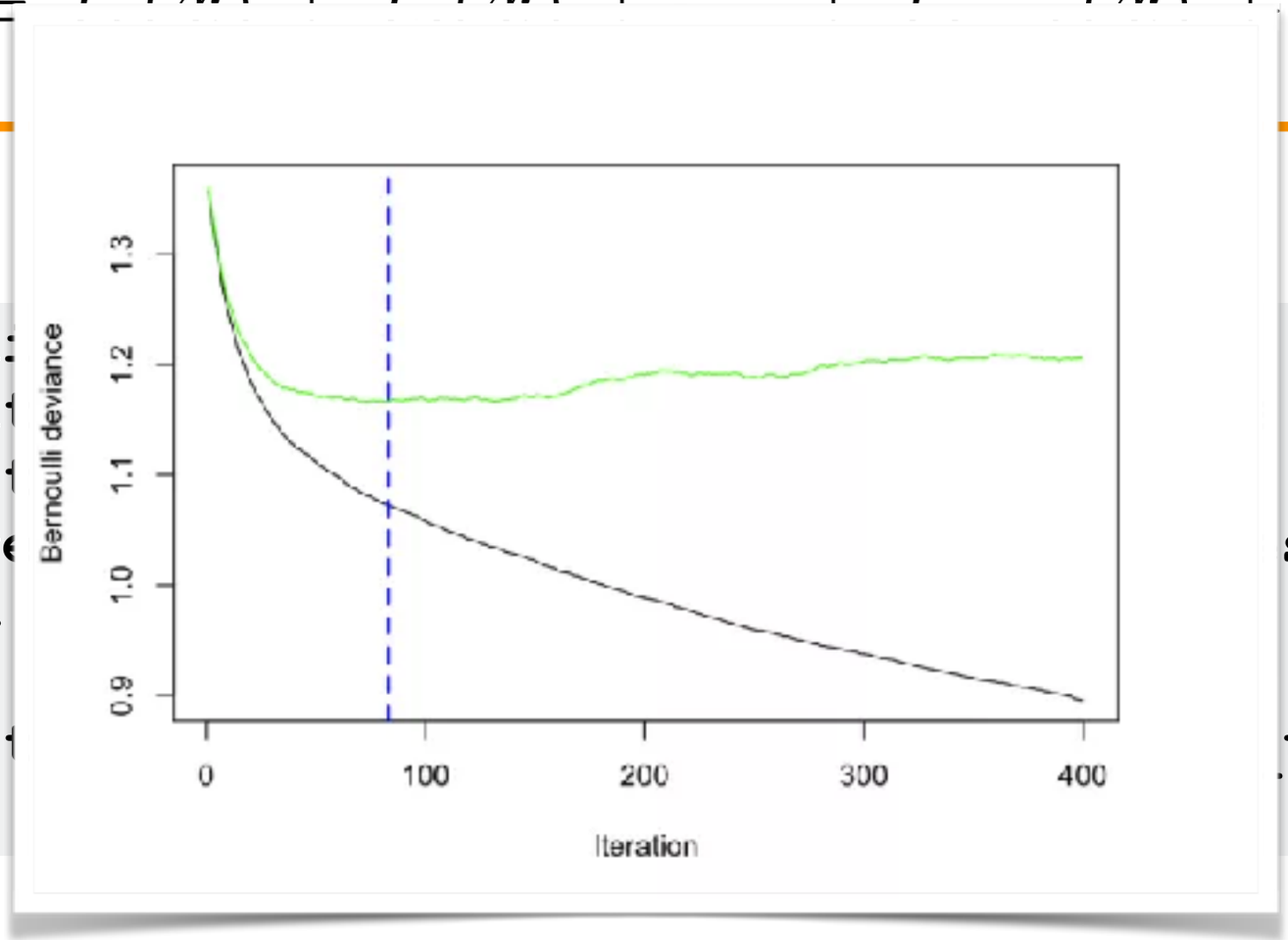
```
Input: Training Data, T (# of iterations)
Initialization: F(x) = 0 and current residuals r_i = y
for t = 1 to T do
|   Fit a regression tree f_t to the current residuals r_i
|   Add f_t to F:
|                F = F + f_t        f_t <— η f_t
|   Update the current residual r_i <- r_i - f_t(x_i)
|   end
```

Tuning parameters: **learning rate η**, number of trees T, complexity of f_t's (depth of trees), and subsampling rate.

