Tree-based Models for Regression

- Regression trees
- Regression forests
  - `randomForest` based on bagging
  - `gbm` based on boosting

Tree-based Models for Regression

- Input vector $X = (X_1, X_2, \dots, X_p) \in \mathcal{X}$
- Response variable $Y \in \mathbb{R}$
- Trees are constructed by recursively splitting regions of $\mathcal{X}$ into two sub-regions, beginning with the whole space $\mathcal{X}$.

  For simplicity, focus on recursive binary partitions.
- `R` page: check the fitted regression tree on `BostonHousingData` based on two features `lon` and `lat`.

- Notation: node (t), child node (t_L, t_R), split (var j, value s), leaf/terminal node.
- Every leaf node (i.e. a rectangle region R_m in $\mathcal{X}$) is assigned with a constant for regression tree

$$\hat{f}(X) = \sum_m c_m I\{X \in R_m\}.$$

Advantages of Trees

- Easy to interpret
- Variable selection and interactions between variables are handled automatically
- Invariant under any monotone transformation of predictors

How to Build a Tree?

Three elements:

1. Where to split?

2. When to stop?

3. How to predict at each leaf node?

Prediction at Leaf Nodes

Each leaf node (corresponds to region R_m) contains some samples.

Assign the prediction for a leaf node to be the average (of the response variable Y).

$$\hat{f}(X) = \sum_m c_m I\{X \in R_m\}.$$

$$\min_{c_m} \sum_{i=1,\ x_i \in R_m}^{n} (y_i - c_m)^2,$$

$$\Longrightarrow c_m = \text{ average of } y_i\text{'s whose } x_i \in R_m$$

Where to Split?

- A split is denoted by (j, s): split the data into two parts based on whether "var j < value s".
- For each split, define a split criterion $\Phi(j, s)$
  - deduction of RSS for regression
- Trees are built in a top-down greedy fashion. Start with the root: try all possible variables $j = 1 : p$ and all possible split values[a], and pick the best split, i.e., the split having the best Φ value. Now, data are divided into the left node and right node. Repeat this procedure in each node.

[a]For each variable j, sort the n values (from n samples), and choose s to be a middle point of two adjacent values. So at most $(n-1)$ possible values for s.

Goodness of Split $\Phi(j, s)$

For Regression tree, we look at the deduction of RSS if we split samples at node t into t_R and t_L:

$$\Phi(j, s) = \text{RSS}(t) - \Big[\text{RSS}(t_R) + \text{RSS}(t_L)\Big],$$

where

$$\begin{aligned} \text{RSS}(t) &= \sum_{x_i \in t} (y_i - c_t)^2, \\ c_t &= \text{AVE}\{y_i : x_i \in t\}. \end{aligned}$$

Note that $\Phi(j, s)$ is always positive if we split the data into two groups (even randomly), unless the mean of the left node and the one of right node are the same.

Issues: Split Categorical Predictors

- For a categorical predictor with m levels, there are $2^{m-1} - 1$ possible partitions of the m labels into two groups.
- However, for regression with square error, the computation simplifies: order the m levels by their mean values of Y, and then split the categorical variable as if it were an ordered predictor — there are only $(m - 1)$ potential splits.

Issues: Missing Predictor Values

- Discard any observation with missing values $\longrightarrow$ serious depletion of the training set.
- Splitting criteria are evaluated on non-missing observations.
- Once a split (j, s) is determined, what to do with observations missing X_j?

- Find surrogate variables that can predict the binary outcome "$X_j < s$" and "$X_j \geq s$" using a one-split tree.
- Rank those surrogate variables along with the blind rule "go with majority".
- Any observation that is missing X_j is then classified with the first surrogate variable, or if missing that, the second surrogate variable (or the blind rule) is used, and etc.

When to Stop?

- A simple one : stop splitting at a node if the gain from any split is less than some pre-specified threshold.
- BUT, this is short-sighted.
- Another strategy: grow a large tree and then prune it (i.e., cut some branches).

Preliminaries for Pruning

First, grow a vary large tree $T_{\max}$

1. until all terminal nodes are nearly pure;

2. or when the number of data in each terminal node is less than certain threshold;

3. or when the tree reaches certain size.

As long as the tree is sufficiently large, the size of the initial tree is not critical.

Notation : *subtree* $T' \prec T$, *branch* T_t.

Minimum Complexity-cost Pruning

For any subtree $T \prec T_{\max}$, define the Complexity-cost

$$R_\alpha(T) = R(T) + \alpha|T|, \tag{1}$$

- $R(T)$: RSS for regression tree T
- $|T|$: tree size, i.e., the number of leaf nodes
- $\alpha > 0$: cost (penalty) of adding a split

Questions: *i)* How to minimize (1) for a given α? *ii)* How to choose α?

Pick the best subtree that minimizes the cost

$$T(\alpha) = \text{argmin}_{T \preceq T_{\max}} R_\alpha(T) = \text{argmin}_{T \preceq T_{\max}} \Big[R(T) + \alpha |T| \Big]$$

$T(\alpha)$ may not be unique.

Define the optimal subtree $T^*(\alpha)$ to be the smallest one among $T(\alpha)$'s

(1) $R_\alpha(T^*(\alpha)) = \min_{T \preceq T_{\max}} R_\alpha(T)$.

