CMSC/LING/STAT 208: Machine Learning

Abhishek Chakraborty [Much of the content in these slides have been adapted from *ISLR2* by James et al. and *HOMLR* by Boehmke & Greenwell]

Remedies for Class Imbalance

For binary classification problems, in some contexts, the class of interest (`Positive` class) might have a very low frequency. We will see how to approach the ML process when such cases arise.

We will look at three possible remedies.

- Model tuning
- Alternative cutoffs
- Subsampling

We will work with the `attrition.rds` dataset.

Linear Model Selection and Regularization

The standard linear model,

$$Y = \beta_0 + \beta_1 X_1 + \cdots + \beta_p X_p + \epsilon$$

- **Benefits**: Simple, interpretable, often shows good predictive performance.
- **Limitations**:
  - **Prediction Accuracy**: If n is not big (also when $p > n$), variance can be high resulting in overfitting and poor predictive performance.
  - **Model Interpretability**: Irrelevant features lead to unnecessary model complexity.

Alternatives to Least Squares

Extensions/Modifications/Improvements to Least Squares:

- **Subset Selection**: Identify a subset of p predictors and then fit a linear model using least squares.
- **Shrinkage/Regularization**: Fit a model using p predictors, but shrink some of the estimated coefficients towards zero. Reduces variance and can also perform variable selection.
- **Dimension Reduction**: Project p predictors onto a M-dimensional subspace, where $M < p$. Achieved by computing M different linear combinations or projections of the p predictors. Fit a linear model using these M predictors by least squares.

Shrinkage/Regularization Methods

Fit a model containing all p predictors using a technique that **shrinks** the coefficient estimates towards zero.

- Ridge Regression
- Lasso

Shrinking the coefficient estimates significantly reduces their variance.

The Lasso

Acronym for **Least Absolute Shrinkage and Selection Operator**.

- **Standard Linear Model**

Given a training dataset, for $i = 1, \ldots, n$

$$\hat{y}_i = b_0 + b_1 x_{i1} + \ldots + b_p x_{ip}$$

$$\text{SSE} = \sum_{i=1}^{n} \left(y_i - \hat{y}_i \right)^2 = \sum_{i=1}^{n} \left(y_i - (b_0 + b_1 x_{i1} + \ldots + b_p x_{ip}) \right)^2$$

- **Lasso**

$$\text{SSE} + \lambda \sum_{j=1}^{p} |b_j|$$

- $\lambda \sum_{j=1}^{p} |b_j|$: **Shrinkage Penalty**
- $\lambda \geq 0$: **Tuning/Regularization Parameter**

The Lasso

- This method not only shrinks the coefficient estimates towards zero, but also makes some of the coefficient estimates exactly equal to zero (when the tuning parameter λ is sufficiently large).
- Hence, the lasso performs **variable selection**.
- We say that the lasso yields **sparse models**, that is, models that involve only a subset of the variables.

The Lasso: Scaling of Predictors

- Standard least squares (regression) coefficient estimates are **scale equivariant**: multiplying X_j by a constant c simply leads to a scaling of the least squares coefficient estimates by a factor of $1/c$. In other words, regardless of how the j^{th} predictor is scaled, $X_j\hat{\beta}_j$ will remain the same.
- In contrast, the lasso coefficient estimates can change substantially when multiplying a given predictor by a constant. Apply lasso after **standardizing** the predictors.

$$\tilde{x}_{ij} = \frac{x_{ij}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_{ij} - \overline{x}_j)^2}}$$

The Lasso: Implementation

Ames Housing Dataset

```
ames <- readRDS("AmesHousing.rds")   # load dataset
```

```
# reorder levels of 'Overall_Qual'
ames$Overall_Qual <- factor(ames$Overall_Qual, levels = c("Very_Poor", "Poor", "Fair", "Below_Average",
                                                          "Average", "Above_Average", "Good", "Very_Good",
                                                          "Excellent", "Very_Excellent"))
```

```
# split data

set.seed(050724)   # set seed

train_index <- createDataPartition(y = ames$Sale_Price, p = 0.7, list = FALSE)   # consider 70-30 split

ames_train <- ames[train_index,]   # training data

ames_test <- ames[-train_index,]   # test data
```

The Lasso: Implementation

Ames Housing Dataset

```
# create recipe and blueprint, prepare and apply blueprint

set.seed(050724)   # set seed

ames_recipe <- recipe(Sale_Price ~ ., data = ames_train)   # set up recipe

blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%   # filter out zv/nzv predictors
  step_impute_mean(Gr_Liv_Area) %>%                                    # impute missing entries
  step_integer(Overall_Qual) %>%                                       # numeric conversion of levels of the predictors
  step_center(all_numeric(), -all_outcomes()) %>%                      # center (subtract mean) all numeric predictors
  step_scale(all_numeric(), -all_outcomes()) %>%                       # scale (divide by standard deviation) all numeric predictors
  step_other(Neighborhood, threshold = 0.01, other = "other") %>%      # lumping required predictors
  step_dummy(all_nominal(), one_hot = FALSE)                           # one-hot/dummy encode nominal categorical predictors


prepare <- prep(blueprint, data = ames_train)    # estimate feature engineering parameters based on training data


baked_train <- bake(prepare, new_data = ames_train)   # apply blueprint to training data

baked_test <- bake(prepare, new_data = ames_test)    # apply blueprint to test data
```

The Lasso: Implementation

Ames Housing Dataset

Implement CV to tune the hyperparameter λ.

```
set.seed(050724)   # set seed

cv_specs <- trainControl(method = "repeatedcv", number = 5, repeats = 5)   # CV specifications

lambda_grid <- 10^seq(-3, 4, length = 100)   # grid of lambda values to search over

library(glmnet)  # for LASSO

lasso_cv <- train(blueprint,
                  data = ames_train,
                  method = "glmnet",   # for lasso
                  trControl = cv_specs,
                  tuneGrid = expand.grid(alpha = 1, lambda = lambda_grid),   # alpha = 1 implements lasso
                  metric = "RMSE")

# results from the CV procedure

lasso_cv$bestTune$lambda    # optimal lambda
```

```
## [1] 1204.504
```

```
min(lasso_cv$results$RMSE)   # RMSE for optimal lambda
```