Forward Stage-wise Optimization

$$F(x) = f_1(x) + f_2(x) + \dots + f_{T-1}(x) + f_T(x)$$

```
Input: Tra
Initializat                                          r_i = y
for t = 1 t
|   Fit a re                                        siduals r_i
|   Add f_t
|
|   Update f                                        t(x_i)
|   end
```



Tuning parameters: learning rate η, **number of trees T**, complexity of f_t's (depth of trees), and subsampling rate.

Forward Stage-wise Optimization

$$F(x) = f_1(x) + f_2(x) + \cdots + f_{T-1}(x) + f_T(x)$$

```
Input: Training Data, T (# of iterations)
Initialization: F(x) = 0 and current residuals r_i = y
for t = 1 to T do
|   Fit a regression tree f_t to the current residuals r_i
|   Add f_t to F:
|                 F = F + f_t
|   Update the current residual r_i <- r_i - f_t(x_i)
|   end
```

Tuning parameters: learning rate η, number of trees T, complexity of f_t's (**depth of trees**), and subsampling rate.

Relatively small trees (weak regression trees)

Forward Stage-wise Optimization

$$F(x) = f_1(x) + f_2(x) + \cdots + f_{T-1}(x) + f_T(x)$$

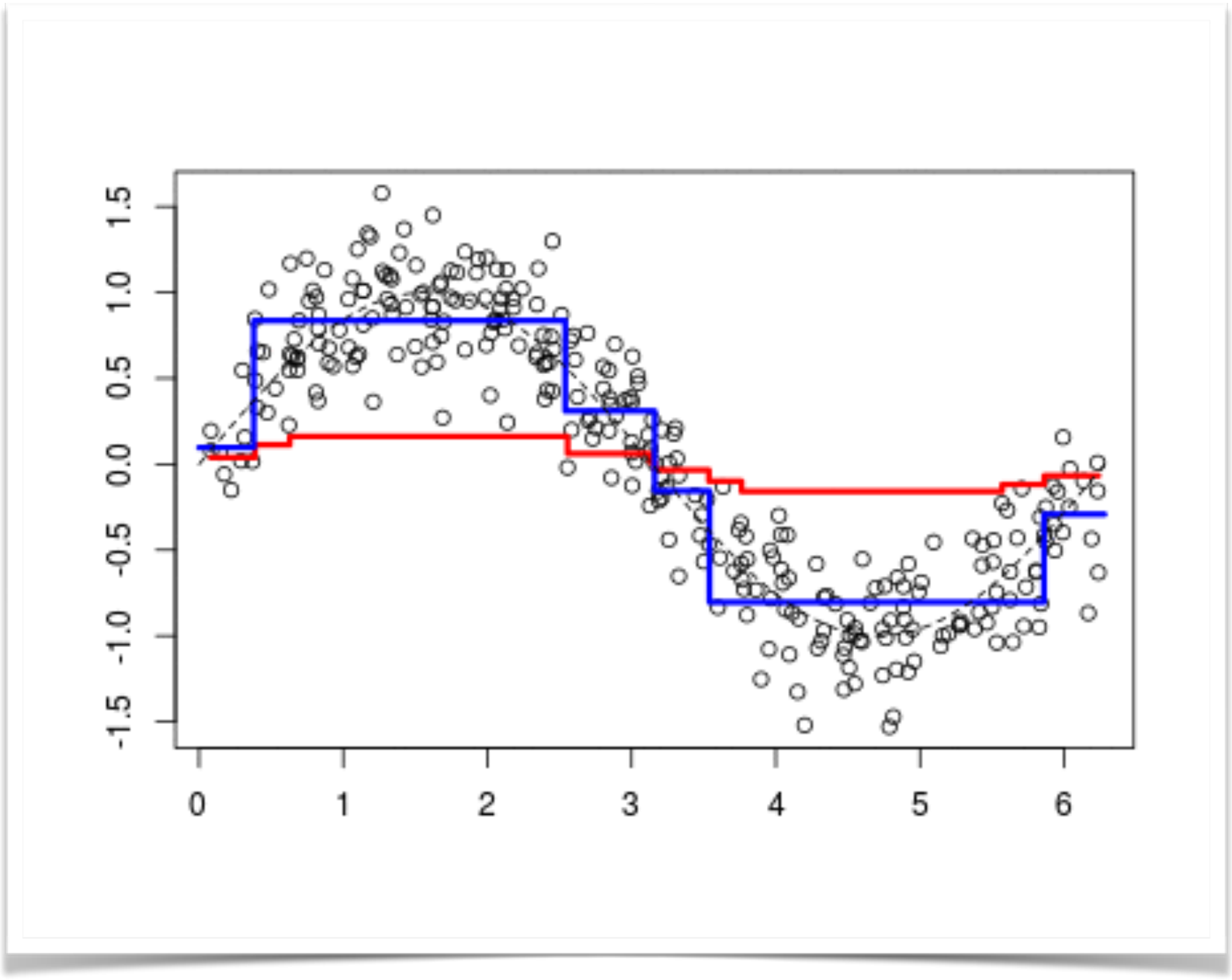
```
Input: Training Data, T (# of iterations)
Initialization: F(x) = 0 and current residuals r_i = y
for t = 1 to T do
|   Fit a regression tree f_t to the current residuals r_i
|   Add f_t t
|
|   Update th
|   end
```