(2) $T^*(\alpha) \preceq$ any $T(\alpha)$.

$T^*(\alpha)$ is unique.

$$R_\alpha(T) = R(T) + \alpha|T|$$

Some Facts

- For a pair of leaf nodes (t_L, t_R), there exists α^*, such that

  1. for any $\alpha \geq \alpha^*$, we would like to collapse them to just node t;
  2. for any $\alpha < \alpha^*$, keep the two leaf nodes.

  That is, α^* is the maximal price we would like to pay to keep that split.

  Next we extend this calculation to compute the maximal price we would like to pay to keep a branch T_t.

- For any non-leaf node t, do the following calculation to find out the maximal price we'd like to pay for keeping the whole branch T_t.

  Focus only on samples at node t.

  - Cost for keeping branch T_t: $R_\alpha(T_t) = R(T_t) + \alpha|T_t|$
  - Cost for cutting branch T_t: $R_\alpha(\{t\}) = R(\{t\}) + \alpha$
  - Calculate

$$\alpha^* = \frac{R(\{t\}) - R(T_t)}{|T_t| - 1}.$$

  That is, if the given $\alpha > \alpha^*$, then it is too expensive to keep this branch and we would like to cut the whole branch and make t a leaf node.

Weakest-Link Pruning

The weakest-link pruning algorithm.

- Start with $T_0 = T_{\text{max}}$ and $\alpha_0 = 0$.
- For any non-leaf node t, denote the maximal price we'd like to pay to keep T_t by $\alpha(t)$.
- $\alpha_1 = \min_t \alpha(t)$. The corresponding (non-terminal node) t_1 is called the weakest link. Cut the branch at t_1.
- Next update the maximal price for each non-leaf node (we only need to recompute the maximal price for nodes that are parents/grandparents of t_1). Find α_2 and cut the branch at the 2nd weakest link. Keep doing this until we get to the root.

The steps above generate a Solution Path:

$$T_{\mathsf{max}} = T_0 \succ T^*(\alpha_1) \succ T^*(\alpha_2) \succ \cdots \succ \{\text{root node}\}$$

$$0 = \alpha_0 < \alpha_1 < \alpha_2 < \cdots$$

All possible values of α are grouped into $(m+1)$ intervals:

$$\begin{aligned} I_0 &= [0, \alpha_1) \\ I_1 &= [\alpha_1, \alpha_2) \\ &\vdots \\ I_m &= [\alpha_m, \infty) \end{aligned}$$

where all $\alpha \in I_i$ share the same optimal subtree $T^*(\alpha_i)$.

Cross-validation

How to Choose α? K-fold Cross-validation (`rpart`):

1. Fit a big tree $T_{\max}$ and compute $I_0, I_1, \ldots, I_m$

$$\begin{aligned} \text{Set } \beta_0 &= 0 \\ \beta_1 &= \sqrt{\alpha_1 \alpha_2} \\ &\vdots \\ \beta_{m-1} &= \sqrt{\alpha_{m-1}\alpha_m} \\ \beta_m &= \infty \end{aligned}$$

where each β_j is a 'typical value' for its interval I_j.

2. Divide data into K groups and repeat $k = 1, \ldots, K$:
    - Fit a full model on the data set except the k-th group and determine the optimal subtrees:

$$T_0 \succ T^*(\beta_1) \succ \cdots \succ T^*(\beta_m) \succ \{\text{root node}\}$$

    - Compute the prediction error on the k-th group for each tree models.
3. Produce the CV plot over different α values and pick the optimal α_{min} or α_{1se}.

Tree-based Models for Regression

- Regression trees
- Regression forests
  - `randomForest` based on bagging
  - `gbm` based on boosting

Bagging (Bootstrap Aggregation)

- Training data: $\mathbf{Z} = \{(x_i, y_i)_{i=1}^n\}$
- Bootstrap samples[a]: $\mathbf{Z}^{*b} = \{(x_i^{*b}, y_i^{*b})_{i=1}^n\}$, where $b = 1 : B$

$$
\begin{array}{ccl}
\mathbf{Z}^{*1} & : & (x_1^{*1}, y_1^{*1}),\ (x_2^{*1}, y_2^{*1}),\quad \cdots,\quad (x_n^{*1}, y_n^{*1}) \\
\mathbf{Z}^{*2} & : & (x_1^{*2}, y_1^{*2}),\ (x_2^{*2}, y_2^{*2}),\quad \cdots,\quad (x_n^{*2}, y_n^{*2}) \\
\vdots & & \\
\mathbf{Z}^{*B} & : & (x_1^{*B}, y_1^{*B}),\ (x_2^{*B}, y_2^{*B}),\quad \cdots,\quad (x_n^{*B}, y_n^{*B})
\end{array}
$$

[a] sample with replacement from $\mathbf{Z}$.

Bagging (Bootstrap Aggregation)

- Training data: $\mathbf{Z} = \{(x_i, y_i)_{i=1}^n\}$
- Bootstrap samples[a]: $\mathbf{Z}^{*b} = \{(x_i^{*b}, y_i^{*b})_{i=1}^n\}$, where $b = 1 : B$
- $\hat{f}^{*b}$: classification/regression function trained by $\mathbf{Z}^{*b}$
- The bagging estimate is defined to be

$$\hat{f}_{\mathsf{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}.$$

- Advantage: reduce variance. So works well for high-variance, low-bias procedures, such as trees.