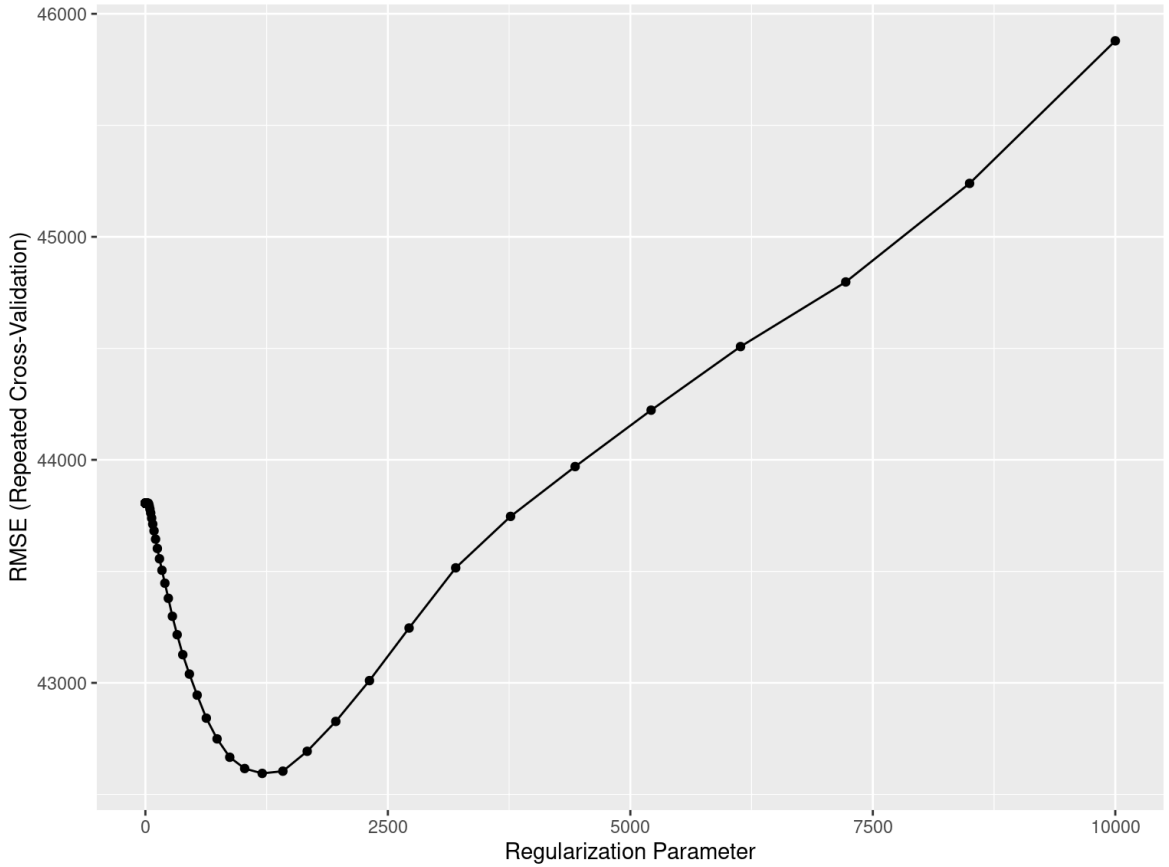
```
## [1] 42593.99
```

The Lasso: Implementation

Ames Housing Dataset

Results from the CV procedure.

```
ggplot(lasso_cv)   # lambda vs. RMSE plot
```



The Lasso: Implementation

Ames Housing Dataset

We will now build the optimal lasso model on the modified training data using the optimal λ.

```
# create datasets required for 'glmnet' function

X_train <- model.matrix(Sale_Price ~ ., data = baked_train)[, -1]   # training features without intercept

Y_train <- baked_train$Sale_Price    # training response

X_test <- model.matrix(Sale_Price ~ ., data = baked_test)[, -1]   # test features without intercept
```

```
# build optimal lasso model

final_model <- glmnet(x = X_train,
                      y = Y_train,
                      alpha = 1,                          # alpha = 1 builds lasso model
                      lambda = lasso_cv$bestTune$lambda,  # using optimal lambda from CV
                      standardize = FALSE)                # already standardized during data preprocessing
```

```
# obtain predictions and test set RMSE

final_model_preds <- predict(final_model, newx = X_test)    # obtain predictions

sqrt(mean((final_model_preds - baked_test$Sale_Price)^2))   # calculate test set RMSE
```

```
## [1] 29903.92
```

