Improving Regression and Communicating Results with Stochastic Gradient Boosting and LASSO

Charles Harrison

Salford Systems

Introduction

Classification and Regression Trees (CART) are easily interpreted models that can be presented to an audience

One of the drawbacks of a single CART tree is that its predictive performance relative to stochastic gradient boosting tends to be weaker

Although stochastic gradient boosting generally outperforms CART trees, one disadvantage of this technique is that the interpretation of the final model is diminished

Introduction

Often it is necessary to accept a tradeoff between a model that can be interpreted and one that has the best performance.

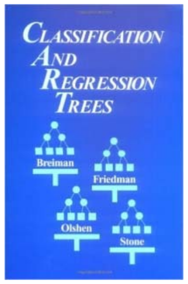
Central question: **Is there a technique that approximately maintains the predictive power of a gradient boosted tree and has a useful interpretation?**

Yes. We can take the “rules” (more on this later) from a gradient boosted tree and use them as predictors in a regularized regression model (i.e. Lasso, Ridge Regression, Elastic Net etc.)

Predictive Learning via Rule Ensembles (Friedman and Popescu, 2005)

Outline

CART: Introduction



Classification And Regression Trees

Authors: Breiman, Friedman, Olshen, and Stone (1984)

CART is a decision tree algorithm used for both regression and classification problems

1. Classification: tries to separate classes by choosing variables and points that best separate them
2. Regression: chooses the best variables and split points for reducing the squared or absolute error criterion

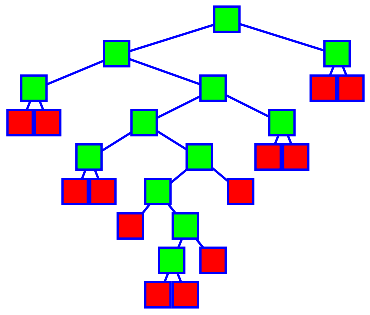
CART is available exclusively in the SPM® 8 Software Suite and was developed in close consultation with the original authors

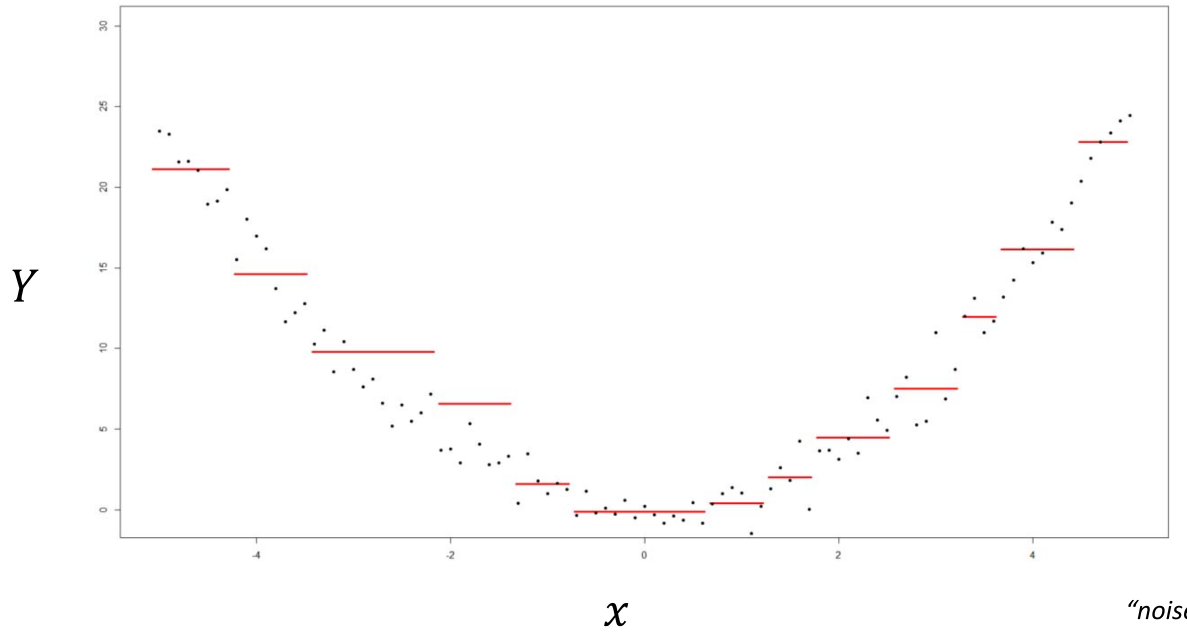
CART®

CART Introduction

Main Idea: divide the predictor variables (often people say “partition” instead of “divide”) into different regions so that the dependent variable can be predicted more accurately.

The following shows the predicted values from a CART tree (i.e. the red horizontal bars) to the curve $Y = x^2 + \varepsilon; \quad \varepsilon \sim \mathrm{N}(0,1)$.





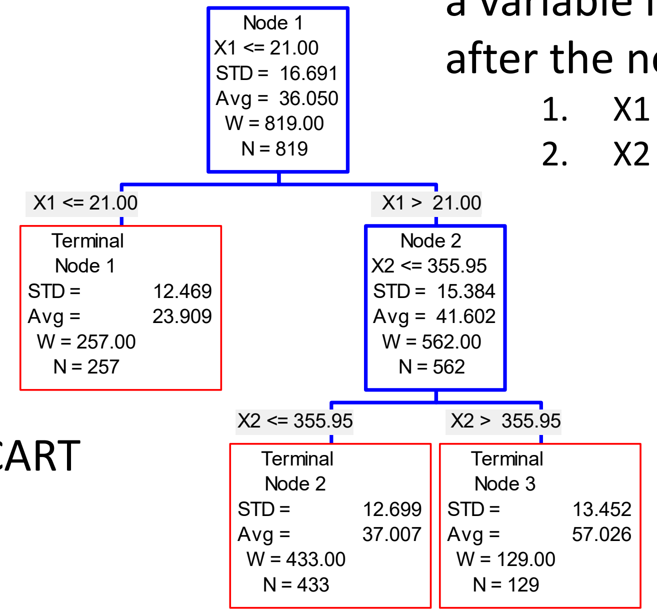
“noise” is from a N(0,1)

CART: Terminology

The node at the top of the tree is called the ***root node***

A tree ***split*** occurs when a variable is partitioned (in-depth example starts after the next slide). This tree has two splits:

1. X1 <=177
2. X2 <=355.95



A node that has no sub-branch is a ***terminal node***

This tree has three terminal nodes (i.e. red boxes in the tree)

The **predicted value** in a CART regression model is the average of the target variable (i.e. "Y") for the records that fall into one of the terminal nodes

Example: If X1 = 22 and X2 = 375 then the predicted value is 57.026

CART: Algorithm

Step 1: Grow a large tree

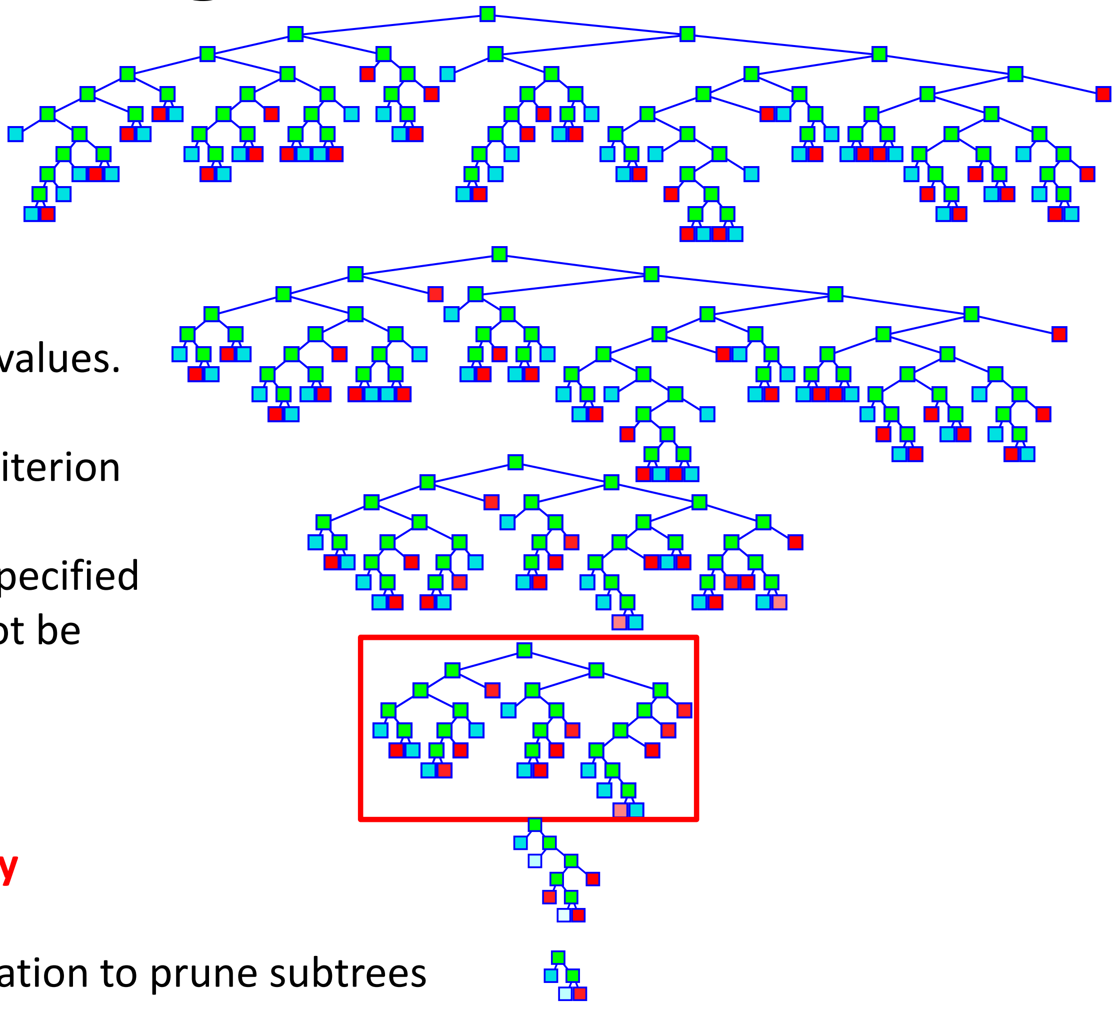
This is done for you automatically

All variables are considered at each split in the tree

Each split is made using <u>one</u> variable and a specific value or set of values.

Splits are chosen to optimize a split criterion

The tree is grown until either a user-specified criterion is met or until the tree cannot be grown further



Step2: Prune the large tree

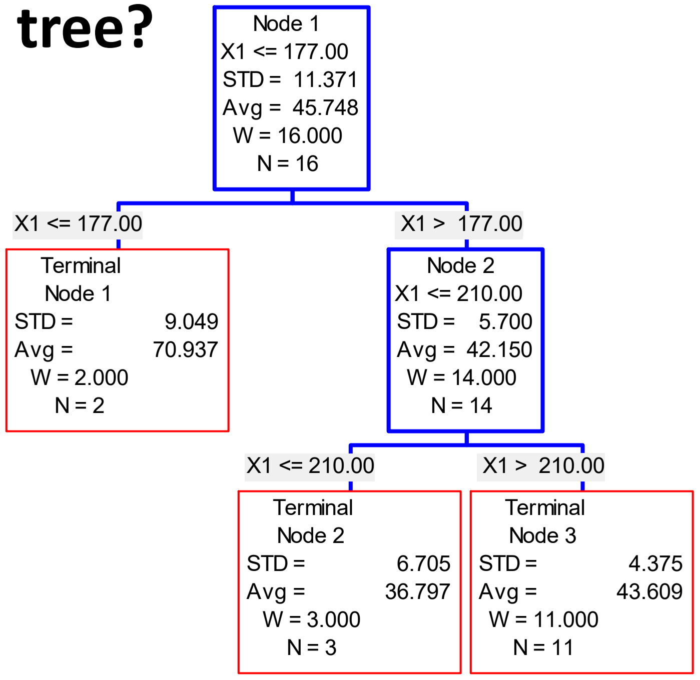
This is also done for you automatically

Use either a test sample or cross validation to prune subtrees

CART: Splitting Procedure

Consider the following CART tree grown on this dataset

How exactly do we get this tree?



Y	X_1	X_2
79.9861	162	28
61.8874	162	28
40.2695	228	270
41.0528	228	365
44.2961	192	360
47.0298	228	90
43.6983	228	365
36.4478	228	28
45.8543	228	28
39.2898	228	28
38.0742	192	90
28.0217	192	28
43.013	228	270
42.3269	228	90
47.8138	228	28
52.9083	228	90

CART: Splitting Procedure

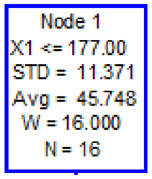
Step 1: Find the best split point for the variable X_1

- Sort the variable X_1
- Compute the split improvement for each split point
- Best split for X_1 :
  - $\boldsymbol{X_1 \leq 177}$

Split Improvement:

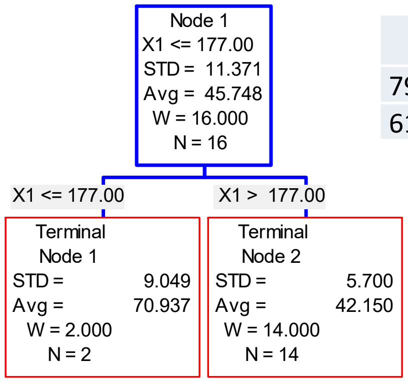
$$\Delta R(s,t) = R(t) - R(t_L) - R(t_R)$$

$$R(t) = \frac{1}{N} \sum_{x_n \epsilon t} (y_n - \bar{y}(t))^2 \ (Least\ Squares)$$



Note: the midpoint between X_1 = 192 and X_1 = 162 is 177

Y	X_1	X_2
79.99	162	28
61.89	162	28
28.02	192	28
38.07	192	90
44.3	192	360
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365



Y	X_1	X_2
79.99	162	28
61.89	162	28

Y	X_1	X_2
28.02	192	28
38.07	192	90
44.3	192	360
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365

CART: Splitting Procedure

Step 2: Find the best split point for the variable X_2

- Sort the variable X_2
- Compute the split improvement for each split point
- Best Split for X_2:
  - $\boldsymbol{X_2 \leq 59}$

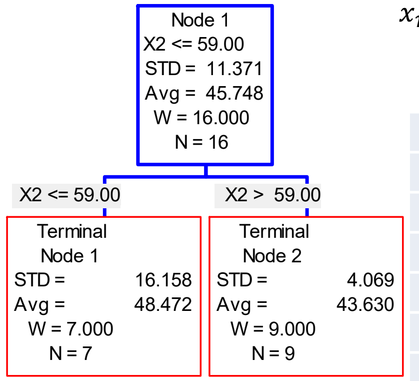
Split Improvement:

$$\Delta R(s,t) = R(t) - R(t_L) - R(t_R)$$

$$R(t) = \frac{1}{N} \sum_{x_n \epsilon t} (y_n - \bar{y}(t))^2 \ (Least\ Squares)$$

Note: the midpoint between X_2 = 28 and X_2 = 90 is 59

Y	X_1	X_2
79.99	162	28
61.89	162	28
28.02	192	28
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
38.07	192	90
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
44.30	192	360
41.05	228	365
43.70	228	365



Y	X_1	X_2
79.99	162	28
61.89	162	28
28.02	192	28
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28

Y	X_1	X_2
38.07	192	90
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
44.30	192	360
41.05	228	365
43.70	228	365

CART: Splitting Procedure

At this point CART has evaluated all possible split points for our two variables, X_1 and X_2, and determined the optimal split points for each.

Splitting on either X_1 or X_2 will yield a different tree, so what is the best split? The one with the largest split improvement.