Sampling without replacement, i.e., less # of samples will be used when fitting a single tree (computation advantage).

Tuning parameters: learning rate η, number of trees T, complexity of f_t's (depth of trees), and **subsampling rate**.

Demo: Boosting Tree for Curve Fitting

http://uc-r.github.io/public/images/analytics/gbm/boosted_stumps.gif



Comparison

	randomForest	BoostingTree	SingleTree
Accuracy	★★	★★★	★
Interpretation	★	★	★★★
Easy-to-Use	★★★	★	★★★
Computation	★	★★	★
Tree Size	Large	Small	
Parallel	Yes	No	

Overview

Regression Tree

1. How to build a tree
2. How to use tree to form prediction
3. Pros and cons of tree models

Regression Forest

1. **randomForest** based on bagging
2. **gbm** based on boosting

Case Study: Ames Housing Data

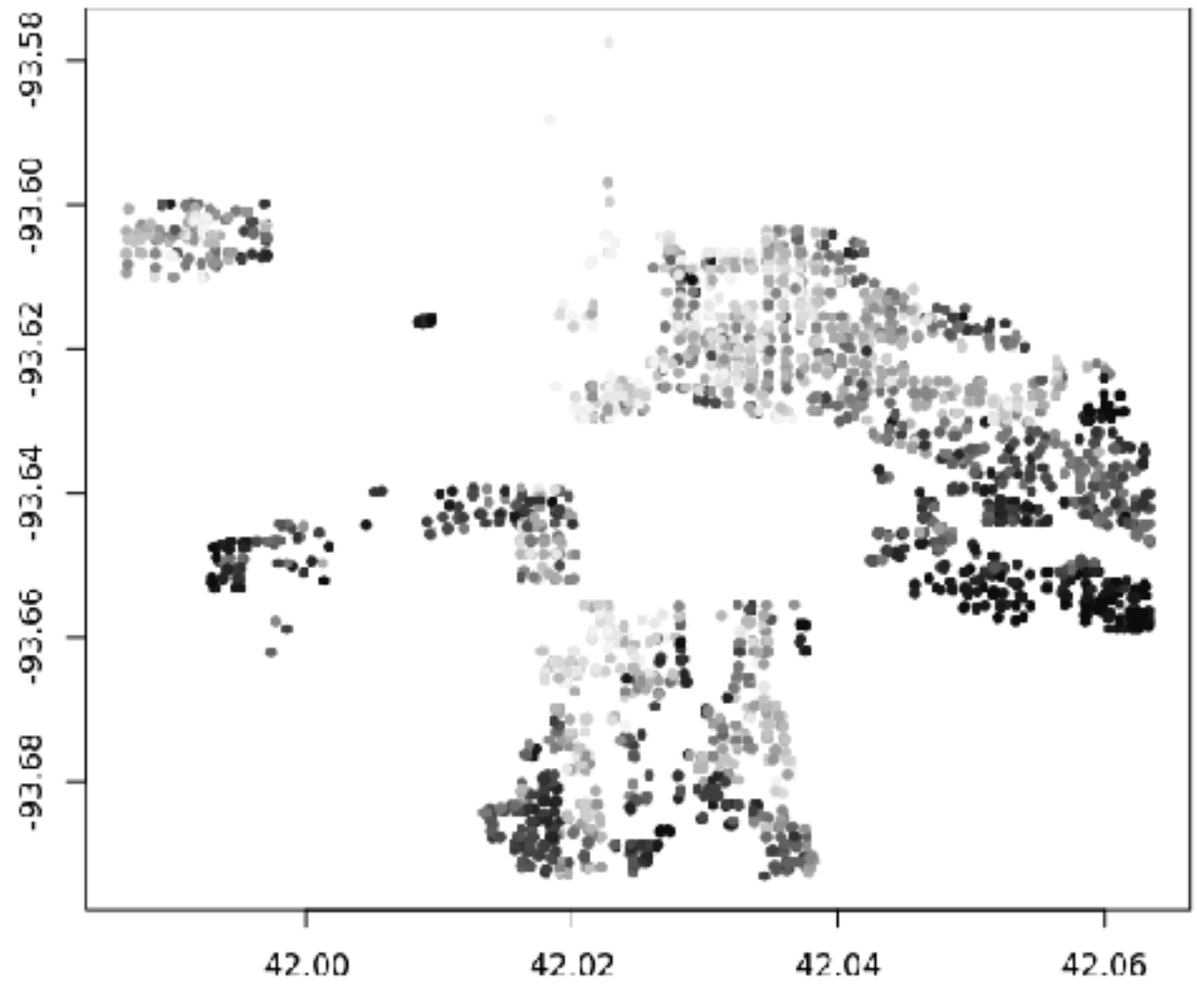
Ames Housing Data

https://www.kaggle.com/c/house-prices-advanced-regression-techniques

Full data from R package [AmesHousing]

- n = 2930 (houses)
- p = 83 (features, including **PID** and **Sale_Price**)

```
[1] "PID"                "MS_SubClass"        "MS_Zoning"
 [4] "Lot_Frontage"       "Lot_Area"           "Street"
 [7] "Alley"              "Lot_Shape"          "Land_Contour"
[10] "Utilities"          "Lot_Config"         "Land_Slope"
[13] "Neighborhood"       "Condition_1"        "Condition_2"
[16] "Bldg_Type"          "House_Style"        "Overall_Qual"