[a] sample with replacement from $\mathbf{Z}$.

Random Forest

1. For $b = 1 : B$:
   (a) Draw a bs sample $\mathbf{Z}^{*b}$ from the training data.
   (b) Grow a BIG tree T_b (with some restriction).
2. Output the forest $\{T_b\}_{b=1}^{B}$.

To make a prediction at a new point x

Regression: $\frac{1}{B}\sum T_b(x)$.

Restriction when growing a tree in the forest:

- At each split, randomly select m variables from the p variables, and then pick the best split among them.
- The recommended value for m is $\sqrt{p}$ for classification and $p/3$ for regression.
- Purpose: reduce the correlation between trees in the forest.

Out-of-Bag (OOB) Samples

- OOB samples: sample points which are not included in $\mathbf{Z}^{*b}$, i.e., they are not used in building the tree T_b
- The OOB samples can be used to get a test error for T_b.
- The prediction and error rate returned by randomForest are calculated based on OOB. The error is usually close to a CV error.

Variable Importance

- Measure the importance of a variable by the improvement of RSS contributed by this variable.
- At each split, attribute the improvement of RSS to the corresponding splitting variable.
- For each variable, accumulate its improvement of RSS across the tree and then averaged over all the tress in the forest.

- Another measure is computed from permuting OOB samples: For each tree T_b in the forest, calculate the prediction error (MSE for regression) based on OOB samples. Then the same is done after permuting the jth predictor in the OOB samples. The difference between the two (before and after permutation) is then averaged over all trees, and further normalized by the corresponding standard deviation[a].

[a]If the standard deviation of the differences is equal to 0 for a variable, then the division is not applied.

Boosting Trees

- Boost the performance of a set of weak regression trees by cleverly combing them.
- Forward stagewise additive modeling: consider an 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 it using a forward stagewise greedy algorithm.

Forward Stagewise Optimization

1. $F(x) = 0$ and record the current residual $r_i^{(0)} = y_i$
2. For $t = 1$ to T
   - Fit a regression tree f_t to the current residual $r_i^{(t-1)}$
   - Add f_t to F: $F = F + f_t$
   - Update the current residual $r_i^{(t)} = r_i^{(t-1)} - f_t(x_i)$

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

- Advantages of ensemble methods based on trees
  - Less-processing is needed, e.g., NA can be handled automatically, and no scaling/normalization is required
  - Can handle large number of predictors
- GBM vs randomForest
  - randomForest has less number of tuning parameters, while GBM has more, but with proper tuning, GBM can perform better than randomForest.
- Categorical Predictors
  - Each package treats categorical predictors differently: maximal 32 levels for randomForest and 1024 levels for GBM; XGBoost and some python packages only take numerical input.

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

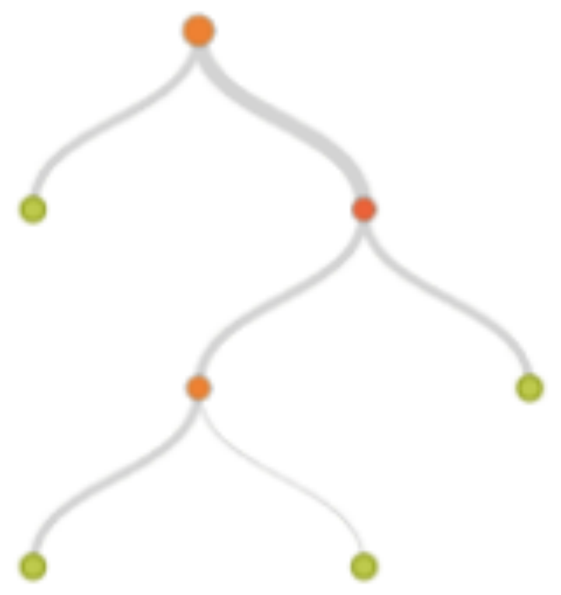
Tree Models

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

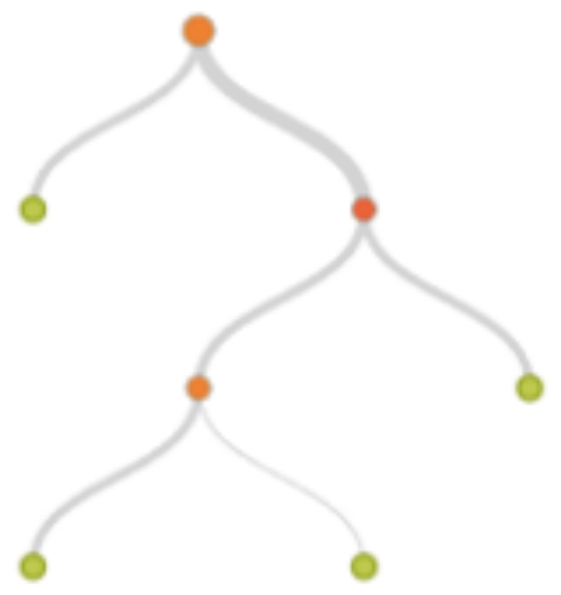
http://www.r2d3.us/visual-intro-to-machine-learning-part-1/