The Lasso: Implementation

Ames Housing Dataset

The coefficients for the optimal lasso model can be obtained from

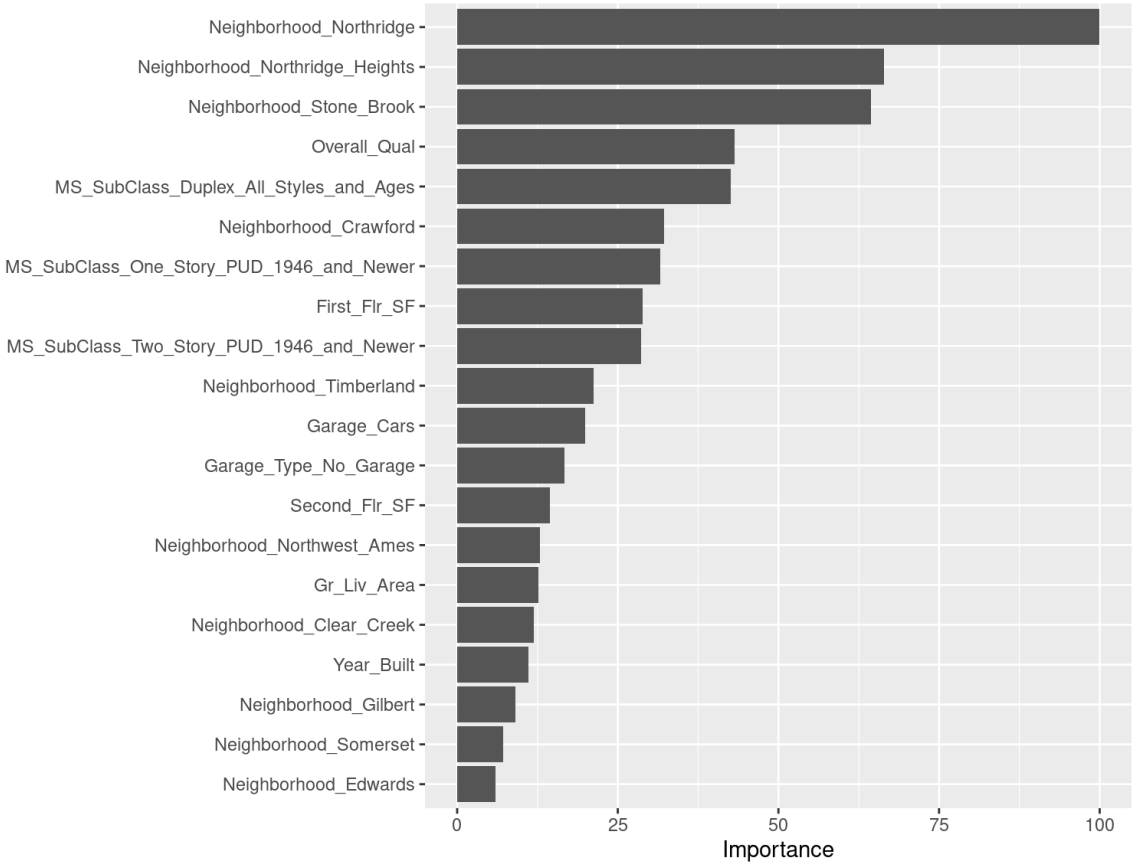
```
coef(final_model)    # estimated coefficients from final lasso model
```

```
## 54 x 1 sparse Matrix of class "dgCMatrix"
##                                                        s0
## (Intercept)                                    181313.5819
## Gr_Liv_Area                                      6917.8414
## Garage_Cars                                     10466.0529
## Garage_Area                                       217.2498
## Overall_Qual                                    35210.8237
## Lot_Area                                         2714.6661
## Lot_Frontage                                      612.9751
## Open_Porch_SF                                     316.2876
## TotRms_AbvGrd                                    -338.8492
## First_Flr_SF                                    23960.4079
## Second_Flr_SF                                   12380.3553
## Year_Built                                       7099.8829
## Garage_Type_Basment                                    .
## Garage_Type_BuiltIn                                    .
## Garage_Type_CarPort                                    .
## Garage_Type_Detchd                                     .
## Garage_Type_More_Than_Two_Types                        .
## Garage_Type_No_Garage                                  .
## Neighborhood_College_Creek                             .
## Neighborhood_Old_Town                                  .
## Neighborhood_Edwards                                   .
## Neighborhood_Somerset                                  .
## Neighborhood_Northridge_Heights                 15388.8984
## Neighborhood_Gilbert                                   .
## Neighborhood_Sawyer                                    .
## Neighborhood_Northwest_Ames                            .
## Neighborhood_Sawyer_West                               .
## Neighborhood_Mitchell                                  .
## Neighborhood_Brookside                                 .
## Neighborhood_Crawford                                  .
## Neighborhood_Iowa_DOT_and_Rail_Road                    .
## Neighborhood_Timberland                                .
## Neighborhood_Northridge                          5196.8736
## Neighborhood_Stone_Brook                               .
## Neighborhood_South_and_West_of_Iowa_State_University   .
```

The Lasso: Implementation

Ames Housing Dataset

```
# variable importance

vip(object = lasso_cv, num_features = 20, method = "model")
```



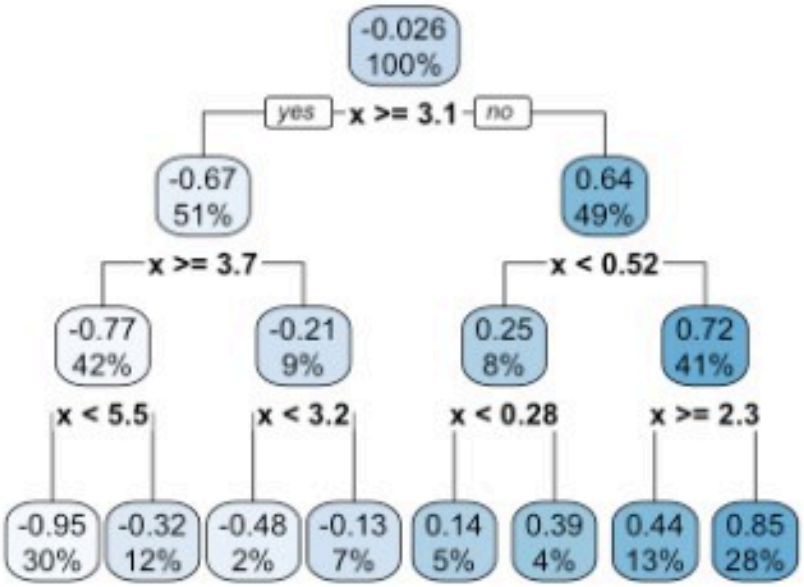
Tree-Based Methods

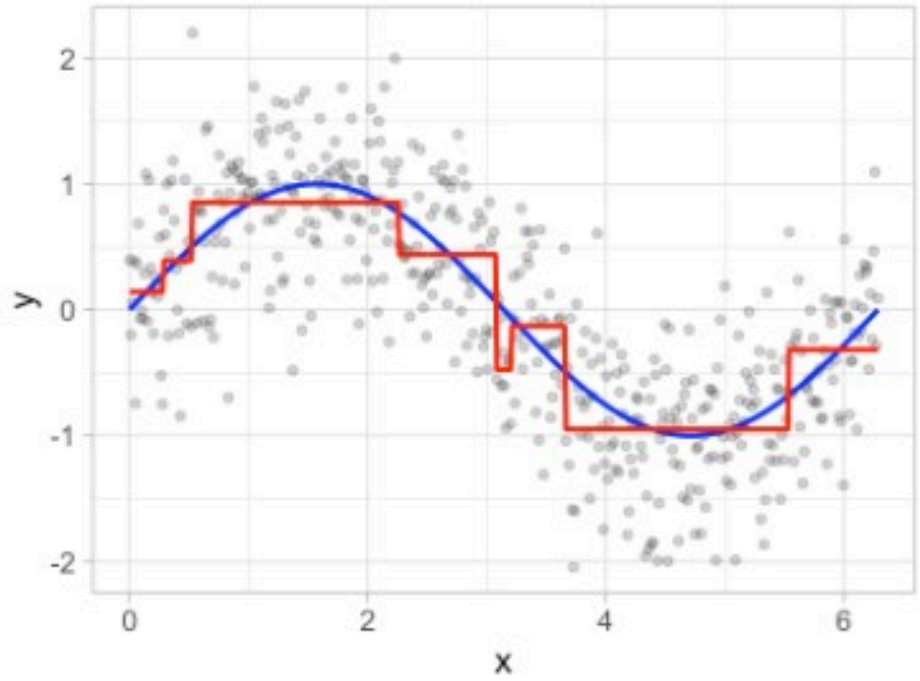
- Involves **stratifying** or **segmenting** the predictor space into a number of simple regions.
- The set of splitting rules used to segment the predictor space can be summarized in a tree, thus, the name **decision tree** methods.
- Can be used for both classification and regression.
- Tree-based methods are simple and useful for interpretation, however, not the best in terms of prediction accuracy.
- Methods such as **bagging**, **random forests**, and **boosting** grow multiple trees and then combine their results.