[19] "Overall_Cond"       "Year_Built"         "Year_Remod_Add"
[22] "Roof_Style"         "Roof_Matl"          "Exterior_1st"
[25] "Exterior_2nd"       "Mas_Vnr_Type"       "Mas_Vnr_Area"
[28] "Exter_Qual"         "Exter_Cond"         "Foundation"
[31] "Bsmt_Qual"          "Bsmt_Cond"          "Bsmt_Exposure"
[34] "BsmtFin_Type_1"     "BsmtFin_SF_1"       "BsmtFin_Type_2"
[37] "BsmtFin_SF_2"       "Bsmt_Unf_SF"        "Total_Bsmt_SF"
[40] "Heating"            "Heating_QC"         "Central_Air"
[43] "Electrical"         "First_Flr_SF"       "Second_Flr_SF"
[46] "Low_Qual_Fin_SF"    "Gr_Liv_Area"        "Bsmt_Full_Bath"
[49] "Bsmt_Half_Bath"     "Full_Bath"          "Half_Bath"
[52] "Bedroom_AbvGr"      "Kitchen_AbvGr"      "Kitchen_Qual"
[55] "TotRms_AbvGrd"      "Functional"         "Fireplaces"
[58] "Fireplace_Qu"       "Garage_Type"        "Garage_Yr_Blt"
[61] "Garage_Finish"      "Garage_Cars"        "Garage_Area"
[64] "Garage_Qual"        "Garage_Cond"        "Paved_Drive"
[67] "Wood_Deck_SF"       "Open_Porch_SF"      "Enclosed_Porch"
[70] "Three_season_porch" "Screen_Porch"       "Pool_Area"
[73] "Pool_QC"            "Fence"              "Misc_Feature"
[76] "Misc_Val"           "Mo_Sold"            "Year_Sold"
[79] "Sale_Type"          "Sale_Condition"     "Longitude"
[82] "Latitude"           "Sale_Price"
```



```
> sort(abs(y.pred - y.test), decreasing = TRUE)[1:10]
```