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

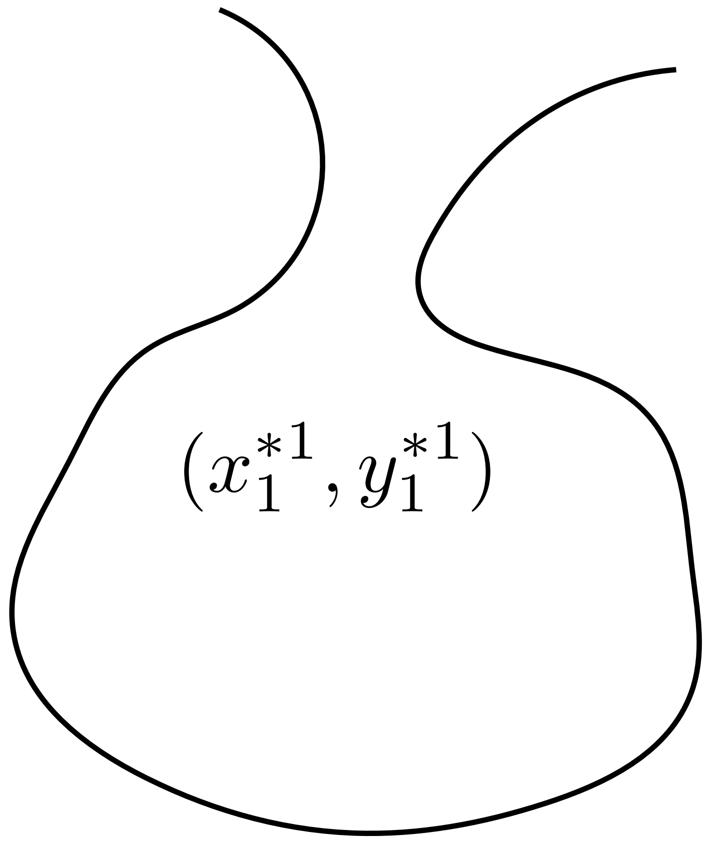
Case Study: Ames Housing Data

Bagging (Bootstrap Aggregation)