Split Improvement:

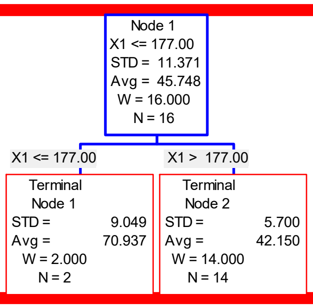
$$\Delta R(s,t) = R(t) - R(t_L) - R(t_R)$$

$$R(t) = \frac{1}{N} \sum_{x_n \epsilon t} (y_n - \bar{y}(t))^2$$

$(Least\ Squares)$

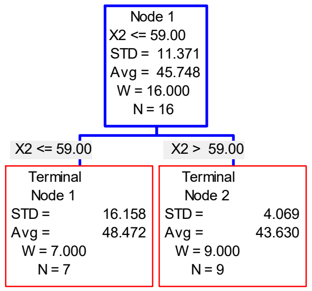
Best split for X_1: $X_1 \leq 177$

Improvement Value: 90.64



Best split for X_2: $X_2 \leq 59$

Improvement Value: 5.77



Recap: 1st Split in CART

Our best first split in the tree is $X_1 \leq 177$ which leads to the following tree and partitioned dataset

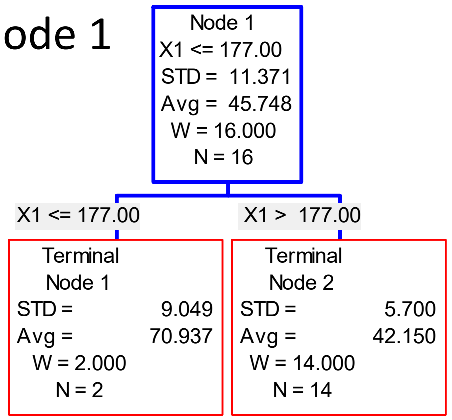
Y	X_1	X_2
79.99	162	28
61.89	162	28
28.02	192	28
38.07	192	90
44.3	192	360
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365

Y	X_1	X_2
79.99	162	28
61.89	162	28

Terminal Node 1

Y	X_1	X_2
28.02	192	28
38.07	192	90
44.3	192	360
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365

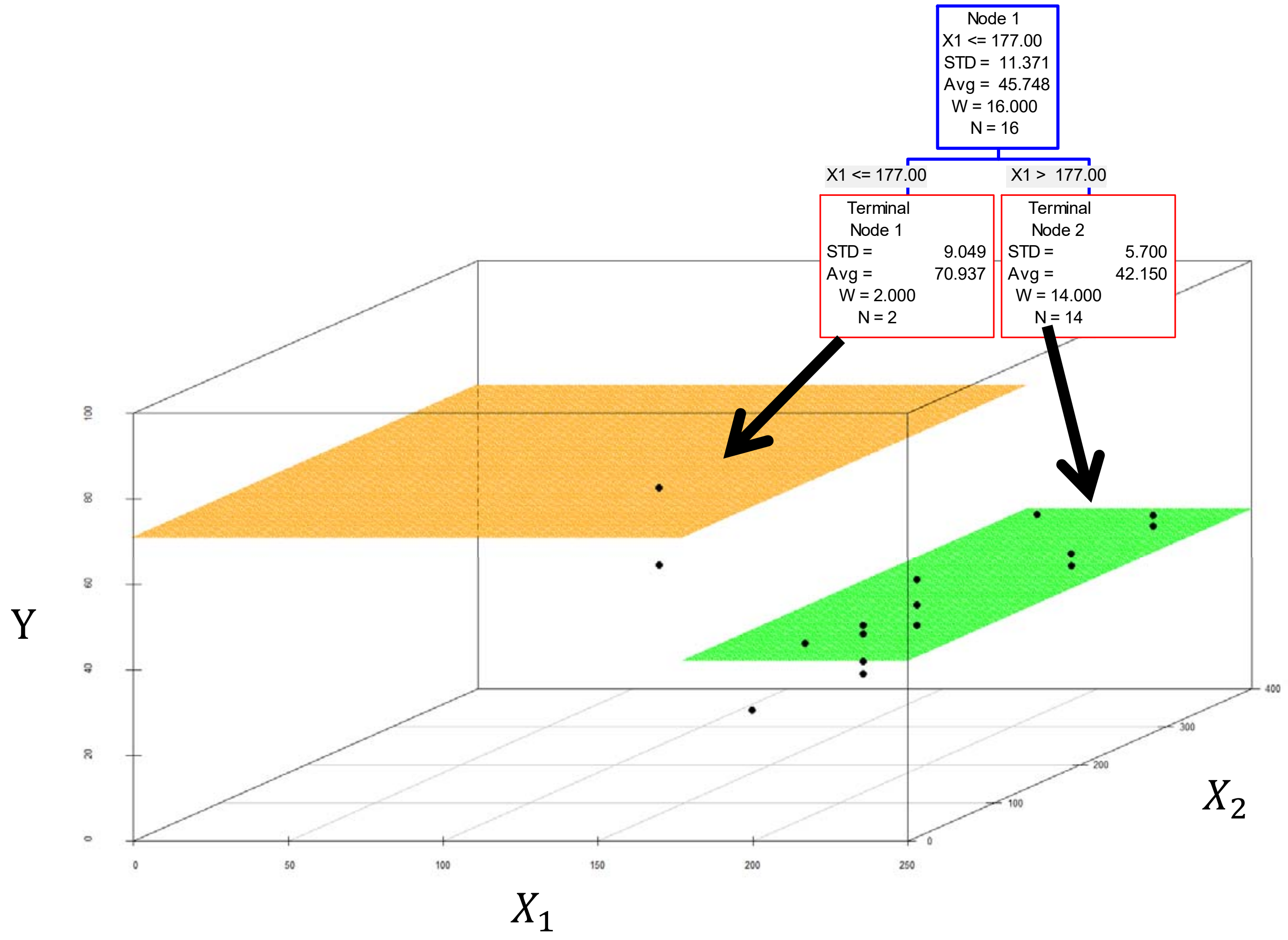
Terminal Node 2



Note: the predicted values for this tree are the respective target variable averages in each terminal node.

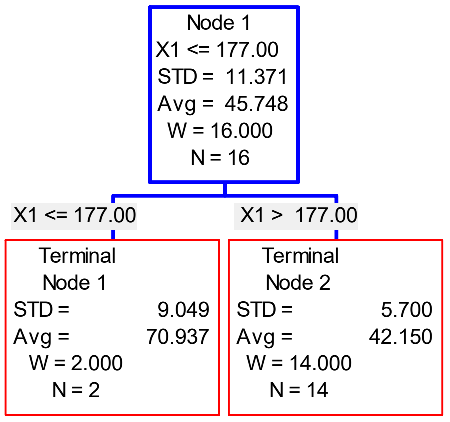
Terminal Node 1 predicted value: 79.99+61.89 ≈ 70.94

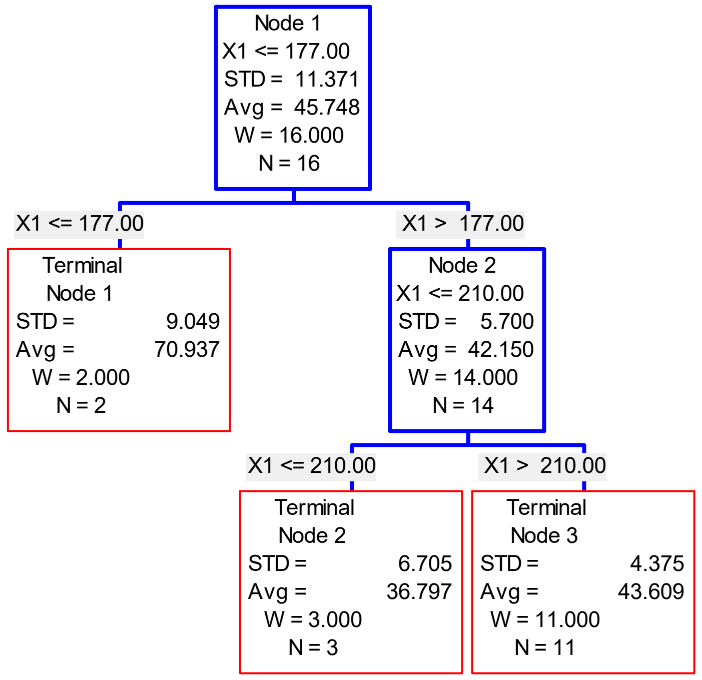
CART Geometry



CART: Splitting Procedure

So how do we get to our final tree?





CART: Splitting Procedure

We now perform the same procedure again, but this time for **each partition** of the data (we can only split one partition at a time)

***Best Split**: Split Partition 2 at $X_1 \leq 210$*

Partition 1

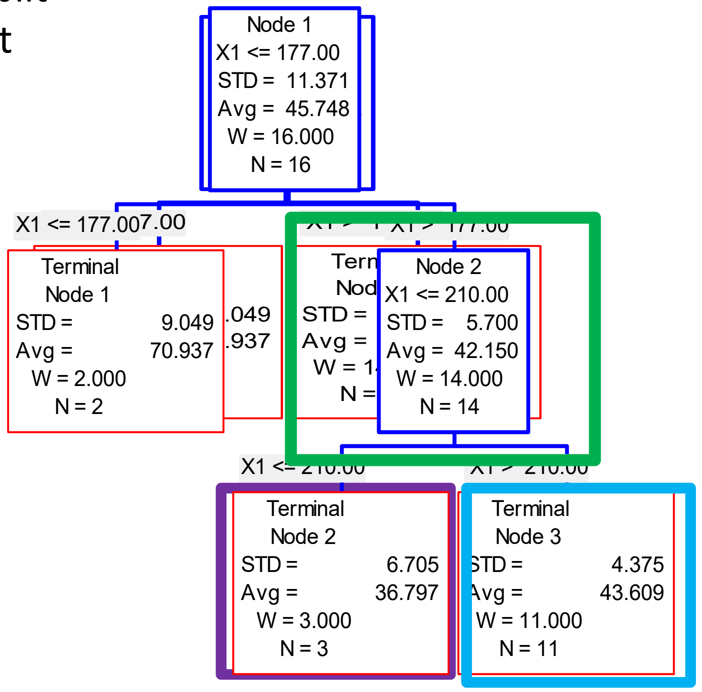
Y	X_1	X_2
79.99	162	28
61.89	162	28

Partition 2

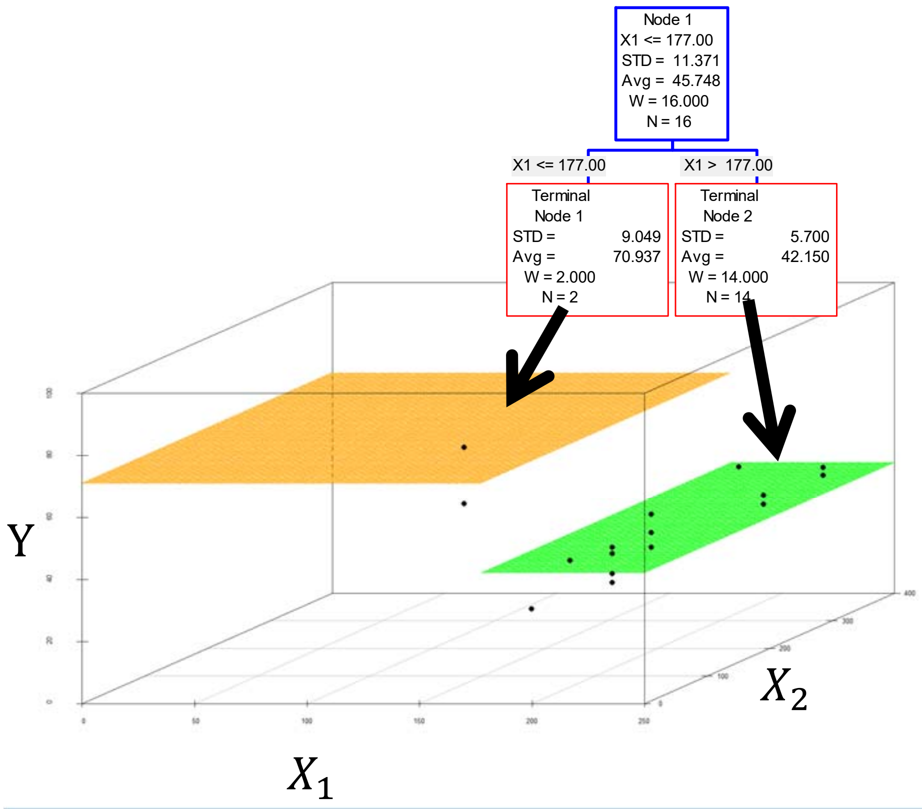
Y	X_1	X_2
28.02	192	28
38.07	192	90
44.3	192	360
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365

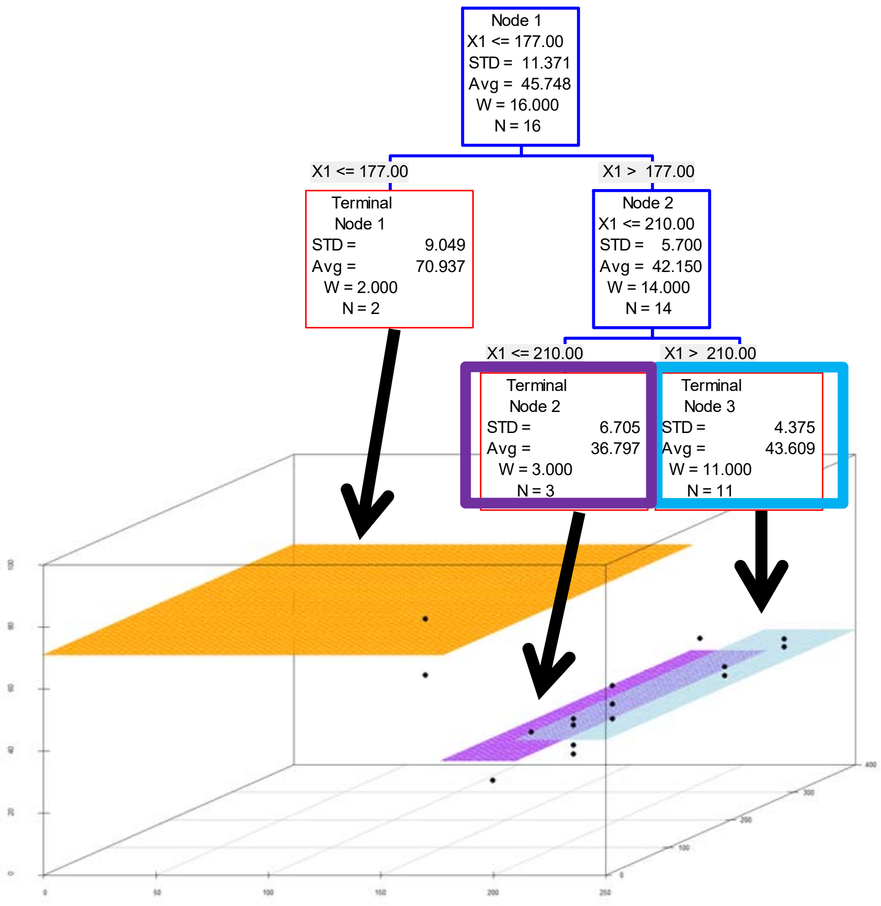
Y	X_1	X_2
28.02	192	28
38.07	192	90
44.3	192	360

Y	X_1	X_2
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365



CART Geometry





Where are we?

- ✓ CART Introduction
- ✓ CART Splitting Process
- ❑ **Advantages of CART**
- ❑ Gradient Boosting Section
- ❑ RuleLearner Section

CART Advantages

In practice, **you can build CART models with dirty data** (i.e. missing values, lots of variables, nonlinear relationships, outliers, and numerous local effects)

This is due to CART's desirable properties:

1. Automatic handling of the following:
   a) Variable selection **(see appendix)**
   b) Variable interaction modeling **(see video links and appendix)**
   c) Local effect modeling **(see video links and appendix)**
   d) Nonlinear relationship modeling **(see appendix)**
   e) Missing values **(see appendix)**
   f) Outliers **(see appendix)**
2. Not affected by monotonic transformations of variables **(see appendix)**

Where are we?