randomForest

```
> library("randomForest")
> set.seed(123)
> m1 = randomForest(Sale_Price ~ ., importance=TRUE,
                    tmpdata[-test.id, ], ntree=500)
> y.pred = predict(m1, newdata = tmpdata[test.id, ])
```

```
> y.test = tmpdata$Sale_Price[test.id]
> sqrt(mean((y.pred - y.test)^2))
```

Minimal pre-processing:
— no transformation;
— fill in some missing values;
— no one-hot coding for cat.

GBM

```
> library("gbm")
> gbm.fit <- gbm(
    formula = Sale_Price ~ .,
    distribution = "gaussian",
    data = tmpdata[-test.id, ],
    n.trees = 5000,
    interaction.depth = 2,
    shrinkage = 0.01,
    cv.folds = 5,
    bag.fraction = 0.75,
    verbose = FALSE
)
```

```
> y.pred = predict(gbm.fit, n.trees = 4971, testdata)
> sqrt(mean((y.test - y.pred)^2))
```

```
> sort(abs(y.pred - y.test), decreasing = TRUE)[1:10]
```

randomForest

```
> library("randomForest")
> set.seed(123)
> m1 = randomForest(Sale_Price ~ ., importance=TRUE,
                    tmpdata[-test.id, ], ntree=500)
> y.pred = predict(m1, newdata = tmpdata[test.id, ])
```

```
> y.test = tmpdata$Sale_Price[test.id]
> sqrt(mean((y.pred - y.test)^2))
```