Training Data

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

Bootstrap Sample 1

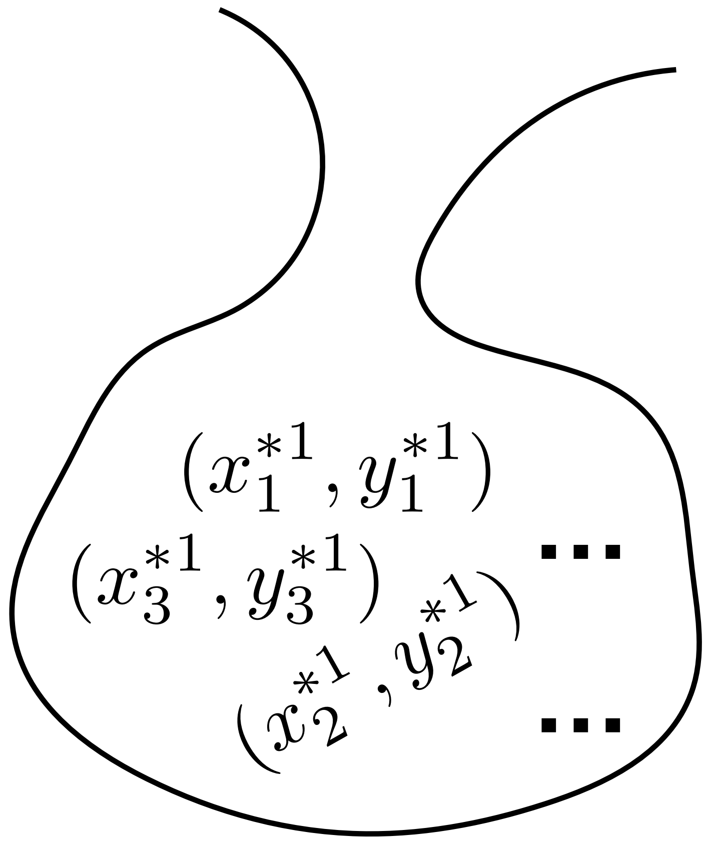


Sample with Replacement

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

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

Bootstrap Sample 1

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

Bootstrap Sample 2

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

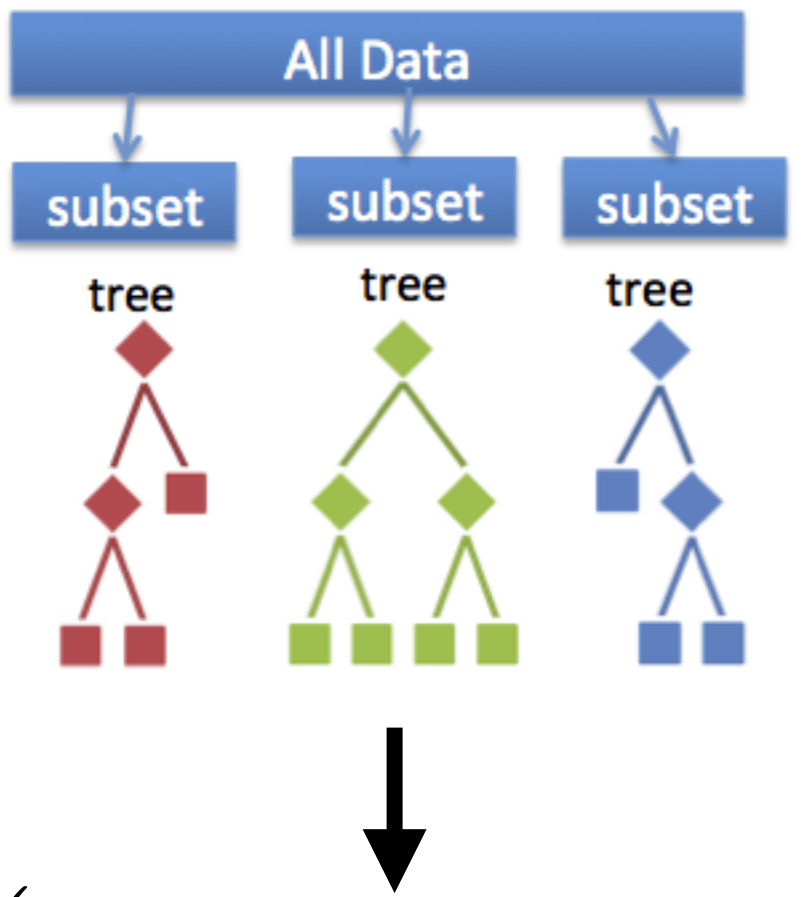
......

Bootstrap Sample B

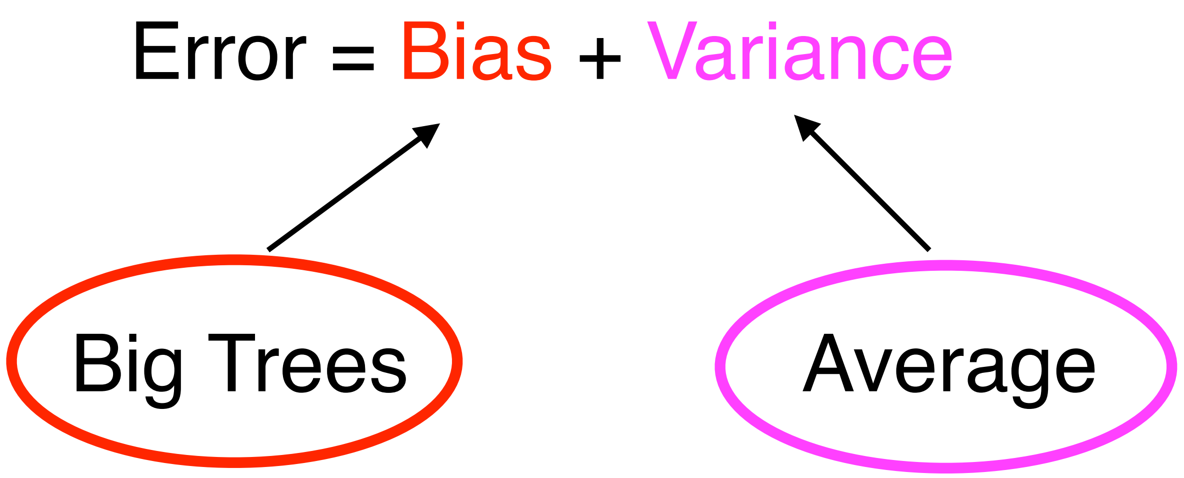
$$\boxed{\mathbf{Z}^{*B} = \{x_1^{*B}, y_1^{*B}),\ (x_2^{*B}, y_2^{*B}), \quad \cdots, \quad (x_n^{*B}, y_n^{*B})\}}$$

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.

$$\begin{tikzcd} \text{Error} = & \text{Bias} & + & \text{Variance} \\ & \boxed{\text{Big Trees}} \arrow[u] & & \boxed{\text{Average}} \arrow[u, "?"'] \end{tikzcd}$$

$$\downarrow$$

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

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

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

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

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

```
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

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