✓ CART Introduction

✓ CART Splitting Process

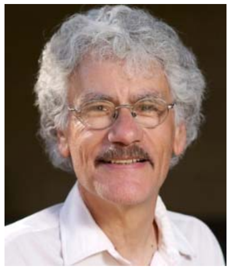
✓ Advantages of CART

❑ **Gradient Boosting Section**

❑ **RuleLearner Section**

Introduction to Stochastic Gradient Boosting

Main Idea: iteratively fit CART trees to "*generalized residuals*" (much more on this later)



Creator: Jerome Friedman

Friedman also co-created CART decision trees, created MARS regression splines, and co-created RuleLearner rule ensembles

The SPM modeling engine that implements the Stochastic Gradient Boosting algorithm is called TreeNet

The TreeNet code was originally written by Friedman himself

Gradient Boosting Algorithm Sketch (Least Squares)

For the least squares loss function in gradient boosting we are fitting CART trees to the residuals from the previous step

Main Idea:

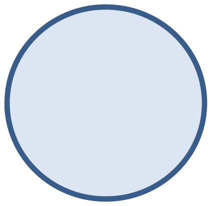
1. Fit an initial CART tree to the data and compute the residuals
2. Sample some records randomly
3. For the records sampled, fit a CART tree to the residuals from the previous step
4. Use the tree in Step 3 to update the model but shrink the update by a small number

Repeat Steps 2-4 M times

Gradient Boosting: Iteration 1 (Least Squares Loss)

Initial Prediction: Fit a CART tree, called $CART_{Init}$, to the records in the LEARN data. This produces an initial prediction for the records in the LEARN data

1. Draw a random sample S from the records in the learn data

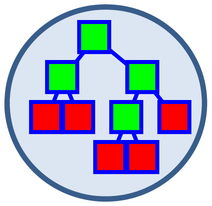


Sample

2. Compute the generalized residual $\tilde{y}_i = y_i - CART_{Init}$ for the records in S

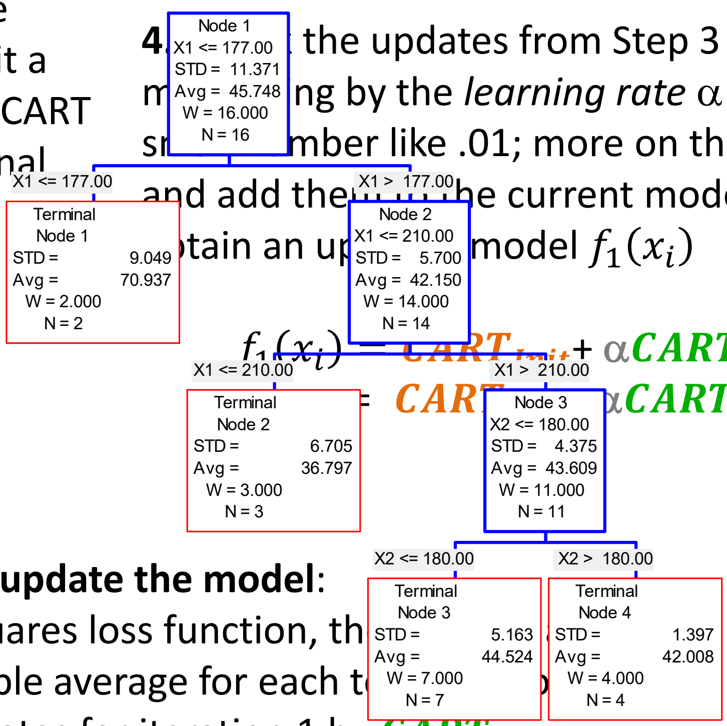
$\tilde{y}_i$	x_1	x_2
-3	11	13
.1	56	10
-.3	32	31

3. Using $\tilde{y}_i$ as the target variable, fit a small, unpruned CART tree with J terminal nodes using the records in S



Sample

4. [illegible] the updates from Step 3 by m[illegible]ng by the *learning rate* α (α is a sm[illegible]mber like .01; more on this later) and add the[illegible]he current model f_0 to [illegible]tain an up[illegible] model $f_1(x_i)$



$$f_1(x_i) = CART_{Init} + \alpha CART_1$$

$$= CART[illegible] \alpha CART_1$$

Use the tree to update the model:
for the least squares loss function, th[illegible] the target variable average for each t[illegible] Denote the updates for iteration 1 by $CART_1$

Least Squares Loss Function:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

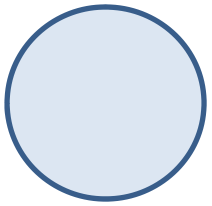
Gradient Boosting: Iteration 2 (Least Squares Loss)

Current Model: $f_1(x_i) = CART_{Init} + \alpha CART_1$

1. Draw a random sample S from the records in the learn data

2. Compute the generalized residual $\tilde{y}_i = y_i - f_1(x_i)$ for the records in S

1) How exactly do you compute a generalized residual ?



$\tilde{y}_i$	x_1	x_2
-1	7	12
-2	65	7
11	36	13

$$y_i - f_1(x_i) = y_i - CART_{Init} + \alpha CART_1 = ?$$

Least Squares Loss Function:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Computing a Generalized Residual

Current Model after the first iteration: $f_1(x_i) = CART_{Init} + \alpha CART_1$

Current Generalized Residual: $\tilde{y}_i = y_i - f_1(x_i) = y_i - (CART_{Init} + \alpha CART_1)$

$CART_1$ is the prediction via the tree fit to the generalized residuals during the first iteration: y_i- $CART_{Init}$

α is a small number like .01. We call α the "learning rate"

Let's say that the initial prediction for a randomly selected record is $CART_{Init}$ =42, α=.01, and a randomly selected record has the following values

Y	X1	X2
39	400	12.5

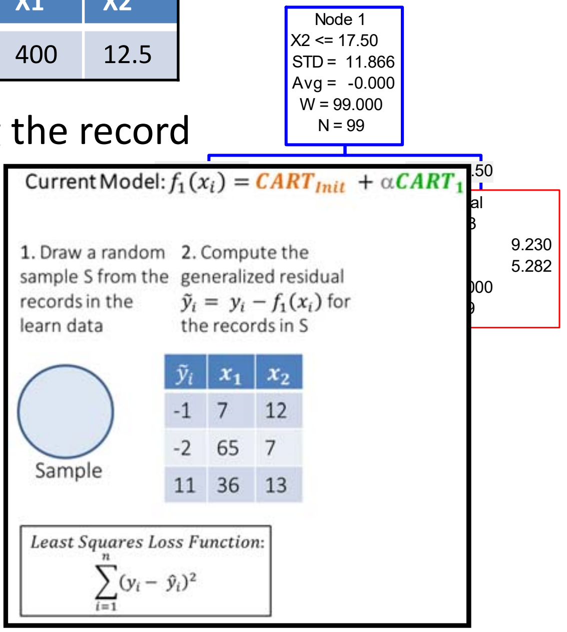
The value for $CART_1$ is just the terminal node average after running the record down the tree

$CART_1$ = - 2.027

Thus the generalized residual for this record is

$y_i - (CART_{Init} + \alpha CART_1)$ = 39 – (42 + .01*(-2.027)) = $\boxed{\text{- 2.97}}$

Repeat this process for all of the records selected in Step 1



Current Model: $f_1(x_i) = CART_{Init} + \alpha CART_1$

1. Draw a random sample S from the records in the learn data

2. Compute the generalized residual $\tilde{y}_i = y_i - f_1(x_i)$ for the records in S

Sample

$\tilde{y}_i$	x_1	x_2
-1	7	12
-2	65	7
11	36	13

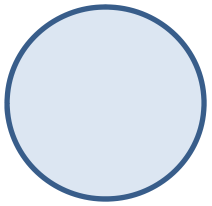
Least Squares Loss Function:

$$\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Gradient Boosting: Iteration 2 (Least Squares Loss)

Current Model: $f_1(x_i) = \boldsymbol{CART_{Init}} + \alpha \boldsymbol{CART_1}$

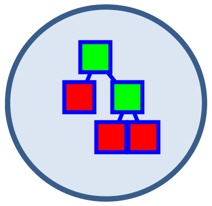
1. Draw a random sample S from the records in the learn data



2. Compute the generalized residual $\tilde{y}_i = y_i - f_1(x_i)$ for the records in S

$\tilde{y}_i$	x_1	x_2
-1	7	12
-2	65	43
11	36	19

3. Using $\tilde{y}_i$ as the target variable, fit a small, unpruned CART tree with J terminal nodes using the records in S



4. Shrink the updates from Step 3 by multiplying by the *learning rate* α (α is a small number like .01; more on this later) and add them to the current model $f_1(x_i)$ to obtain an updated model $f_2(x_i)$.

$$f_2(x_i) = (f_1(x_i)) + \alpha \boldsymbol{CART_2}$$
$$= (\boldsymbol{CART_{Init}} + \alpha \boldsymbol{CART_1}) + \alpha \boldsymbol{CART_2}$$

Least Squares Loss Function:

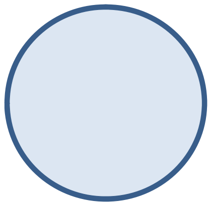
$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

We use the tree to update the model:
for the least squares loss function, the updates are the target variable average for each terminal node.
Denote the updates for iteration 2 by $\boldsymbol{CART_2}$

Gradient Boosting :Iteration 3 (Least Squares Loss)

Current Model: $f_2(x_i) = \boldsymbol{CART_{Init}} + \alpha \boldsymbol{CART_1} + \alpha \boldsymbol{CART_2}$

1. Draw a random sample S from the records in the learn data



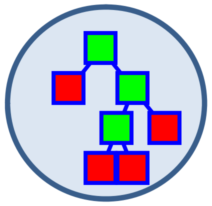
2. Compute the generalized residual $\tilde{y}_i = y_i - f_2(x_i)$ for the records in S

$\tilde{y}_i$	x_1	x_3
-.3	2	12
-2	5	1
.3	7	7

Least Squares Loss Function:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

3. Using $\tilde{y}_i$ as the target variable, fit a small, unpruned CART tree with J terminal nodes using the records in S



4. Multiply the predictions from Step 3 by the learning rate α = .01 and add them to the current model $f_2(x_i)$ to obtain an updated model $f_3(x_i)$.

$$f_3(x_i) = (f_2(x_i)) + \alpha \boldsymbol{CART_3}$$

$$= (\boldsymbol{CART_{Init}} + \alpha \boldsymbol{CART_1} + \alpha \boldsymbol{CART_2}) + \alpha \boldsymbol{CART_3}$$

We use the tree to update the model:
for the least squares loss function, the updates are the target variable average for each terminal node.
Denote the updates for iteration 3 by $\boldsymbol{CART_3}$

Gradient Boosting: Last Iteration (Least Squares Loss)

Note: there are a total of M iterations (in practice, the user specifies the M)

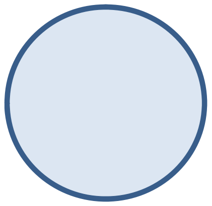
Current Model: $f_{M-1}(x_i) = \boldsymbol{CART}_{\boldsymbol{Init}} + \alpha \boldsymbol{CART}_{\boldsymbol{1}} + \cdots + \alpha \boldsymbol{CART}_{\boldsymbol{M-1}}$

1. Draw a random sample S from the records in the learn data

2. Compute the generalized residual $\tilde{y}_i = y_i - f_{M-1}(x_i)$ for the records in S

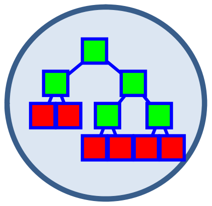
3. Using $\tilde{y}_i$ as the target variable, fit a small, unpruned CART tree with J terminal nodes using the records in S

4. Multiply the predictions from Step 3 by the learning rate α and add them to the current model $f_{M-1}(x_i)$ to obtain an updated model $f_M(x_i)$.



Sample

$\tilde{y}_i$	x_1	x_2
-2	17	0
-3	8	0
3	6	0



Sample

FINAL Model:

$$f_M(x_i) = (f_{M-1}(x_i)) + \alpha \boldsymbol{CART}_{\boldsymbol{M}}$$

$$= (\boldsymbol{CART}_{\boldsymbol{Init}} + \alpha \boldsymbol{CART}_{\boldsymbol{1}} + \cdots + \alpha \boldsymbol{CART}_{\boldsymbol{M-1}}) + \alpha \boldsymbol{CART}_{\boldsymbol{M}}$$

Least Squares Loss Function:

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

We use the tree to update the model:
for the least squares loss function, the updates are the target variable average for each terminal node.
Denote the updates for iteration M by $\boldsymbol{CART}_{\boldsymbol{M}}$

Gradient Boosting: Predicting New Records

This is our final model with **M** iterations:

$$f_M(x_i) = CART_{Init} + \alpha CART_1 + \cdots + \alpha CART_{M-1} + \alpha CART_M$$

Note: M iterations = M CART trees

Word Choice: Depending on the software, you will see either "number of trees" or "number of iterations" (for TreeNet software we use "number of trees")

Predict a New Record:

1. Run the record down each tree each time computing a prediction

   42 0.8 … -0.3 -0.2

2. Multiply each value, except the initial value (orange), by the learning rate

   42 α(0.8) … α(-0.3) α(-0.2)

3. **Final Prediction:** Add the values from Step 2. For this example let α=.01

   42 + .01(0.8)+ … +.01(-0.3)+.01(-0.2)

CART and Gradient Boosting

Because the foundation of gradient boosting is CART, gradient boosted trees automatically handle the following:

a) Variable selection
b) Variable interaction modeling
c) Local effect modeling
d) Nonlinear relationship modeling
e) Missing values (handled differently than in CART)
f) Outliers

Also, boosted trees are not affected by monotonic transformations of variables

This means that you can build gradient boosted trees with dirty data

Gradient Boosting in General

We just saw the gradient boosting algorithm using the least squares loss function, but gradient boosting is more general in nature and can handle a variety of loss functions

Note: The choice of the loss function will change the formulas used for the initial value, generalized residuals, and model updates

Regression Loss Functions	Classification Loss Functions
Least Absolute Deviation (LAD)	Binomial Loss
Least Squares (LS)	Multinomial Loss
Huber-M	Differential Lift
Cox-Survival	
Gamma	
Negative Binomial	
Poisson	
Tweedie	

Gradient Boosting: Binary Classification

Make an initial prediction $f_0 = \frac{1}{2}\log\frac{p}{1-p}$; Let the learn rate be α

For m=1 to MaxTrees

1. Draw a random sample S from the LEARN data
2. Compute the generalized residual $\tilde{y}_i = \frac{2y_i}{1+e^{2y_i F_{m-1}(x_i)}}$ for the records in S
3. Fit a (small) CART tree with J terminal nodes to $\tilde{y}_i$ as the target
4. Determine the optimal response surface updates
   $$\frac{\Sigma \tilde{y}_i}{\Sigma|\tilde{y}_i|(2-\tilde{y}_i)}$$
5. Shrink the updates by α, called the learning rate, and apply the updates

Repeat 1-5 **M** times

$$f_M(x_i) = \boldsymbol{CART}_{\boldsymbol{Init}} + \alpha\boldsymbol{CART}_{\boldsymbol{1}} + \cdots + \alpha\boldsymbol{CART}_{\boldsymbol{M-1}} + \alpha\boldsymbol{CART}_{\boldsymbol{M}}$$

Gradient Boosting: Tuning Parameters

Parameters to consider in gradient boosting include

- **Number of Trees = Number of Iterations** **(See Appendix)**
- Learning Rate **(See Appendix)**
- Subsampling Fraction **(See Appendix)**
- Tree Depth (Maximum number of terminal nodes per tree) **(See Appendix)**

Choosing an optimal parameter is dependent on the particular dataset

Interpreting Boosted Trees

There are a variety of ways to interpret gradient boosted trees **(see Appendix)**

- Partial Dependency Plots
- Interactions Statistics
- Variable Importance Measures

Can we have an interpretation that is similar to regression?