Terminology for Trees

- Every split is considered to be a **node**.
- We refer to the first node at the top of the tree as the **root node** (this node contains all of the training data).
- The final nodes at the bottom of the tree are called the **terminal nodes** or **leaves**.
- Decision trees are typically drawn **upside down**, in the sense that the leaves are at the bottom of the tree.
- The points along the tree where the predictor space is split are referred to as **internal nodes**, that is, every node in between the **root node** and **terminal nodes** is referred to as an **internal node**.
- The segments of the trees that connect the nodes are known as **branches**.

Terminology for Trees





Adapted from HMLR, Boehmke & Greenwell

Building a Tree

- First select the predictor X_j and the cutpoint s such that splitting the predictor space into the regions $\{X|X_j < s\}$ and $\{X|X_j \geq s\}$ leads to the greatest possible reduction in SSE.

For any j and s, define

$$R_1 = \{X|X_j < s\} \text{ and } R_2 = \{X|X_j > s\}$$

Find j and s that minimize

$$SSE = \sum_{i \in R_1} \left(y_i - \hat{y}_{R_1}\right)^2 + \sum_{i \in R_2} \left(y_i - \hat{y}_{R_2}\right)^2$$

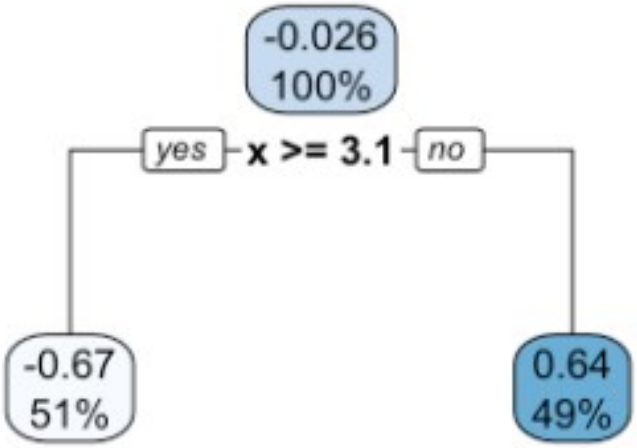
- Next, repeat the process, look for the best predictor and best cutpoint in order to split the data further. However, this time, instead of splitting the entire predictor space, split one of the two previously identified regions.
- The process continues until a stopping criterion is reached; say, we may continue until no region contains more than five observations.

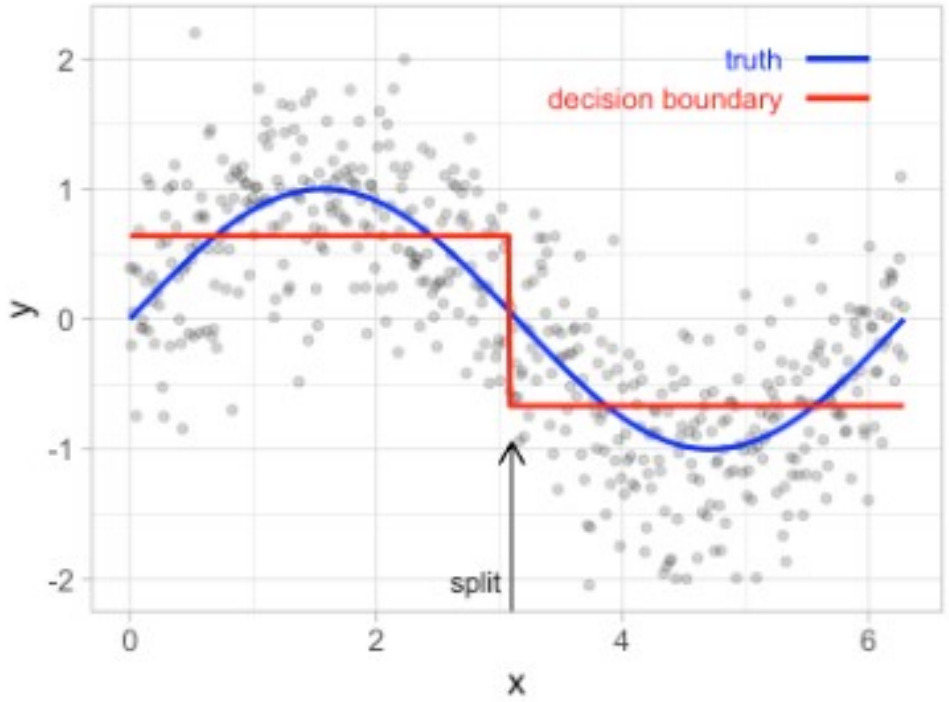
Prediction

For every observation that falls into the region R_j, make the same prediction, which is

- the mean response of the training set observations in R_j (for regression problems),
- majority vote response of the training set observations in R_j (for classification problems).