```
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
```

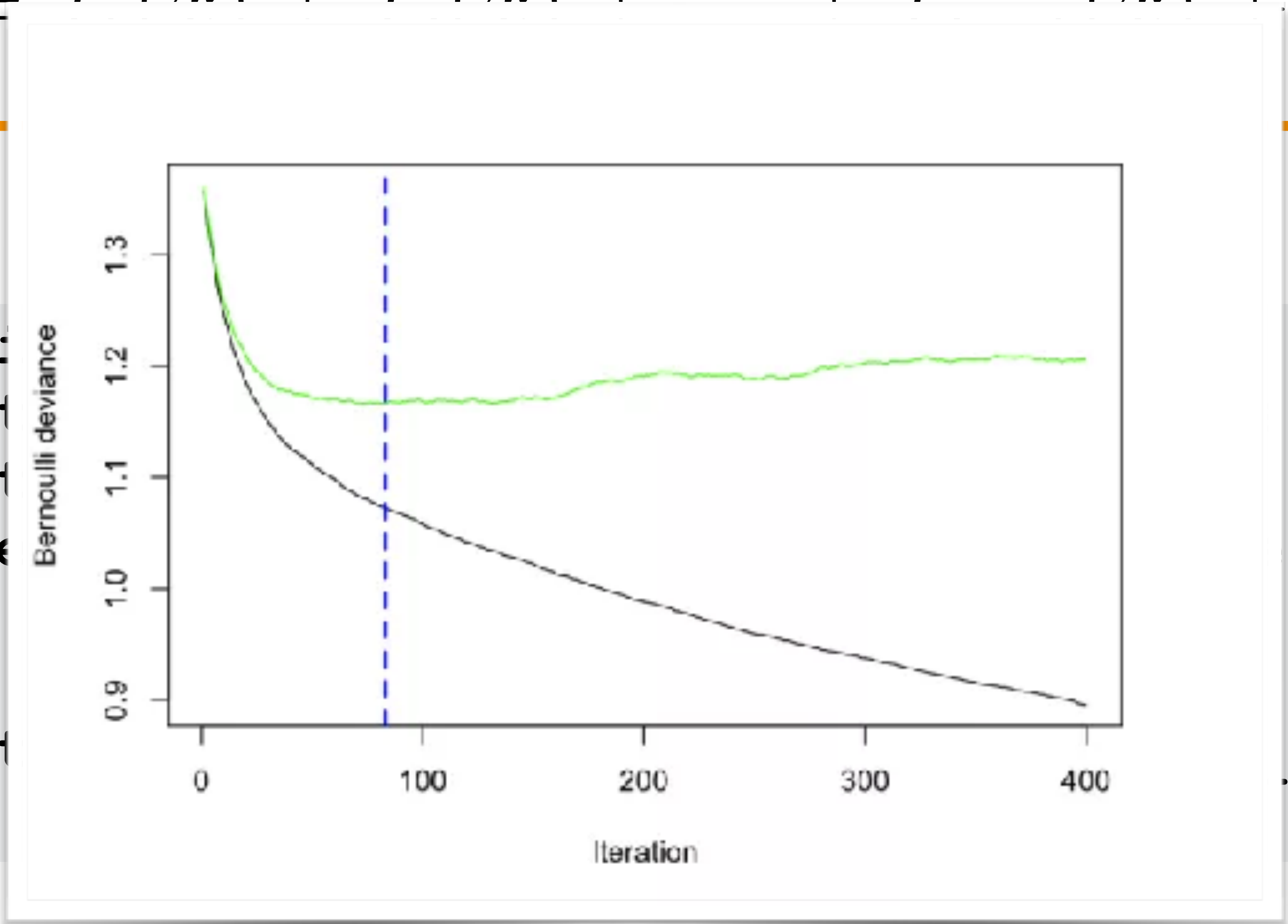
$\boxed{f\_t <\!\!- \ \eta f\_t}$

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

Forward Stage-wise Optimization

$$F(x) = f_{\ }(x) + f_{\ }(x) + \quad + f_{\ }(x) + f_T(x)$$

```
Input: Tra
Initializa                                      r_i = y
for t = 1 t
|   Fit a re                                    siduals r_i
|   Add f_t
|
|   Update t                                    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