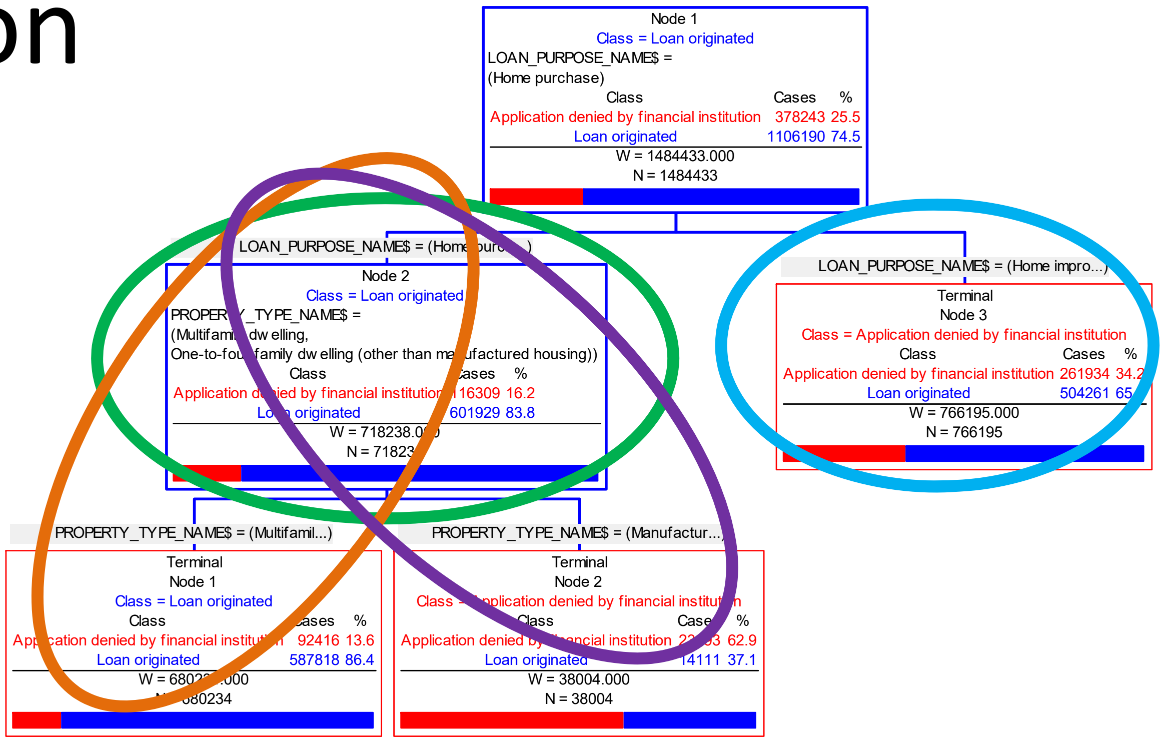
Bridging Gradient Boosting and Regression

Main Idea: use the rules from each tree in a gradient boosted tree model as predictors in a regularized regression model (i.e. LASSO, Elastic Net, etc.) as described in “Predictive Learning via Rule Ensembles” by Jerome Friedman and Bogdan Popescu (2005).

RuleLearner™ Software is a commercial implementation of the technique described in the paper *Predictive Learning via Rule Ensembles* and was developed in close consultation with Jerome Friedman.

Uses: Classification, Regression, and Survival Analysis (i.e. Cox loss function)

Rule Extraction



Node 1
Class = Loan originated
LOAN_PURPOSE_NAME$ = (Home purchase)
Class Cases %
Application denied by financial institution 378243 25.5
Loan originated 1106190 74.5
W = 1484433.000
N = 1484433
LOAN_PURPOSE_NAME$ = (Home purc...)
Node 2
Class = Loan originated
PROPERTY_TYPE_NAME$ = (Multifamily dwelling, One-to-four family dwelling (other than manufactured housing))
Application denied by financial institution 116309 16.2
Loan originated 601929 83.8
W = 718238.000
N = 718238
LOAN_PURPOSE_NAME$ = (Home impro...)
Terminal Node 3
Class = Application denied by financial institution
Application denied by financial institution 261934 34.2
Loan originated 504261 65.8
W = 766195.000
N = 766195
PROPERTY_TYPE_NAME$ = (Multifamil...)
Terminal Node 1
Class = Loan originated
Application denied by financial institution 92416 13.6
Loan originated 587818 86.4
W = 680234.000
N = 680234
PROPERTY_TYPE_NAME$ = (Manufactur...)
Terminal Node 2
Class = Application denied by financial institution
Application denied by financial institution 23893 62.9
Loan originated 14111 37.1
W = 38004.000
N = 38004

Rule 1: I(Loan_Purpose_Name$∈{“Home purchase”})

Rule 2: I(Loan_Purpose_Name$∈{“Home purchase”})∗I(Property_Type_Name$∈{Multifamily dweling, One−to−four family dweling})

Rule 3: I(Loan_Purpose_Name$ ∈{“Home purchase”})*I(Property_Type_Name$∈{“Manufactured Housing”})

Rule 4: I(Loan_Purchase_Name$ ∈ {“Home improvement”, “Refinancing”}

Using RuleLearner

RuleLearner can be used for a variety of tasks

1. Build an interpretable regression model with powerful predictive performance

2. Use RuleLearner to discover rules that correspond to high or low values of the target variable (similar to the goal of the PRIM method)

3. Discover local interactions between variables and use this to improve your regression models

RuleLearner

1. **Fit a gradient boosted tree using TreeNet**
   - **The size of the trees are randomly varied**

2. Extract the rules from each tree
   - Internally, each rule is a binary indicator (1 for if the observation satisfies the rule and 0 if it does not)

3. Use the rules as predictors in a regularized regression model (i.e. lasso, elastic net, ridge, etc.)
   - The rules are not standardized
   - The fast algorithm for computing the solution path is called Generalized Path Seeker (GPS)

Rule1	Rule2	Rule3	Rule4	…
1	0	1	0	…
1	0	0	0	…
1	0	1	1	…
0	0	0	1	…
…	…	…	…	

Introducing Randomness into the Model

Sometimes it can be beneficial to introduce randomness into ensemble models (Friedman and Popescu, 2003)

- **Random Forest:** choose the optimal split point and splitting variable among a random subset of variables for each split in each tree in the forest; also the trees are grown on bootstrap samples of the original data

- **Gradient Boosting:** randomly subsample the records during each iteration (you can also employ the random variable subset strategy)

For the construction of rule ensembles, a method suggested by Friedman and Popescu (2005) was to randomly vary the tree sizes during each iteration of the boosted tree model

- In TreeNet we randomly vary the tree sizes according to a Poisson distribution where the mean is equal to the maximum number of terminal nodes for the trees

RuleLearner

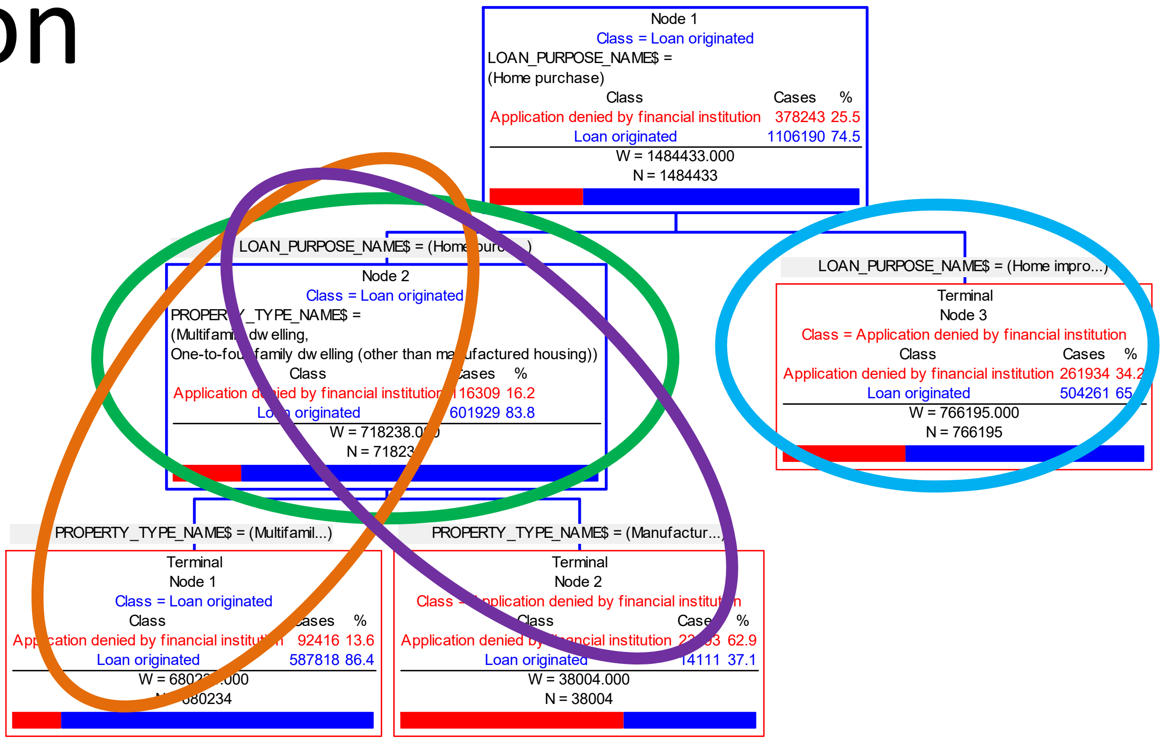
1. Fit a gradient boosted tree using TreeNet
   - The size of the trees are randomly varied

2. **Extract the rules from each tree**
   - **Internally, each rule is a binary indicator (1 for if the observation satisfies the rule and 0 if it does not)**

3. Use the rules as predictors in a regularized regression model (i.e. lasso, elastic net, ridge, etc.)
   - The rules are not standardized
   - The fast algorithm for computing the solution path is called Generalized Path Seeker (GPS)

Rule1	Rule2	Rule3	Rule4	…
1	0	1	0	…
1	0	0	0	…
1	0	1	1	…
0	0	0	1	…
…	…	…	…	

Rule Extraction



Node 1
Class = Loan originated
LOAN_PURPOSE_NAME$ = (Home purchase)
Class Cases %
Application denied by financial institution 378243 25.5
Loan originated 1106190 74.5
W = 1484433.000
N = 1484433
LOAN_PURPOSE_NAME$ = (Home purc...)
Node 2
Class = Loan originated
PROPERTY_TYPE_NAME$ = (Multifamily dwelling, One-to-four family dwelling (other than manufactured housing))
Application denied by financial institution 116309 16.2
Loan originated 601929 83.8
W = 718238.000
N = 718238
LOAN_PURPOSE_NAME$ = (Home impro...)
Terminal Node 3
Class = Application denied by financial institution
Application denied by financial institution 261934 34.2
Loan originated 504261 65.8
W = 766195.000
N = 766195
PROPERTY_TYPE_NAME$ = (Multifamil...)
Terminal Node 1
Class = Loan originated
Application denied by financial institution 92416 13.6
Loan originated 587818 86.4
W = 680234.000
N = 680234
PROPERTY_TYPE_NAME$ = (Manufactur...)
Terminal Node 2
Class = Application denied by financial institution
Application denied by financial institution 23893 62.9
Loan originated 14111 37.1
W = 38004.000
N = 38004

Rule 1: I(Loan_Purpose_Name$∈{"Home purchase"})

Rule 2: I(Loan_Purpose_Name$∈{"Home purchase"})∗I(Property_Type_Name$∈{Multifamily dweling, One−to−four family dweling})

Rule 3: I(Loan_Purpose_Name$ ∈{"Home purchase"})*I(Property_Type_Name$∈{"Manufactured Housing"})

Rule 4: I(Loan Purchase Name$ ∈ {"Home improvement", "Refinancing"}

RuleLearner

1. Fit a gradient boosted tree using TreeNet
   - The size of the trees are randomly varied

2. Extract the rules from each tree
   - Internally, each rule is a binary indicator (1 for if the observation satisfies the rule and 0 if it does not)

3. **Use the rules as predictors in a regularized regression model (i.e. lasso, elastic net, ridge, etc.)**
   - The rules are not standardized
   - The fast algorithm for computing the solution path is called Generalized Path Seeker (GPS)

Rule1	Rule2	Rule3	Rule4	…
1	0	1	0	…
1	0	0	0	…
1	0	1	1	…
0	0	0	1	…
…	…	…	…	

LASSO and Elastic Net

Least **A**bsolute **S**hrinkage **S**election **O**perator (Tibshirani 1996)

$$\min_{\beta} \|Y - XB\|^2$$

$$\text{s.t. } \|B\|_1 \leq \text{t}$$

Elastic Net (Zou and Hastie, 2004)

$$\min_{\beta} \|Y - XB\|^2$$

$$\text{s.t. } \alpha\|B\|_2{}^2 + (1-\alpha)\ \|B\|_1 \leq\ \text{t}$$

Elastic Net Note: there is a rescaling of the estimates

LASSO Advantage: zero out many of the coefficients to produce a "sparse" model

Elastic Net Advantage: zero out many of the coefficients (i.e. a sparse model) and select groups of correlated variables instead of only one from the group (the lasso which has a tendency to selected only one variable from the group)

RuleLearner

1. Fit a gradient boosted tree using TreeNet
   - The size of the trees are randomly varied

2. Extract the rules from each tree
   - Internally, each rule is a binary indicator (1 for if the observation satisfies the rule and 0 if it does not)

3. Use the rules as predictors in a regularized regression model (i.e. lasso, elastic net, ridge, etc.)
   - **The rules are not standardized (linear predictors are however)**
   - The fast algorithm for computing the solution path is called Generalized Path Seeker (GPS)

Rule1	Rule2	Rule3	Rule4	…
1	0	1	0	…
1	0	0	0	…
1	0	1	1	…
0	0	0	1	…
…	…	…	…	

Rules are NOT standardized

The rules are not standardized prior to fitting

This is because rules with larger standard deviations are penalized more than those with smaller standard deviations

More Information: Section 3.2 of Friedman and Popescu (2005)

RuleLearner and Linear Predictors

Since a RuleLearner model is a regression model linear terms can of course be added to the model

In fact this is encouraged and can promote additional sparsity because a single linear term can account for more of the model variance as opposed to multiple rules

Using raw predictors requires that missing values are imputed otherwise record deletion will occur

Standardizing Linear Predictors

Linear predictors are standardized using the following:

$$\frac{.4 l_j(x_j)}{std(x_j)}$$

The .4 is used so that the linear terms have the same prior influence as a typical rule (Section 5, Friedman and Popescu 2005). **0.4 is the average standard deviation for a rule.**

$$E(t_k) = \int_0^1 \sqrt{s_k(1-s_k)}\,ds_k = .3926$$

$$where\ s_k \sim Uni(0,1)$$

RuleLearner

1. Fit a gradient boosted tree using TreeNet
   - The size of the trees are randomly varied
2. Extract the rules from each tree
   - Internally, each rule is a binary indicator (1 for if the observation satisfies the rule and 0 if it does not)
3. Use the rules as predictors in a regularized regression model (i.e. lasso, elastic net, ridge, etc.)
   - The rules are not standardized (linear predictors are however)
   - **The fast algorithm for computing the solution path is called Generalized Path Seeker (GPS)**

Rule1	Rule2	Rule3	Rule4	…
1	0	1	0	…
1	0	0	0	…
1	0	1	1	…
0	0	0	1	…
…	…	…	…	

Generalized Path Seeker

Since we can have potentially thousands of predictors (i.e. rules) a fast algorithm is required for computing the solution paths for the regression models. We use the Generalized Path Seeker algorithm (Friedman, 2008)

Uses an *Elasticity Parameter* in the range of [0,2]

- **Forward Stepwise:** elasticity of 0
- **LASSO:** elasticity of 1
- **Ridge Regression:** elasticity of 2
- **Elastic Net:** elasticity between 1 and 2

GPS® Software is a commercial implementation of the Generalized Path Seeker algorithm (Friedman, 2008). GPS was developed in close consultation with Jerome Friedman.