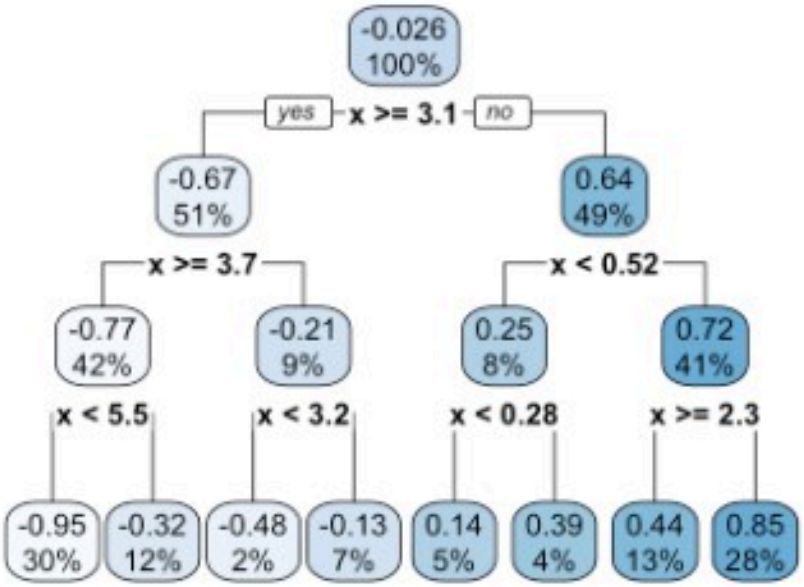
Building a Tree and Prediction

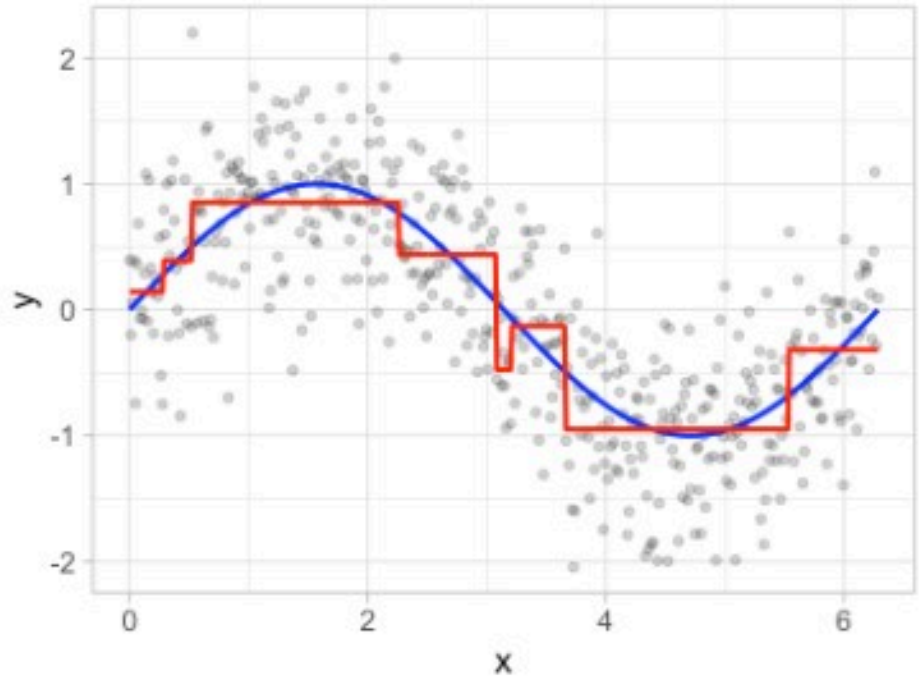




Adapted from HMLR, Boehmke & Greenwell

Building a Tree and Prediction

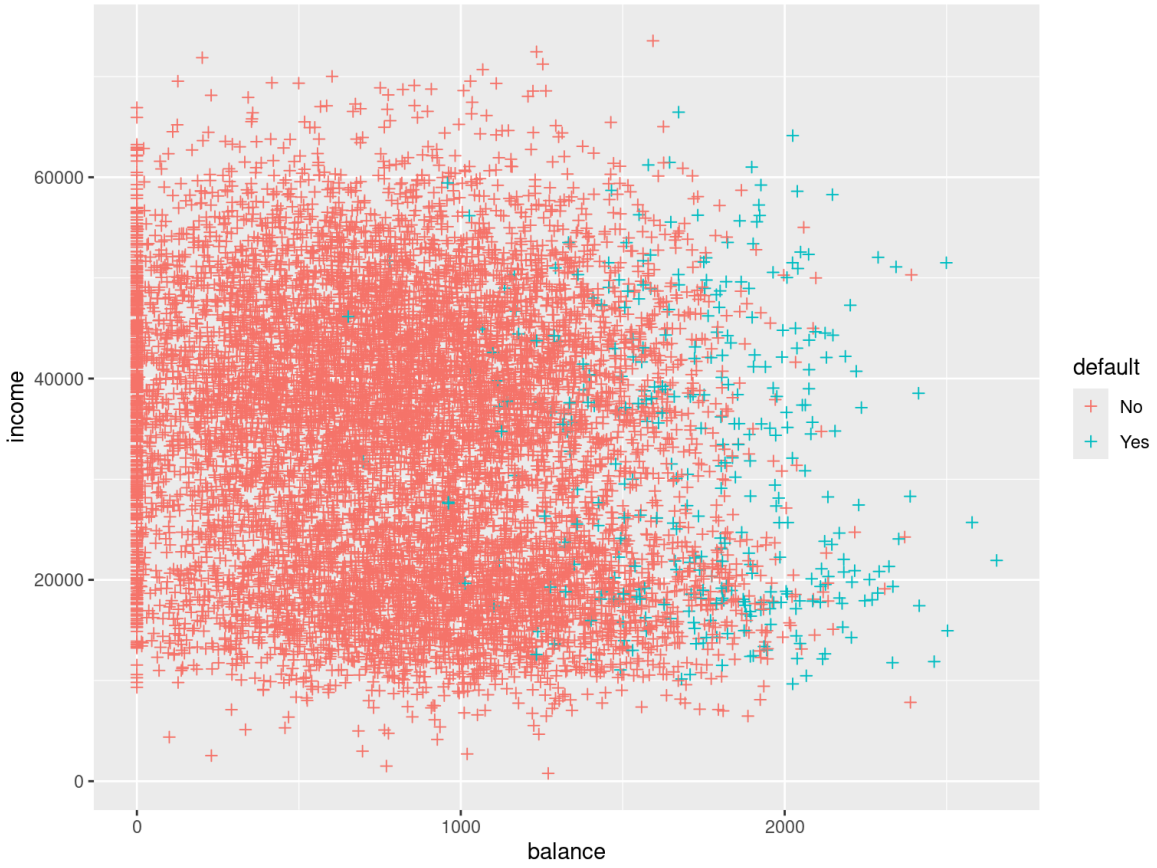




Adapted from HMLR, Boehmke & Greenwell

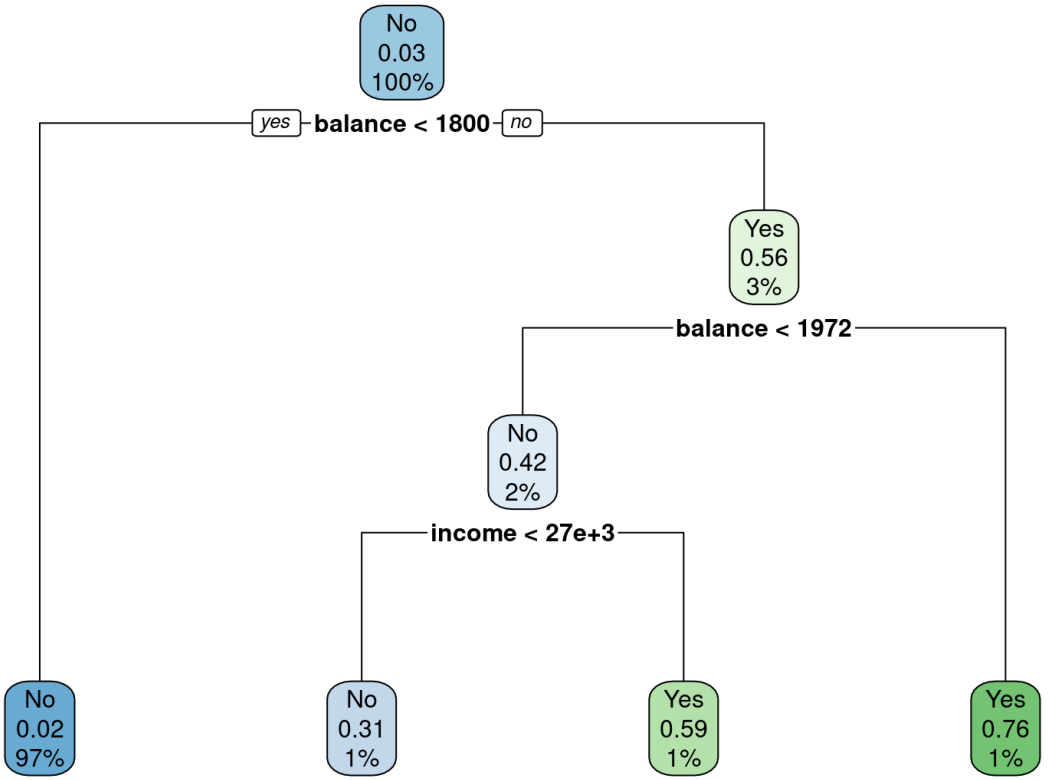
Building a Tree and Prediction

Default Dataset

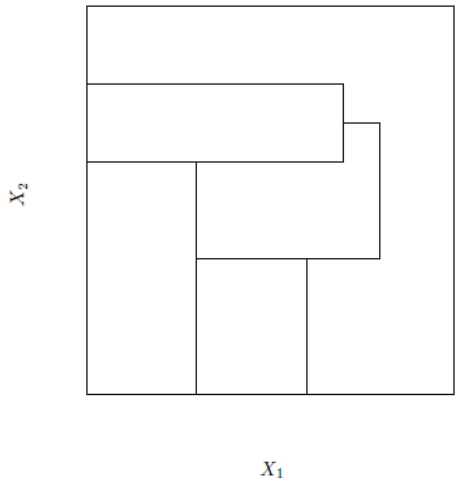


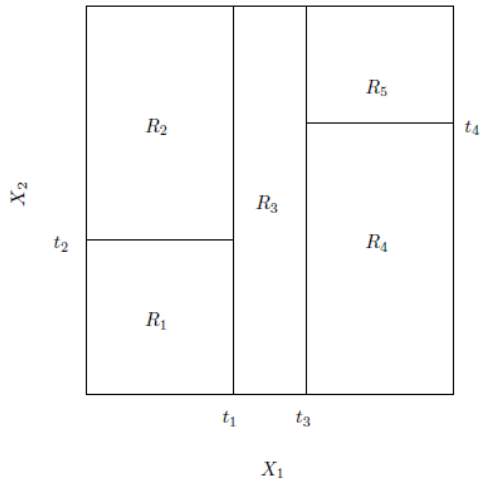
Building a Tree and Prediction

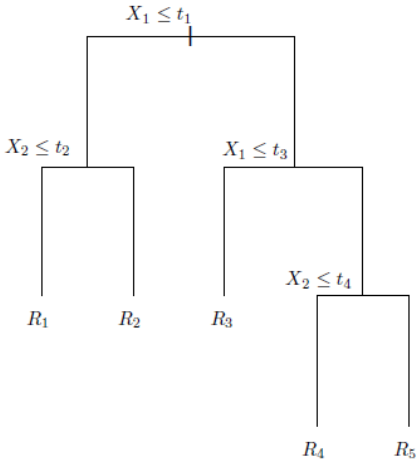
Default Dataset

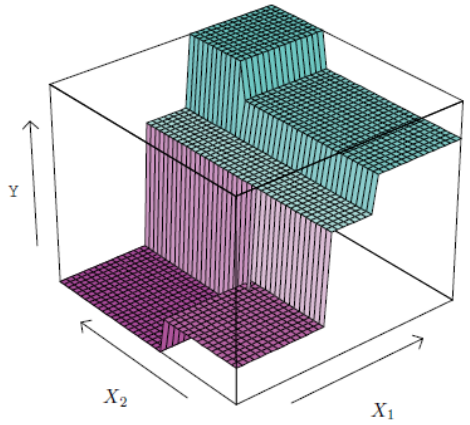


Building a Tree and Prediction









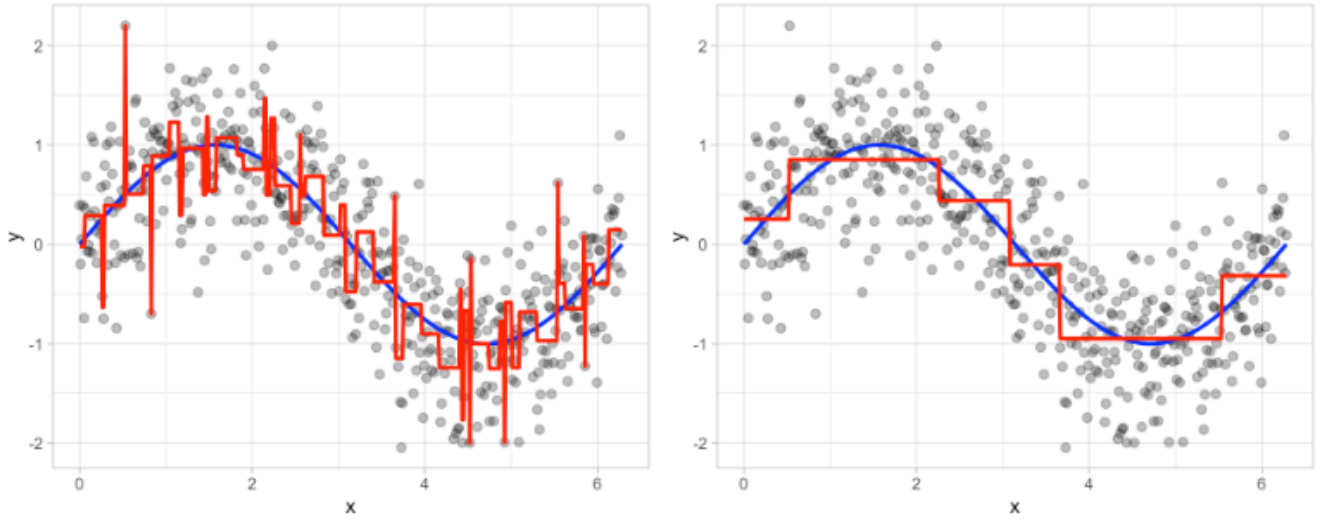
Adapted from ISLR, James et al.

Building a Tree

- It is computationally infeasible to consider every possible partition of the feature space into J boxes.
- For this reason, we take a **top-down, greedy** approach known as **recursive binary splitting**.
  - **top-down** because it begins at the top of the tree and then successively splits the predictor space.
  - **greedy** because at each step of the tree-building process, the best split is made at that particular step, rather than looking ahead and picking a split that will lead to a better tree in some future step.

Tree Pruning

The process described above may **overfit** the data.



Tree Pruning

- Grow a very large tree, and then **prune** it back to obtain a **subtree**.
- The technique uses is known as **cost complexity pruning** (also known as **weakest link pruning**).