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

```
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

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

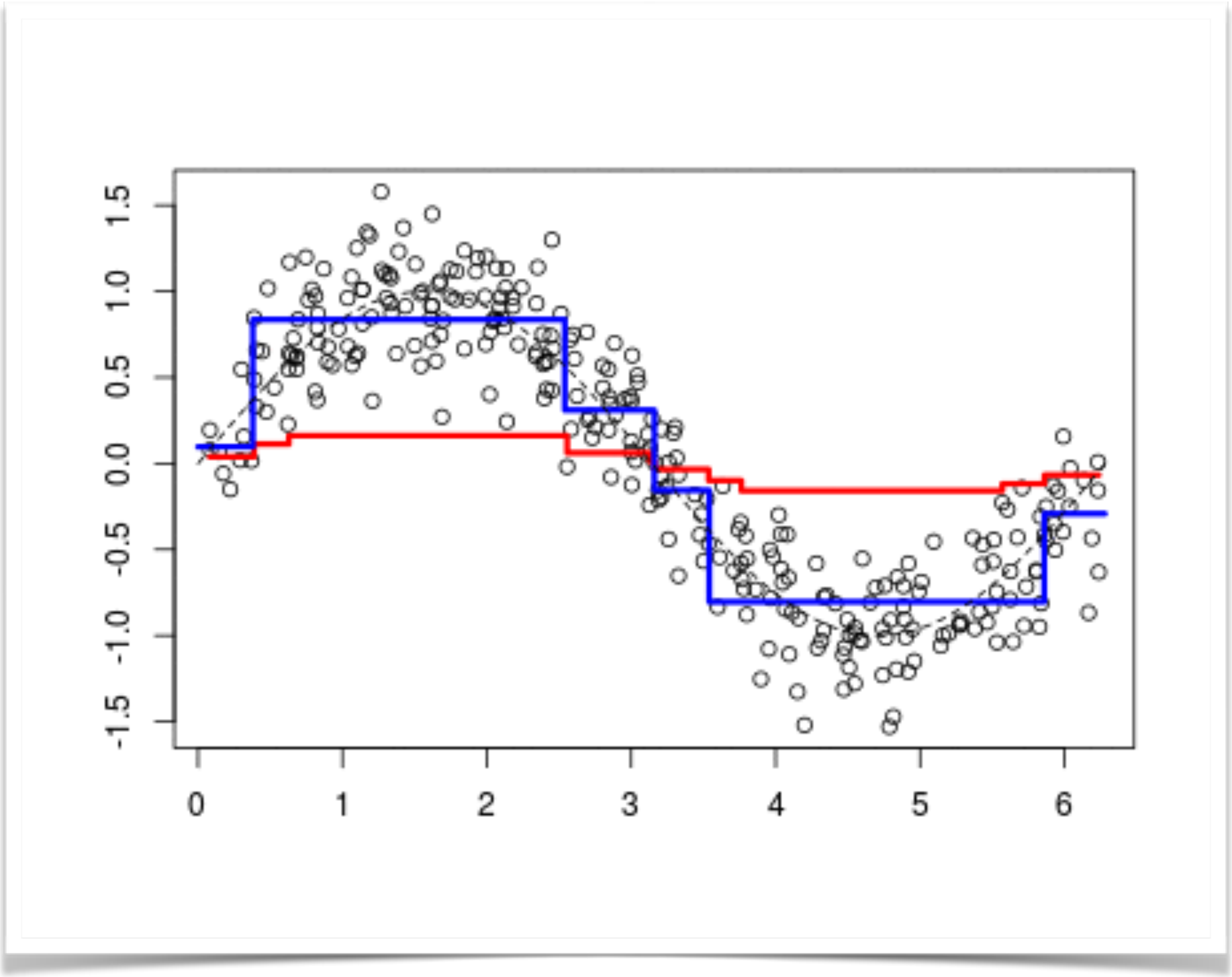
```
Input: Training Data, T (# of iterations)
Initialization: F(x) = 0 and current residuals r_i = y
for t = 1 to T do
|   Fit a regression tree f_t to the current residuals r_i
|   Add f_t t
|
|   Update th
|   end
```

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

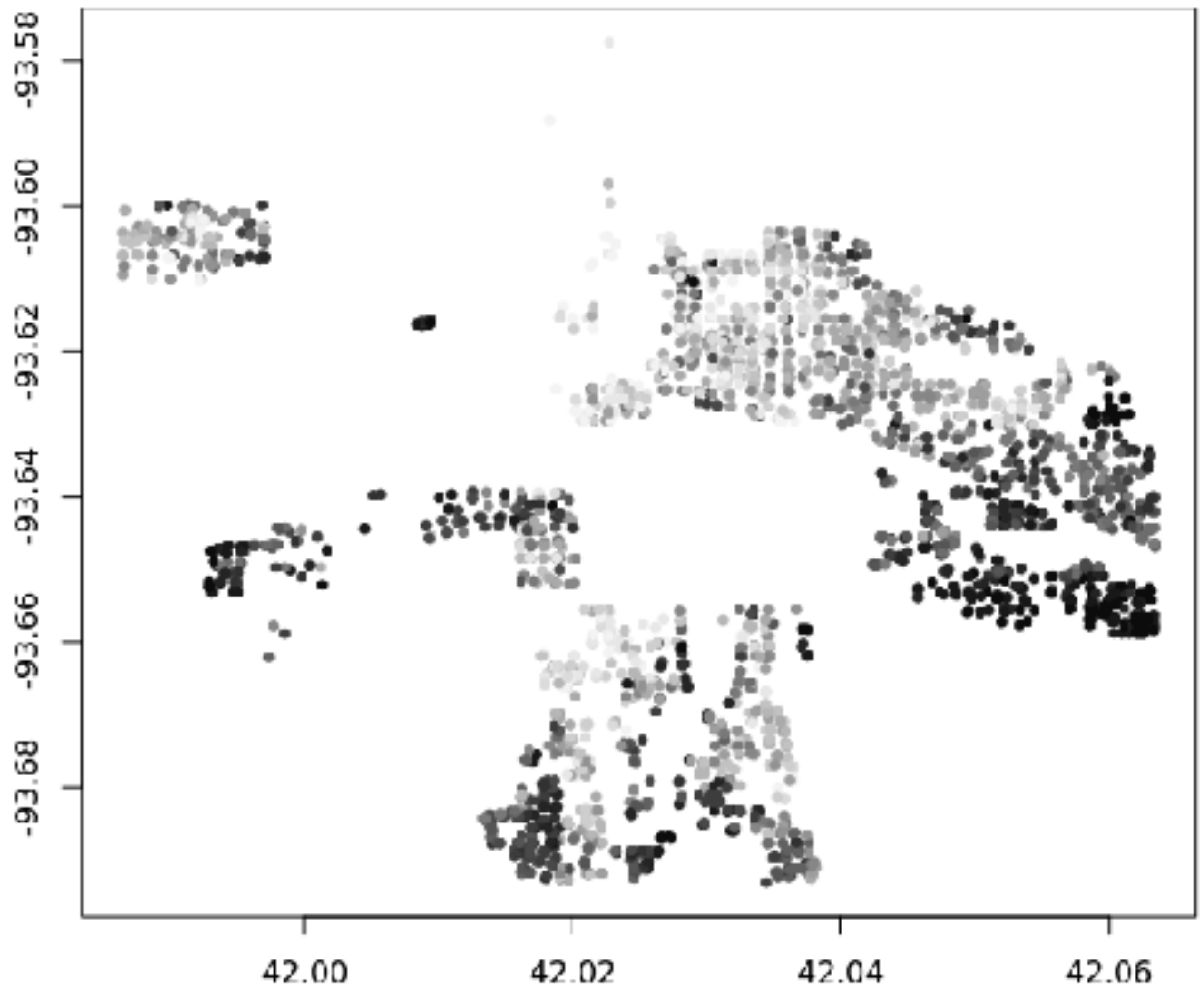
Ames Housing Data

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"
```



randomForest

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

```
> 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))
```