Example: Loan Data

Question: Can we predict if a person's loan application will be accepted or denied?

Source: Consumer Financial Protection Bureau (Publicly available)

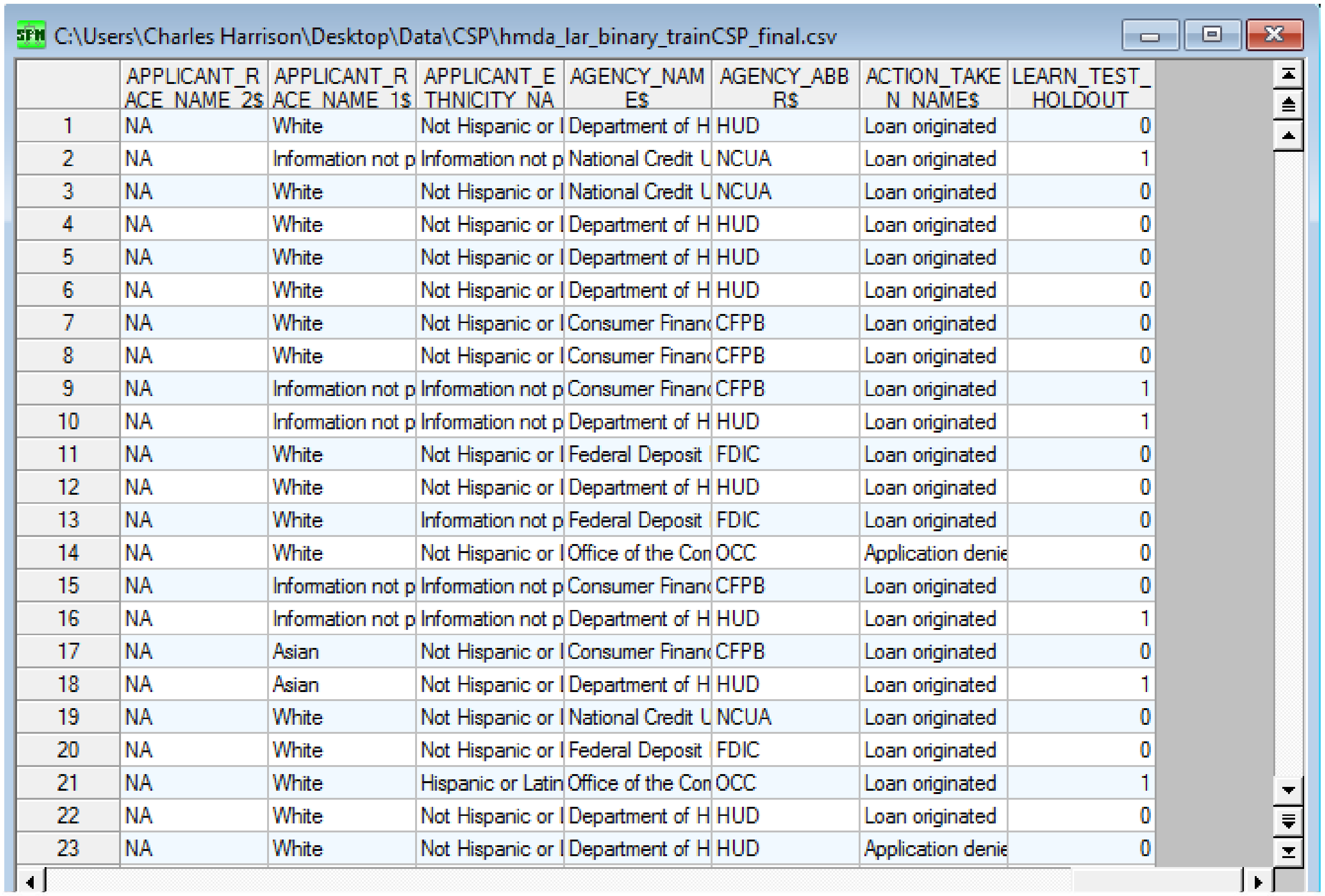
Description: The Home Mortgage Disclosure Act (HMDA), enacted by Congress in 1975, requires many financial institutions to maintain, report, and publicly disclose information about mortgages.

Binary Target Variable: Loan Originated vs. Denied

Predictor Variables: 46

Partition	Records
LEARN	456,632
TEST	113,634

Loan Data



C:\Users\Charles Harrison\Desktop\Data\CSP\hmda_lar_binary_trainCSP_final.csv

	APPLICANT_RACE_NAME_2$	APPLICANT_RACE_NAME_1$	APPLICANT_ETHNICITY_NA	AGENCY_NAMES$	AGENCY_ABBR$	ACTION_TAKEN_NAMES$	LEARN_TEST_HOLDOUT
1	NA	White	Not Hispanic or I	Department of H	HUD	Loan originated	0
2	NA	Information not p	Information not p	National Credit U	NCUA	Loan originated	1
3	NA	White	Not Hispanic or I	National Credit U	NCUA	Loan originated	0
4	NA	White	Not Hispanic or I	Department of H	HUD	Loan originated	0
5	NA	White	Not Hispanic or I	Department of H	HUD	Loan originated	0
6	NA	White	Not Hispanic or I	Department of H	HUD	Loan originated	0
7	NA	White	Not Hispanic or I	Consumer Financ	CFPB	Loan originated	0
8	NA	White	Not Hispanic or I	Consumer Financ	CFPB	Loan originated	0
9	NA	Information not p	Information not p	Consumer Financ	CFPB	Loan originated	1
10	NA	Information not p	Information not p	Department of H	HUD	Loan originated	1
11	NA	White	Not Hispanic or I	Federal Deposit	FDIC	Loan originated	0
12	NA	White	Not Hispanic or I	Department of H	HUD	Loan originated	0
13	NA	White	Information not p	Federal Deposit	FDIC	Loan originated	0
14	NA	White	Not Hispanic or I	Office of the Con	OCC	Application denie	0
15	NA	Information not p	Information not p	Consumer Financ	CFPB	Loan originated	0
16	NA	Information not p	Information not p	Department of H	HUD	Loan originated	1
17	NA	Asian	Not Hispanic or I	Consumer Financ	CFPB	Loan originated	0
18	NA	Asian	Not Hispanic or I	Department of H	HUD	Loan originated	1
19	NA	White	Not Hispanic or I	National Credit U	NCUA	Loan originated	0
20	NA	White	Not Hispanic or I	Federal Deposit	FDIC	Loan originated	0
21	NA	White	Hispanic or Latin	Office of the Con	OCC	Loan originated	1
22	NA	White	Not Hispanic or I	Department of H	HUD	Loan originated	0
23	NA	White	Not Hispanic or I	Department of H	HUD	Application denie	0

Optimizing the Boosted Tree

A hyperparameter search was performed. The number of trees was fixed to 1000 so that the models had enough iterations to converge.

The search grid was defined by the following values

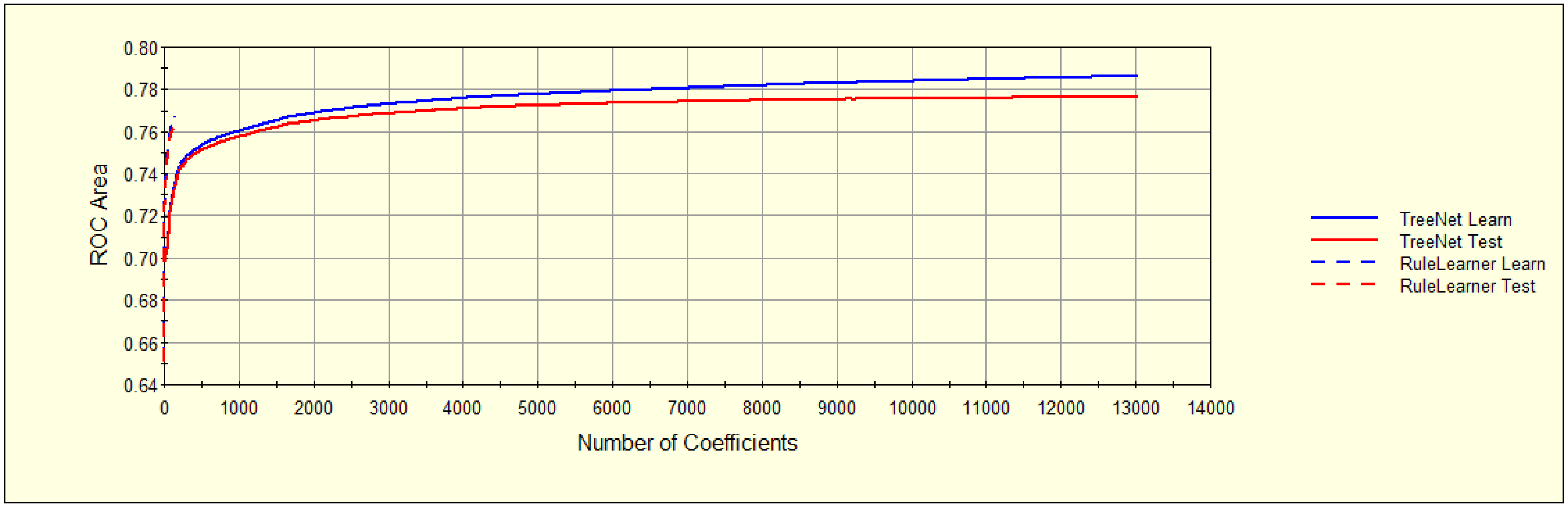
- Random Subsample Size: 10%, 30%, and 50%
- Learning Rate: 0.1 and 0.01
- Number of Terminal Nodes: 2,3,4,5,6,9,12

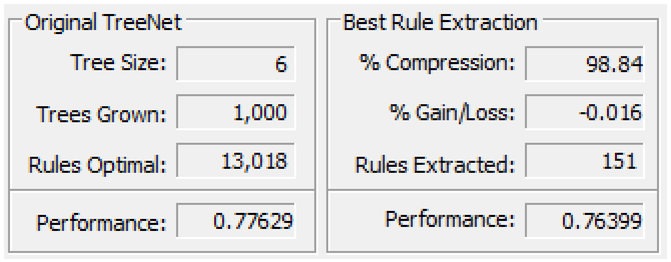
For each of the 3*2*7 = 42 combinations of parameters, 20 models were constructed with the learning and test partitions determined randomly each time

The optimal set of parameters were determined by examining the mean error rate values as well as the values for the standard deviation of the error rates. Based on this search, the optimal set of parameter values was

- Learning Rate: 0.1
- Random Subsample Size: 50%
- Number of Terminal Nodes: 6 (Note: choosing 4 or 5 nodes gives you around the same error and 9 or 12 nodes gives a negligible improvement)

TreeNet Gradient Boosting vs. RuleLearner (ROC Area)

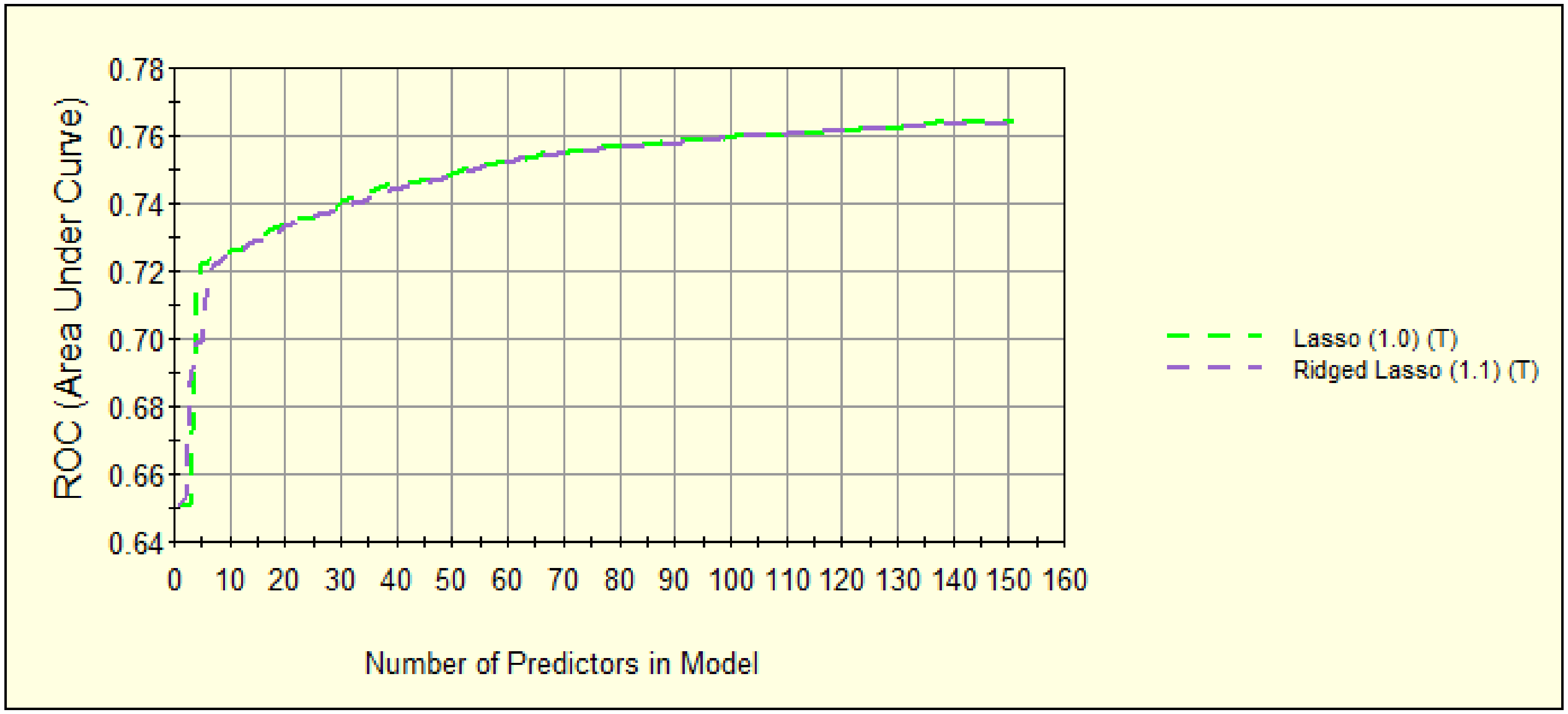


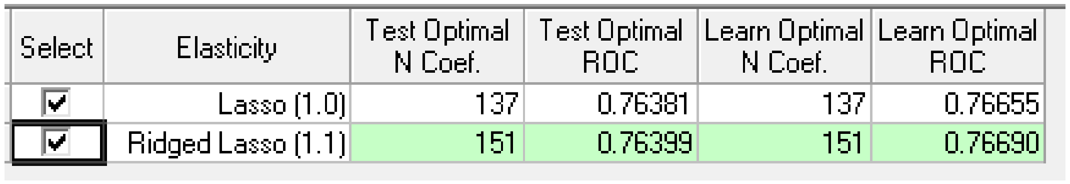


Original TreeNet		Best Rule Extraction	
Tree Size:	6	% Compression:	98.84
Trees Grown:	1,000	% Gain/Loss:	-0.016
Rules Optimal:	13,018	Rules Extracted:	151
Performance:	0.77629	Performance:	0.76399