- Consider a sequence of trees indexed by α. For each α, consider the tree that minimizes

$$SSE + \alpha \left|T\right|$$

- Choose optimal α by CV.

Regression Tree: Implementation

Ames Housing Dataset

```
ames <- readRDS("AmesHousing.rds")   # load dataset
```

```
# reorder levels of 'Overall_Qual'
ames$Overall_Qual <- factor(ames$Overall_Qual, levels = c("Very_Poor", "Poor", "Fair", "Below_Average",
                                                          "Average", "Above_Average", "Good", "Very_Good",
                                                          "Excellent", "Very_Excellent"))
```

```
# split data

set.seed(050924)   # set seed

train_index <- createDataPartition(y = ames$Sale_Price, p = 0.7, list = FALSE)   # consider 70-30 split

ames_train <- ames[train_index,]   # training data

ames_test <- ames[-train_index,]   # test data
```

Regression Tree: Implementation

Ames Housing Dataset

```
# create recipe and blueprint, prepare and apply blueprint

set.seed(050924)   # set seed

ames_recipe <- recipe(Sale_Price ~ ., data = ames_train)   # set up recipe

blueprint <- ames_recipe %>%
  step_nzv(Street, Utilities, Pool_Area, Screen_Porch, Misc_Val) %>%   # filter out zv/nzv predictors
  step_impute_mean(Gr_Liv_Area) %>%                                    # impute missing entries
  step_integer(Overall_Qual) %>%                                       # numeric conversion of levels of the predictors