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
)
```

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

```
> 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

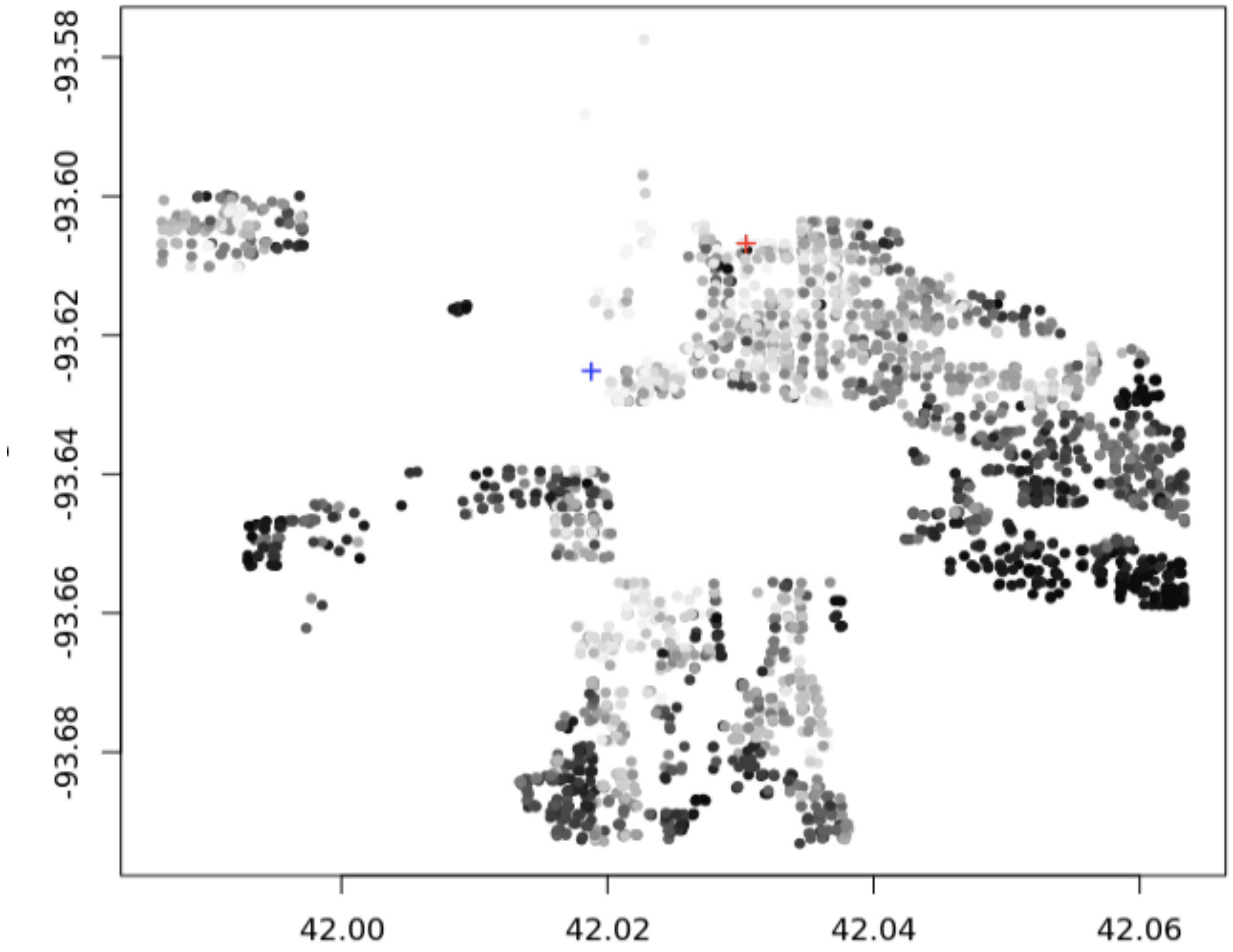
10 Splits of Training/Test
Target perf < 0.132

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 | 00023344577777778801123333367889
 112 | 003333334455566678888888899999999999999000001111111222223333444455555+19
 114 | 000000001111111122222233344444455555556666666677778888888899999999+98
 116 | 000000000001111111111111111111111111111222222223333333333333333444444+406
 118 | 0000000000000000001111111111111111111111111111111111111111111111111+585
 120 | 0000000000000000000000000111111111111111111111111111111111122222222222+474
 122 | 000000000000001111111111111111111111111111111111122222222222222222223333+314
 124 | 000000000111111111112222222222222222333333333333333333333333344444444455+173
 126 | 00001111111111111122222333333444444445555556666666666666677777777777778888+56
 128 | 0012222233333333333344444555556666666778889999900011111222223344444455666
 130 | 01122334455667789122239
 132 | 012338892235
 134 | 23
```