The optimal boosted tree is composed of 13,018 total rules whereas the optimal RuleLearner model has only 151 rules and approximately the same performance

Elastic Net vs. LASSO

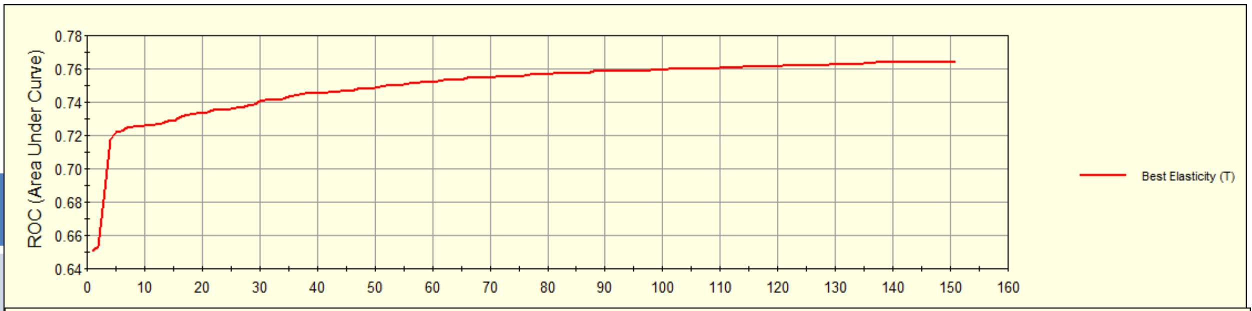




Select	Elasticity	Test Optimal N Coef.	Test Optimal ROC	Learn Optimal N Coef.	Learn Optimal ROC
✓	Lasso (1.0)	137	0.76381	137	0.76655
✓	Ridged Lasso (1.1)	151	0.76399	151	0.76690

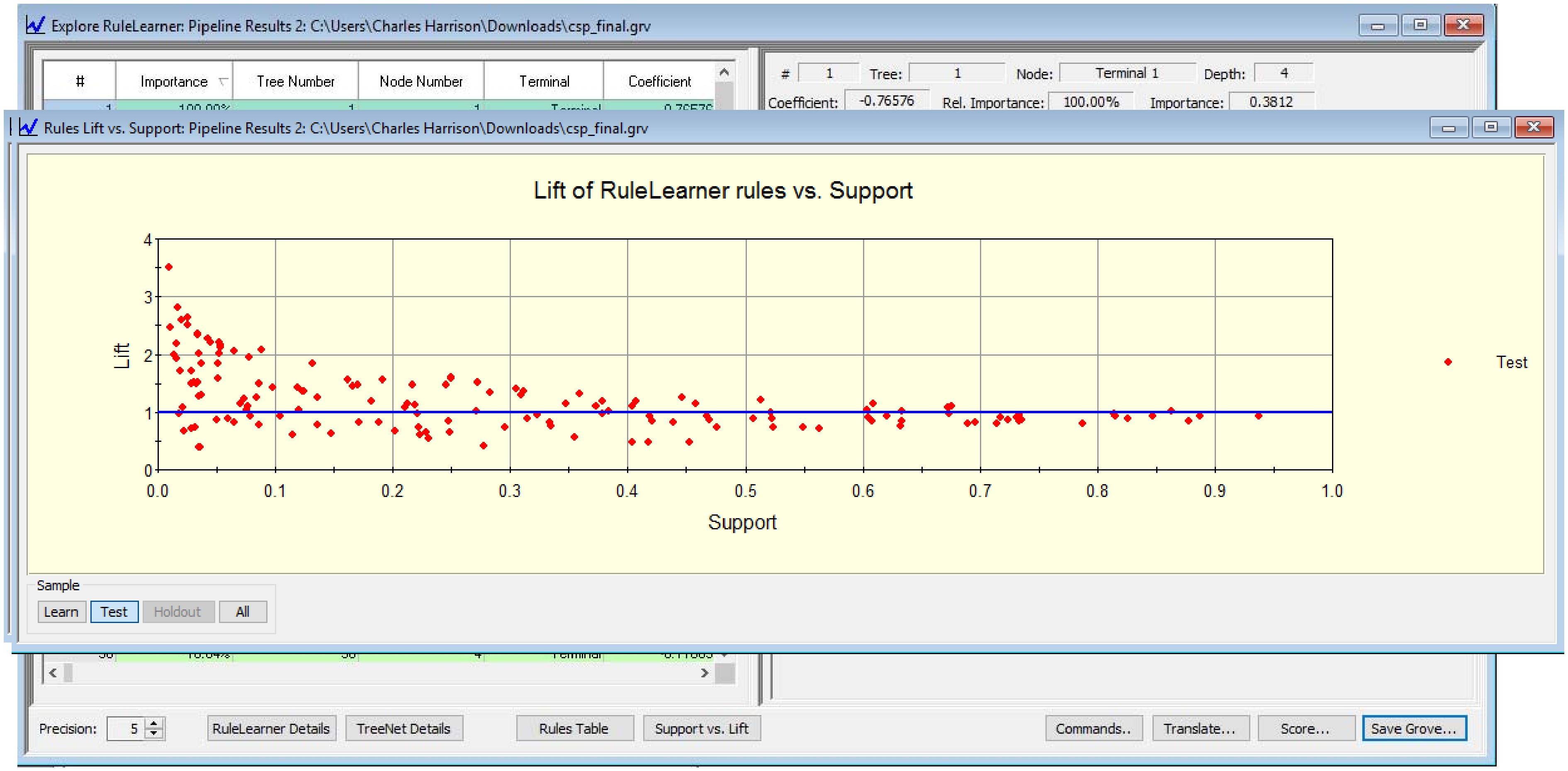
The best LASSO model is more sparse than the Elastic Net, but the Elastic Net's performance is slightly better than the LASSO. The Elastic Net was chosen due to its performance and ability to selected correlated predictors.

RuleLearner®: Performance



	Rul
Best Rule	1

RuleLearner Output



Explore RuleLearner: Pipeline Results 2: C:\Users\Charles Harrison\Downloads\csp_final.grv

#	Importance	Tree Number	Node Number	Terminal	Coefficient

#	1	Tree:	1	Node:	Terminal 1	Depth:	4
Coefficient:	-0.76576	Rel. Importance:	100.00%	Importance:	0.3812		

Rules Lift vs. Support: Pipeline Results 2: C:\Users\Charles Harrison\Downloads\csp_final.grv

Evaluating RuleLearner Models

Metrics to consider include

1. **Importance Measures**
2. Support of a rule
3. Coefficient associated with a rule

Importance Measures

Rule Importance:

Standard Deviation of a Rule

$$I_k = |\widehat{a_k}| \underset{\uparrow}{\sqrt{s_k(1-s_k)}}$$

Where s_k is the support for rule k and $\widehat{a_k}$ is the coefficient in the regression model

Linear Predictor Importance:

$$I_j = |\widehat{b_k}| std(l_j(x_j))$$

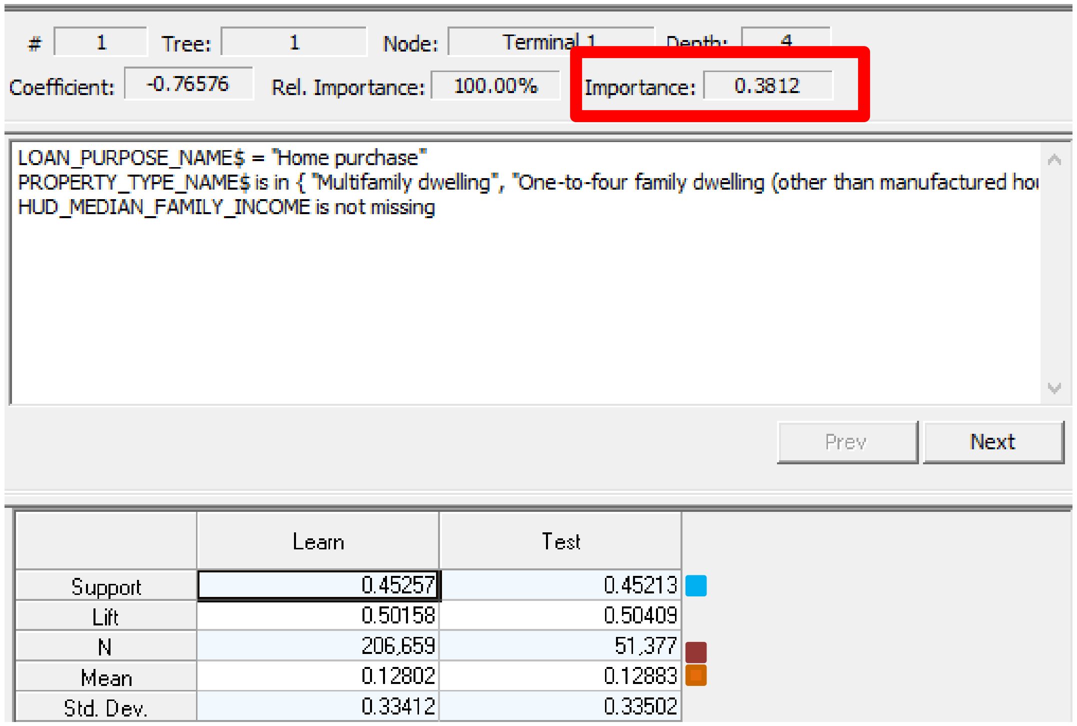
Where $\widehat{b_k}$ is the coefficient for the linear term $l_j(x_j)$ in the model and $std\left(l_j(x_j)\right)$ is the standard deviation of $l_j(x_j)$

Most Important Rule

$$I_1 = |\widehat{a_1}|\sqrt{s_1(1-s_1)} = |-.76576|\sqrt{.45257\,(1-.45257)} = .76576 * .49774 = \boxed{.3812}$$

If a person is applying for a **home purchase loan** **AND** the type of home is either a **multifamily dwelling** or **a one-to-four family dwelling (other than a mobile home)** **AND** the **HUD median family income is not missing** then the probability that a person's loan application is denied is **12.88%**

This rule applies to **45.2%** of the people in the testing data (**51,377** people)



#	1	Tree:	1	Node:	Terminal 1	Depth:	4
Coefficient:	-0.76576	Rel. Importance:	100.00%	Importance:	0.3812		

🟦 LOAN_PURPOSE_NAME$ = "Home purchase"
🟩 PROPERTY_TYPE_NAME$ is in { "Multifamily dwelling", "One-to-four family dwelling (other than manufactured hoı
🟪 HUD_MEDIAN_FAMILY_INCOME is not missing

	Learn	Test	
Support	0.45257	0.45213	🟦
Lift	0.50158	0.50409	
N	206,659	51,377	🟥
Mean	0.12802	0.12883	🟧
Std. Dev.	0.33412	0.33502	

Evaluating Rules: Support

Support- percentage of observations that satisfy a rule

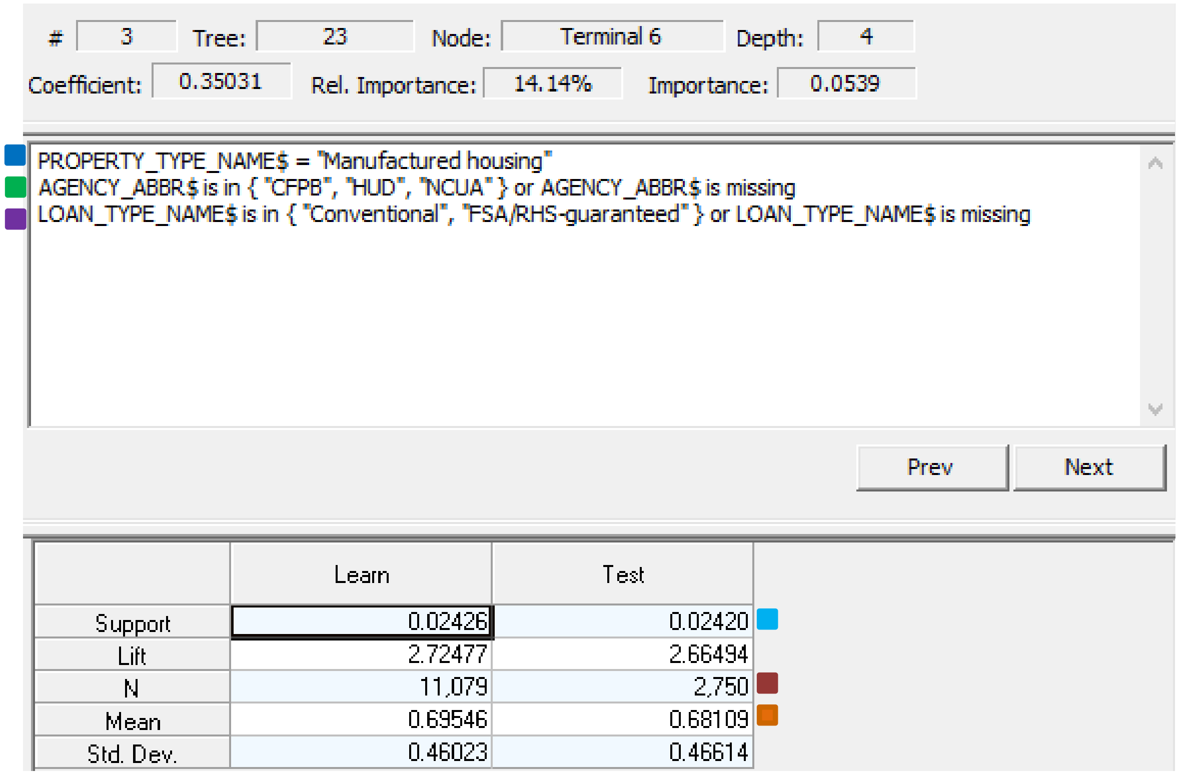
Rules with a small support (say 1-5%) may provide valuable insights into your data

Rules with a very small support (say .0001%) and thus may be less useful in practice because due to their rarity

14th Rule: Interpretation

If the property type is **manufactured housing (i.e. mobile home)** and the agency that the application was sent to was either the **CFPB, HUD, NCUA, or was missing** and the type of loan was either **conventional, FSA/RHS-guaranteed or was missing** then the probability that a person's loan application is denied is **68.11%**

This rule applies to around **2.42%** of the people in the testing data (**2,750** people)



#	3	Tree:	23	Node:	Terminal 6	Depth:	4
Coefficient:	0.35031	Rel. Importance:	14.14%	Importance:	0.0539		

```
🟦 PROPERTY_TYPE_NAME$ = "Manufactured housing"
🟩 AGENCY_ABBR$ is in { "CFPB", "HUD", "NCUA" } or AGENCY_ABBR$ is missing
🟪 LOAN_TYPE_NAME$ is in { "Conventional", "FSA/RHS-guaranteed" } or LOAN_TYPE_NAME$ is missing
```

	Learn	Test	
Support	0.02426	0.02420	🟦
Lift	2.72477	2.66494	
N	11,079	2,750	🟥
Mean	0.69546	0.68109	🟧
Std. Dev.	0.46023	0.46614	

Coefficient Interpretation

This is the 5 rule model:

$$\log\left(\frac{p}{1-p}\right) = .12 - .77R_1 - .26\,R_2 - .24R_3 - .16R_4 + .24R_5$$

If a person only satisfies Rule 1, then the odds of having a loan application denied decreases by (1-.463)*100% = 53.7% (because $e^{-.77} = e^{-.77}$=.463)

	Rules	ROC Area	Lift	Misclass. Rate	Neg. Avg. LL
Best Rule	1	.651	1.41	25.56%	.564
Top 5 Rules	5	.722	2.23	25.56%	.531
Optimal RuleLearner	126	.762	2.69	21.8%	.477
Optimal TreeNet	2,388	.768	2.76	21.4%	.473

Rule 1:

If a person is applying for a **home purchase loan** **AND** the type of home is either a **multifamily dwelling** or **a one-to-four family dwelling (other than a mobile home)** **AND** the **HUD median family income is not missing**

Coefficient Interpretation with Nested Rules

When interpreting the coefficients be sure to check whether a rule is nested inside of another

Rule A: W < 1,000

Rule B: X > 7 and Y = “M”

Rule C: X > 7 and Y = “M” and W < 1,000

You must take this into account when interpreting the coefficients for classification or regression models

Conclusion

RuleLearner is a regularized regression model that uses the rules from a gradient boosted tree as predictors

RuleLearner can be used for both classification and regression

Use RuleLearner to

1. Build powerful regression models that can be used in a highly regulated environment
2. Discover rules that correspond to high or low values of the target variable (similar to the PRIM method)
3. Discover local interactions (i.e. a rule) and then use this to improve your regression model

Thank you

Session Chair: Shankang Qu, PhD

Salford Systems Senior Scientist: Mikhail Golovnya

Salford Systems Marketing Associate: Dorene Paran

References

Classification and Regression Trees

Breiman, Friedman, Olshen, and Stone (1984)

Additive Logistic Regression: A Statistical View of Boosting

Friedman, Hastie, and Tibshirani (2000)

Greedy Function Approximation: A Gradient Boosting Machine

Friedman (2001)

Stochastic Gradient Boosting

Friedman (2001)

Regression Shrinkage and Selection via the LASSO

Tibshirani (1996)

Regularization and Variable Selection via the Elastic Net

Hastie and Zou (2005)

Fast Sparse Regression and Classification

Friedman (2008)

References

Patient Rule Induction Method (PRIM)

- Friedman (1998)
- Note: Not required to understand RuleLearner, but a great introduction to rule-based machine learning algorithms

Importance Sampled Learning Ensembles

- Friedman and Popescu (2003)

Predictive Learning via Rule Ensembles

- Friedman and Popescu (2005)
- ★ Primary reference for RuleLearner
- Note: Read “Importance Sampled Learning Ensembles” first

Additional Video Links

How to Write the Equation of a CART Tree

https://www.youtube.com/watch?v=JdpZT9I0G5M

How are Variable Interactions Modeled in Decision Trees (CART)?

https://www.youtube.com/watch?v=NgwJWWlDdwk

Questions?