  step_center(all_numeric(), -all_outcomes()) %>%                      # center (subtract mean) all numeric predictors
  step_scale(all_numeric(), -all_outcomes()) %>%                       # scale (divide by standard deviation) all numeric predictors
  step_other(Neighborhood, threshold = 0.01, other = "other") %>%      # lumping required predictors
  step_dummy(all_nominal(), one_hot = FALSE)                           # one-hot/dummy encode nominal categorical predictors


prepare <- prep(blueprint, data = ames_train)    # estimate feature engineering parameters based on training data


baked_train <- bake(prepare, new_data = ames_train)   # apply the blueprint to training data

baked_test <- bake(prepare, new_data = ames_test)    # apply the blueprint to test data
```

Regression Tree: Implementation

Ames Housing Dataset

Implement CV to tune the hyperparameter.

```
set.seed(050924)   # set seed

cv_specs <- trainControl(method = "repeatedcv", number = 5, repeats = 5)   # CV specifications


library(rpart)   # for trees

tree_cv <- train(blueprint,
                 data = ames_train,
                 method = "rpart",
                 trControl = cv_specs,
                 tuneLength = 20,                  # considers a grid of 20 possible tuning parameter values
                 metric = "RMSE")
```

```
# results from the CV procedure

tree_cv$bestTune    # optimal hyperparameter
```

```
##           cp
## 1 0.002631828
```

```
min(tree_cv$results$RMSE)   # optimal CV RMSE
```