Project 1 for F18 Stat 542

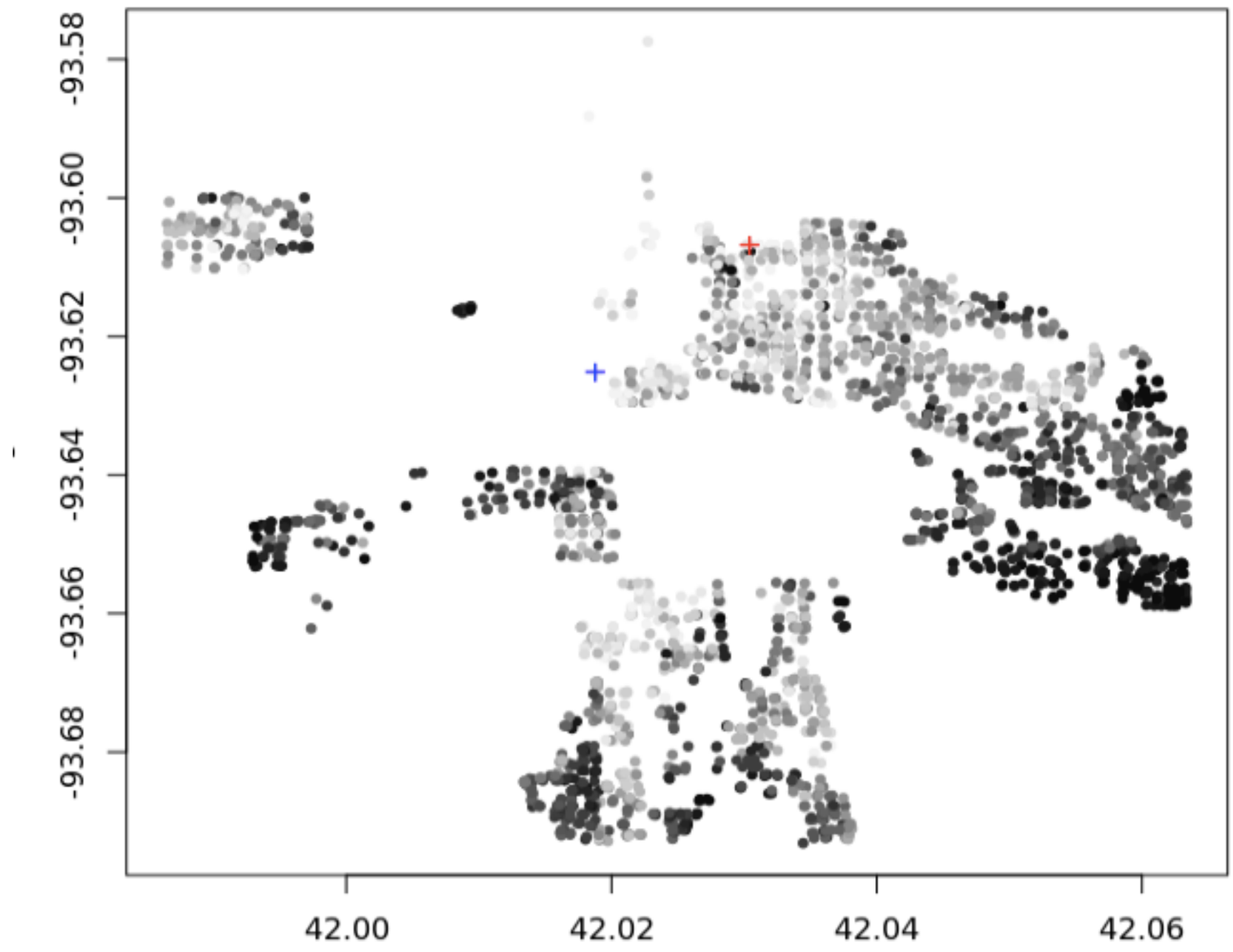
10 Splits of Training/Test
Target perf < 0.132

Very difficult for some splits, e.g., Split 3. Why?

GBM

```
> library("gbm")
> gbm.fit <- gbm(
    formula = Sale_Price ~ .,
    distribution = "gaussian",
    data = tmpdata[-test.id, ],
    n.trees = 5000,
    interaction.depth = 2,
    shrinkage = 0.01,
    cv.folds = 5,
    bag.fraction = 0.75,
    verbose = FALSE
)
```

```
> y.pred = predict(gbm.fit, n.trees = 4971, testdata)
> sqrt(mean((y.test - y.pred)^2))
```




```
 94 | 68
 96 |
 98 |
100 |
102 |
104 | 66748
106 | 0915
108 | 2256722236889
110 | 000233445777777778801123333678 89
112 | 00333333445556667888888899999999999900000111111112222233334444555 5+19
114 | 000000001111111112222223334444445555555666666667777888888899999999 9+98
116 | 0000000000001111111111111111111111111222222223333333333333333444444 4+406
118 | 000000000000000000111111111111111111111111111111111111111111111111 1+585
120 | 000000000000000000000001111111111111111111111111111111122222222222 2+474
122 | 000000000000011111111111111111111111111111111222222222222222223333 3+314
124 | 000000000111111111122222222222233333333333333333333333344444444455 5+173
126 | 000011111111111112222233333344444445555566666666666667777777777778888 8+56
128 | 001222223333333333444445555666666778889999900011111222233444445566 6
130 | 0112233445566778912239
132 | 012338892235
134 | 23
```