Appendix

Some Gradient Boosting Parameters to Consider
- Number of Trees = Number of Iterations
- Learning Rate
- Subsampling Fraction
- Tree Depth (Maximum number of terminal nodes per tree)

Standardizing Linear Predictors in RuleLearner

Variable Importance Measures in Gradient Boosting

Partial Dependency Plots
- Construction Procedure
- Interpretation

CART Advantages

Tuning Parameters in a Gradient Boosting

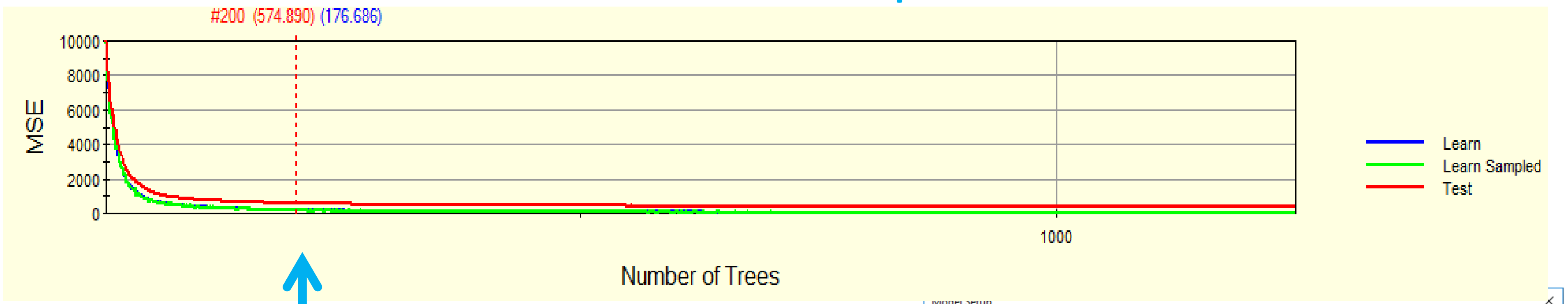
Parameters to consider in gradient boosting include

- **Number of Trees = Number of Iterations**
- Learning Rate
- Subsampling Fraction
- Tree Depth (Maximum number of terminal nodes per tree)

Choosing an optimal parameter is dependent on the particular dataset

Number of Trees (Iterations)

Here there is no material improvement in the error after 300

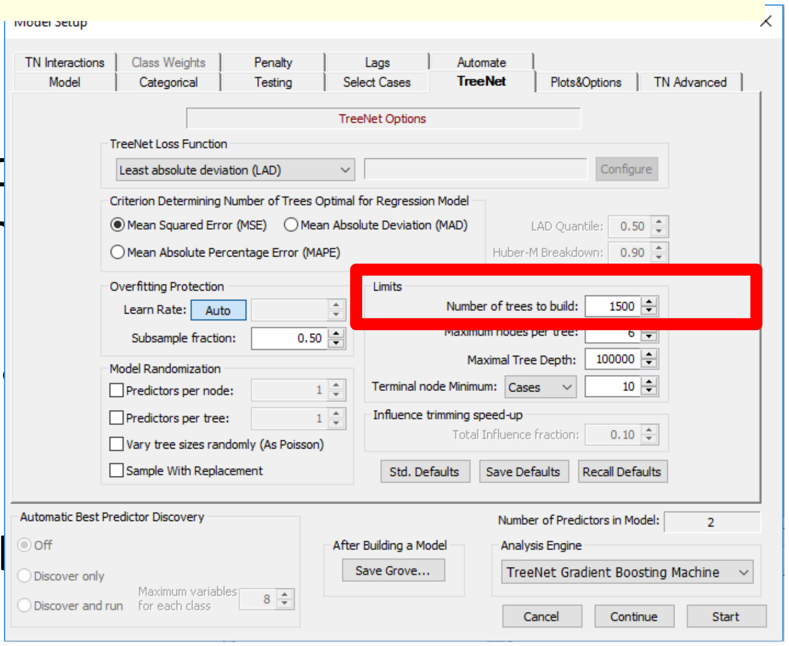


SPM Default: 200 Trees

The number of trees should be set high enc
error "converges" (converges means that th

Note: sometimes you will see that after a cert
error curve will start to increase.

The optimal number of trees will depend on tl



TreeNet Loss Function: Least absolute deviation (LAD)

Criterion Determining Number of Trees Optimal for Regression Model: [✓ Mean Squared Error (MSE)] Mean Absolute Deviation (MAD) Mean Absolute Percentage Error (MAPE)

LAD Quantile: 0.50
Huber-M Breakdown: 0.90

Overfitting Protection
Learn Rate: Auto
Subsample fraction: 0.50

Limits
Number of trees to build: 1500
Maximum nodes per tree: 6
Maximal Tree Depth: 100000
Terminal node Minimum: Cases 10

Model Randomization
Predictors per node: 1
Predictors per tree: 1
Vary tree sizes randomly (As Poisson)
Sample With Replacement

Influence trimming speed-up
Total Influence fraction: 0.10

Automatic Best Predictor Discovery
[✓ Off]
Discover only
Discover and run
Maximum variables for each class: 8

After Building a Model

Number of Predictors in Model: 2

Analysis Engine: TreeNet Gradient Boosting Machine

Tuning Parameters in a TreeNet Model

Parameters to consider in gradient boosting

- Number of Iterations = Number of Trees
- **Learning Rate**
- Subsampling Fraction
- Tree Depth (Maximum number of terminal nodes per tree)

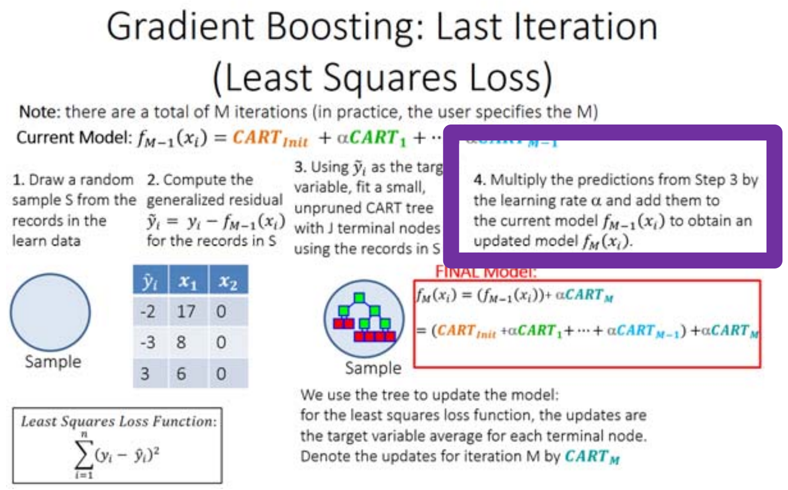
The optimal values for these parameters will be dependent on the particular dataset

Learning Rate

The *learning rate*, denoted by α, is the amount of shrinkage applied to each model update in the jth iteration:

$$f_j(x_i) = f_{j-1} + \alpha CART_j$$

Note: $0 < \alpha \leq 1$



Gradient Boosting: Last Iteration (Least Squares Loss)

Note: there are a total of M iterations (in practice, the user specifies the M)

Current Model: $f_{M-1}(x_i) = CART_{Init} + \alpha CART_1 + \cdots$

1. Draw a random sample S from the records in the learn data

2. Compute the generalized residual $\tilde{y}_i = y_i - f_{M-1}(x_i)$ for the records in S

3. Using $\tilde{y}_i$ as the targ variable, fit a small, unpruned CART tree with J terminal nodes using the records in S

4. Multiply the predictions from Step 3 by the learning rate α and add them to the current model $f_{M-1}(x_i)$ to obtain an updated model $f_M(x_i)$.

$\tilde{y}_i$	x_1	x_2
-2	17	0
-3	8	0
3	6	0

FINAL Model:

$$f_M(x_i) = (f_{M-1}(x_i)) + \alpha CART_M$$
$$= (CART_{Init} + \alpha CART_1 + \cdots + \alpha CART_{M-1}) + \alpha CART_M$$

Least Squares Loss Function: $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$

We use the tree to update the model: for the least squares loss function, the updates are the target variable average for each terminal node. Denote the updates for iteration M by $CART_M$

Choosing the optimal learn rate is accomplished through cross validation or via a testing sample

<u>The learning rate should be kept small; values like .001, .01, or .1 often give good results</u>, but smaller learning rates require more trees

Learning Rate and the Number of Iterations

In gradient boosting the learning rates are set to be smaller. Smaller learning rates require more trees (iterations) for the model error to converge (“converge” means that the test error curve will be flat, that is, no longer decrease)

Intuition: Imagine that you want to walk to the middle of a crater

- If you take small strides (i.e. smaller learning rate) then you will need more steps (i.e. more trees= more iterations) to get to the bottom of the crater (i.e. converge to the minimum error).
  - If you make a mistake (i.e. the model makes an error during one of the iterations) then the error is not as serious because your stride is smaller (i.e. it is easier for the model to recover in the next iteration)
- If you take larger strides (i.e. larger learning rate) then you will need less steps (i.e. less trees = less iterations) to get to the bottom of the crater (i.e. converge to the minimum error)
  - If you make a mistake (i.e. the model makes an error during one of the iterations) then the error is more serious because your stride is larger(i.e. it is not as easy for the model to recover as compared to the situation where the learning rate is smaller)

Gradient Boosting: Least Squares Loss

How does TreeNet model this curve? It makes small improvements (i.e. the learning rate is a small number that "**shrinks**" the model updates)

Tree 1

Tree 10

Tree 50

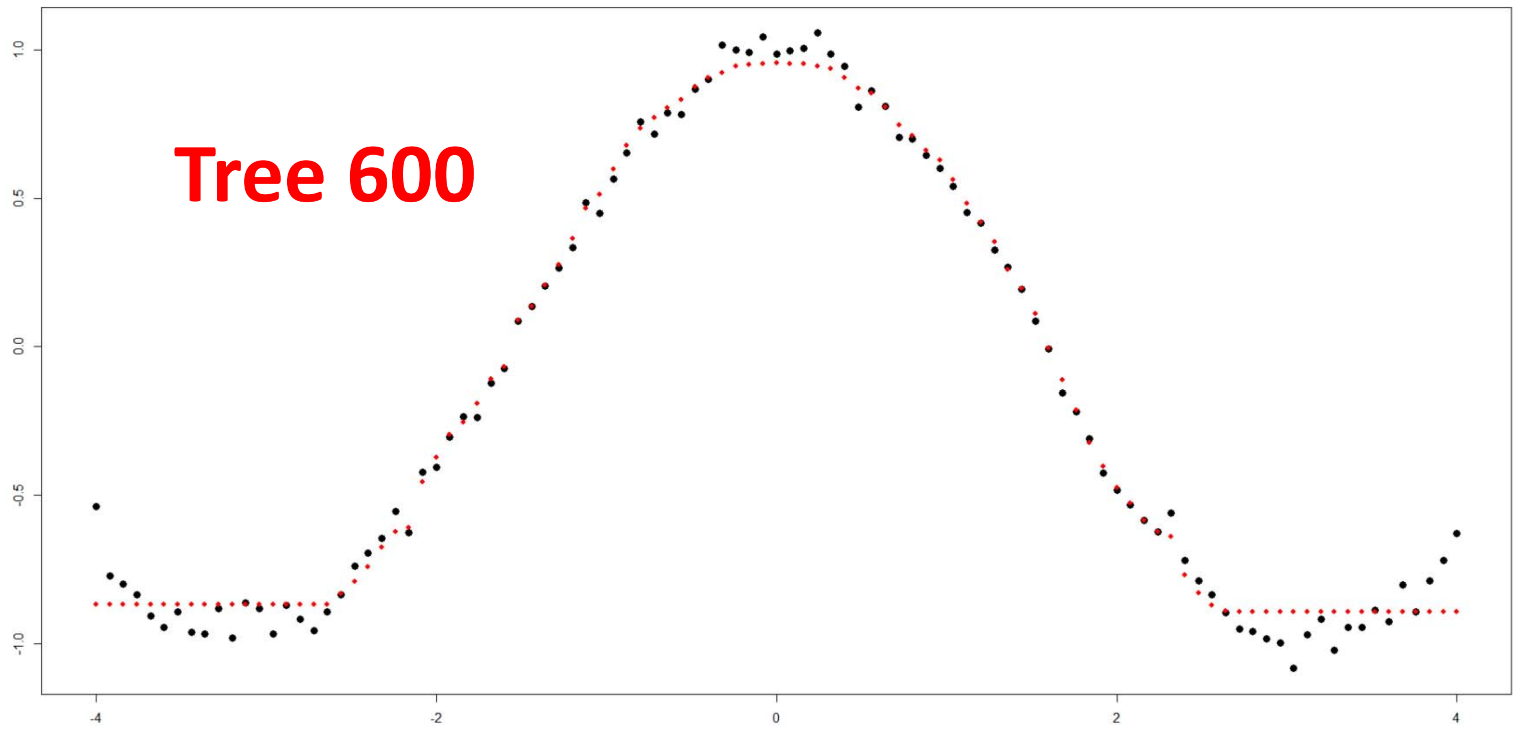
Tree 100

Tree 150

Tree 200

Tree 400

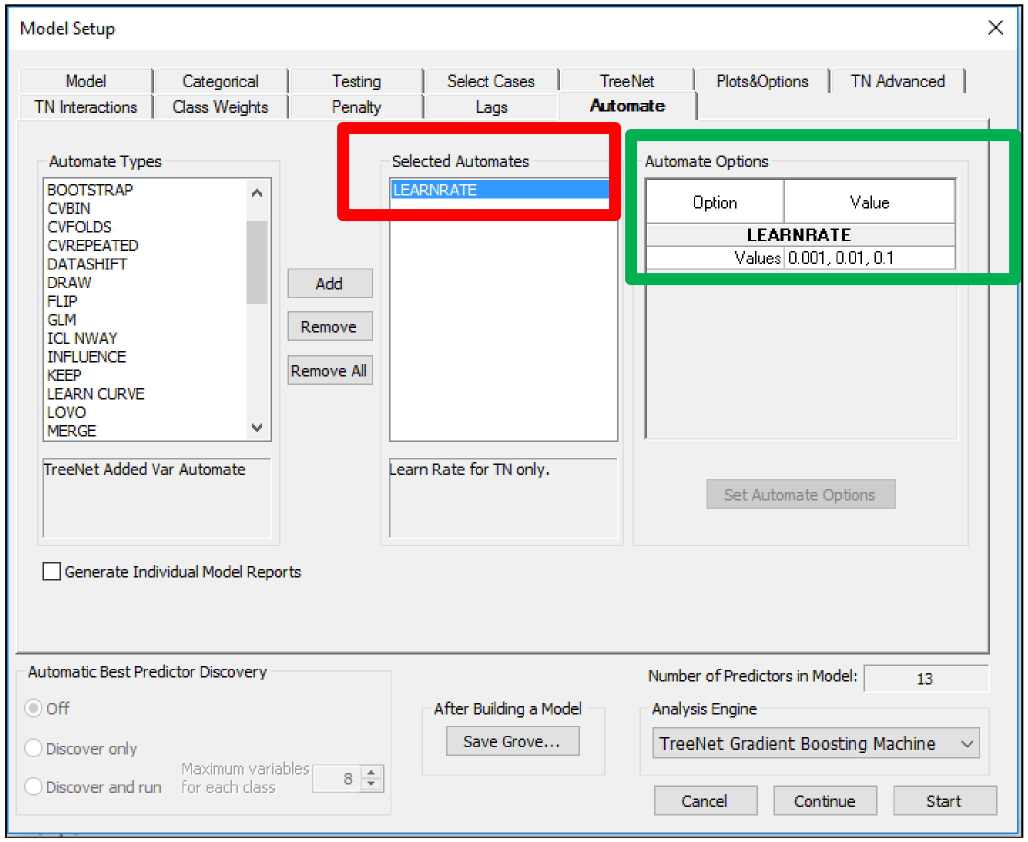
Tree 600



Note: Noise ~ N(0,1)