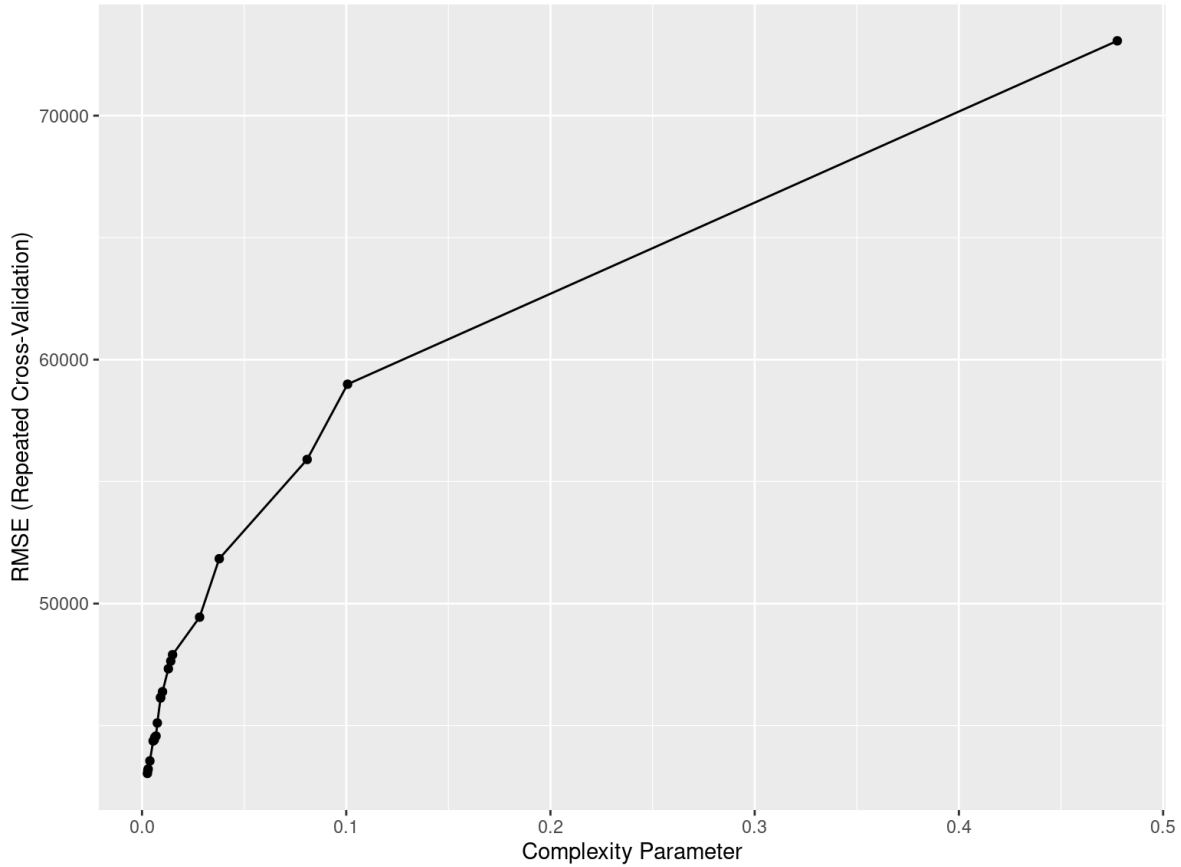
```
## [1] 43039.21
```

Regression Tree: Implementation

Ames Housing Dataset

Results from the CV procedure.

```
ggplot(tree_cv)
```



Regression Tree: Implementation

Ames Housing Dataset

```
# build final model

final_model <- rpart(formula = Sale_Price ~ .,
                     data = baked_train,
                     cp = tree_cv$bestTune$cp,
                     xval = 0,                # no further CV
                     method = "anova")        # for regression
```

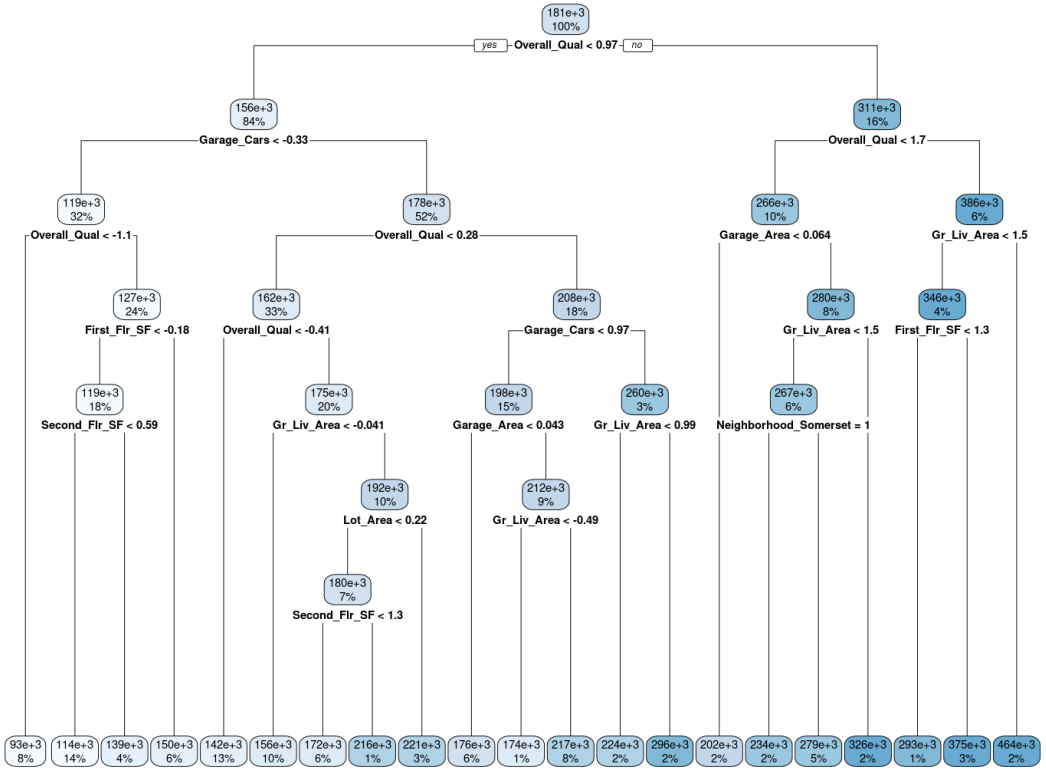
```
# obtain predictions and test set RMSE

final_model_preds <- predict(object = final_model, newdata = baked_test, type = "vector")    # obtain test set predictions

sqrt(mean((final_model_preds - baked_test$Sale_Price)^2))   # calculate test set RMSE
```

```
## [1] 39241.28
```

Regression Tree: Implementation

Ames Housing Dataset

```
library(rpart.plot)
rpart.plot(final_model)
```



Regression Tree: Implementation

Ames Housing Dataset

```
# variable importance

vip(object = tree_cv, num_features = 20, method = "model")
```