Automate LEARNRATE

If you want to experiment with different learning rates without manually refitting a model each time then use Automate LEARNRATE. Automate LEARNRATE automatically builds separate TreeNet models with a different learning rate each time (the other model settings are identical).



Option	Value
LEARNRATE	
Values	0.001, 0.01, 0.1

[ ] Generate Individual Model Reports

Automatic Best Predictor Discovery: [✓ Off] Discover only / Discover and run

Maximum variables for each class: 8

Number of Predictors in Model: 13

Analysis Engine: TreeNet Gradient Boosting Machine

Tuning Parameters in Gradient Boosting

Parameters to consider in gradient boosting

- Number of Trees = Number of Iterations
- Learning Rate
- **Subsampling Fraction**
- Tree Depth (Maximum number of terminal nodes per tree)

Choosing an optimal parameter is dependent on the particular dataset

Subsampling Fraction

Why take a random sample for each iteration?

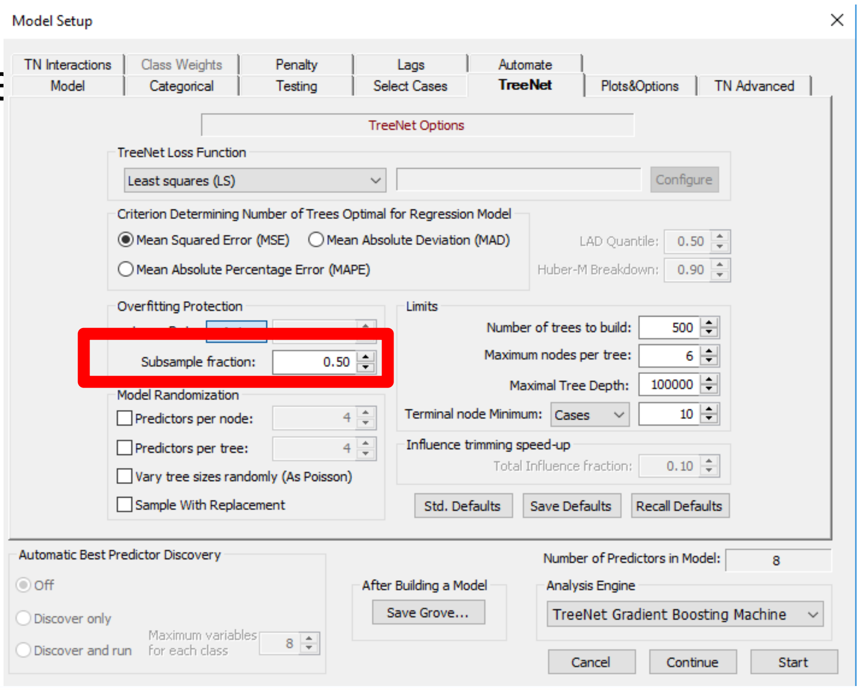
Taking a random sample at each iteration decreases the correlation between the trees at different iterations and this tends to reduce the overall prediction variance of the combined model (meaning the model improves; due to the bias-variance tradeoff).

How large should the random sample be

Answer: Depends on the dataset

Default Value: 50%.

Note: smaller percentages lead to faster model compute times because there is a smaller number of split points to evaluate



TreeNet Loss Function: Least squares (LS)

Criterion Determining Number of Trees Optimal for Regression Model: [✓ Mean Squared Error (MSE)] Mean Absolute Deviation (MAD) Mean Absolute Percentage Error (MAPE)

LAD Quantile: 0.50
Huber-M Breakdown: 0.90

Overfitting Protection
Subsample fraction: 0.50

Model Randomization
Predictors per node: 4
Predictors per tree: 4
Vary tree sizes randomly (As Poisson)
Sample With Replacement

Limits
Number of trees to build: 500
Maximum nodes per tree: 6
Maximal Tree Depth: 100000
Terminal node Minimum: Cases 10

Influence trimming speed-up
Total Influence fraction: 0.10

Automatic Best Predictor Discovery: [✓ Off] Discover only Discover and run
Maximum variables for each class: 8

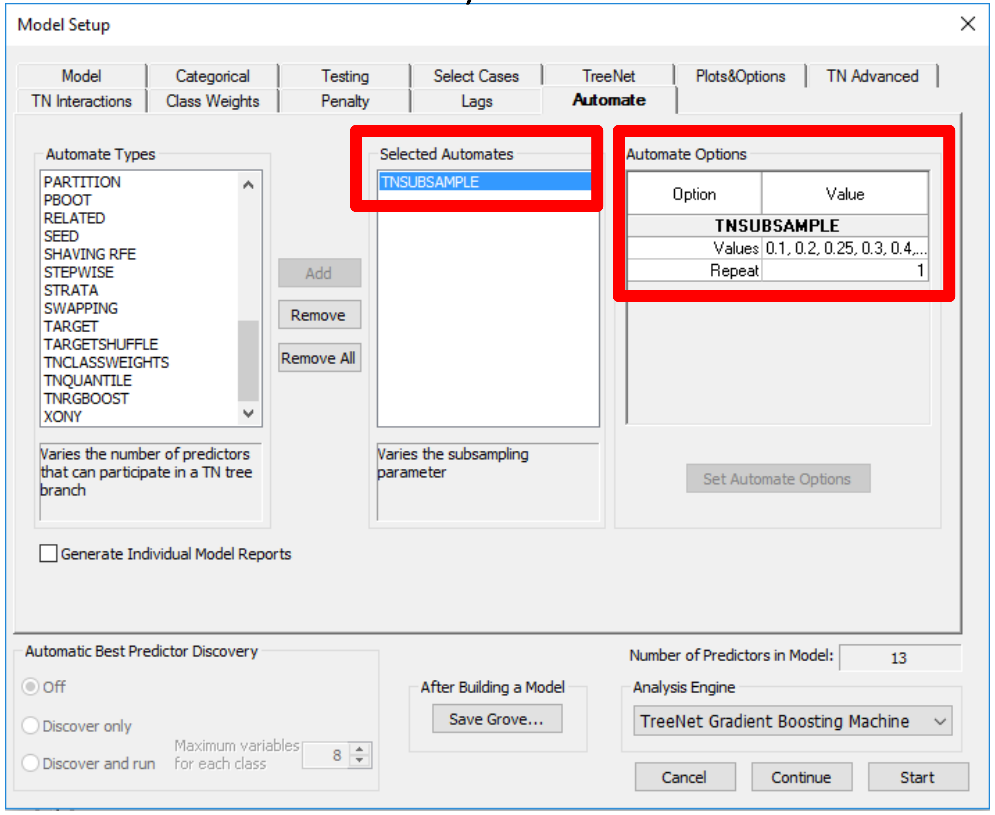
After Building a Model

Number of Predictors in Model: 8

Analysis Engine: TreeNet Gradient Boosting Machine

Automate TNSUBSAMPLE

The optimal subsampling fraction can be determined using Automate TNSUBSAMPLE which builds separate TreeNet models with a different subsampling fraction each time (the other model settings are identical each time)


Model Setup
Model | Categorical | Testing | Select Cases | TreeNet | Plots&Options | TN Advanced
TN Interactions | Class Weights | Penalty | Lags | Automate
Add
Remove
Remove All
Set Automate Options
Save Grove...
Cancel
Continue
Start

Automate Types: PARTITION, PBOOT, RELATED, SEED, SHAVING RFE, STEPWISE, STRATA, SWAPPING, TARGET, TARGETSHUFFLE, TNCLASSWEIGHTS, TNQUANTILE, TNRGBOOST, XONY

Varies the number of predictors that can participate in a TN tree branch

Selected Automates: TNSUBSAMPLE

Varies the subsampling parameter

Automate Options

Option	Value
TNSUBSAMPLE	
Values	0.1, 0.2, 0.25, 0.3, 0.4,...
Repeat	1

Generate Individual Model Reports

Automatic Best Predictor Discovery: Off / Discover only / Discover and run — Maximum variables for each class: 8

After Building a Model

Number of Predictors in Model: 13

Analysis Engine: TreeNet Gradient Boosting Machine

Gradient Boosting: Tuning Parameters

Parameters to consider in gradient boosting

- Learning Rate
- Number of Trees
- Subsampling Fraction
- **Tree Depth (Maximum number of terminal nodes per tree)**

Choosing an optimal parameter is dependent on the particular dataset

Tree Depth (Maximum terminal nodes per tree)

A CART tree with J-terminal nodes can capture up to (J-1)-way interactions

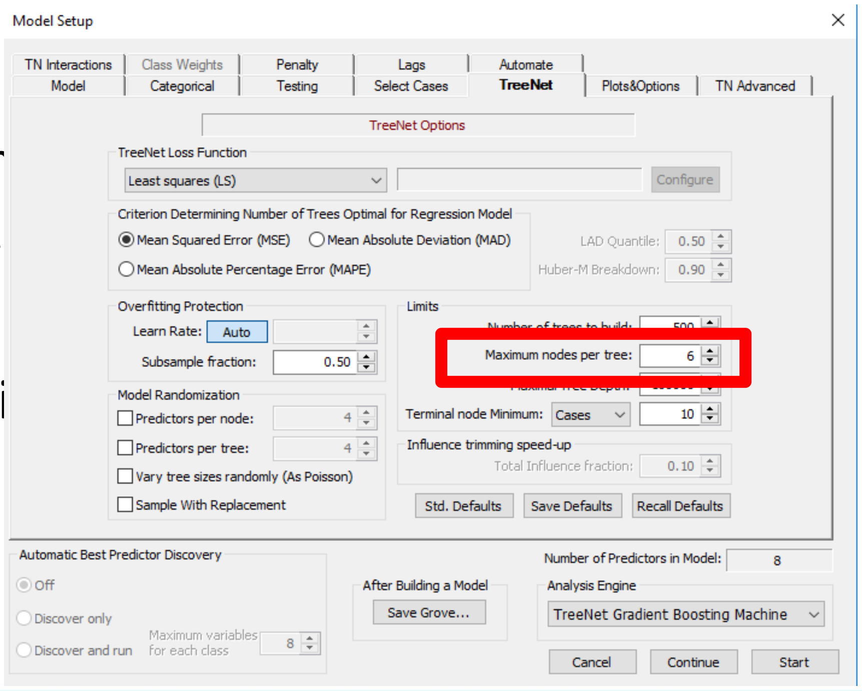
In gradient boosting, the user sets the maximum number of terminal nodes for each CART tree

Example: The maximum number of terr
by default. Thus, using default settings,
TreeNet can capture up to 6-1 = 5-way i

A more detailed explanation of this topi
webinar

How to Write a Decision Tree (CART) as

How CART Handles Interactions



Model Setup
TN Interactions | Class Weights | Penalty | Lags | Automate
Model | Categorical | Testing | Select Cases | TreeNet | Plots&Options | TN Advanced
TreeNet Options
Configure
Std. Defaults | Save Defaults | Recall Defaults
Save Grove...
Cancel | Continue | Start

TreeNet Loss Function: Least squares (LS)

Criterion Determining Number of Trees Optimal for Regression Model: [✓ Mean Squared Error (MSE)] Mean Absolute Deviation (MAD) Mean Absolute Percentage Error (MAPE)

LAD Quantile: 0.50

Huber-M Breakdown: 0.90

Overfitting Protection

Learn Rate: Auto

Subsample fraction: 0.50

Limits

Number of trees to build: 500

Maximum nodes per tree: 6

Terminal node Minimum: Cases 10

Model Randomization

☐ Predictors per node: 4

☐ Predictors per tree: 4

☐ Vary tree sizes randomly (As Poisson)

☐ Sample With Replacement

Influence trimming speed-up

Total Influence fraction: 0.10

Automatic Best Predictor Discovery: [✓ Off] Discover only Discover and run

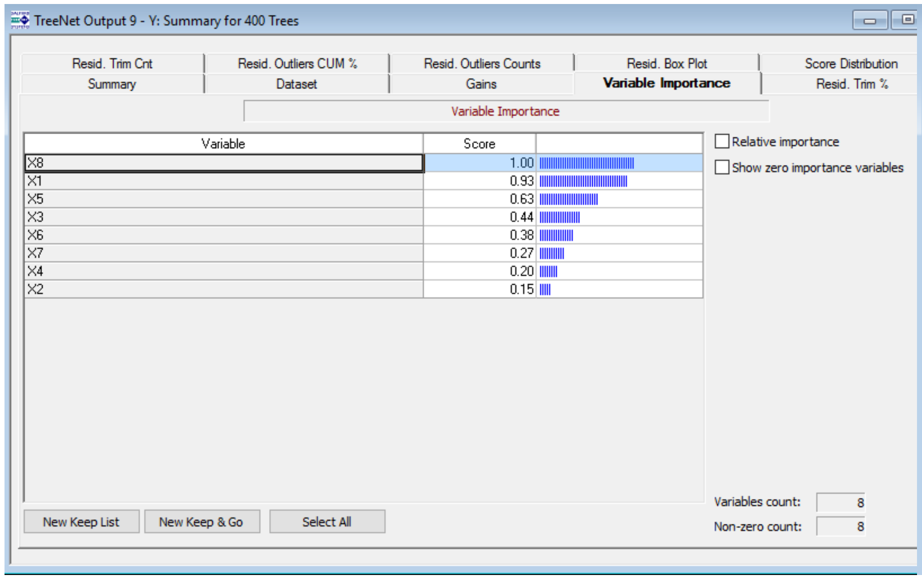
Maximum variables for each class: 8

After Building a Model

Number of Predictors in Model: 8

Analysis Engine: TreeNet Gradient Boosting Machine

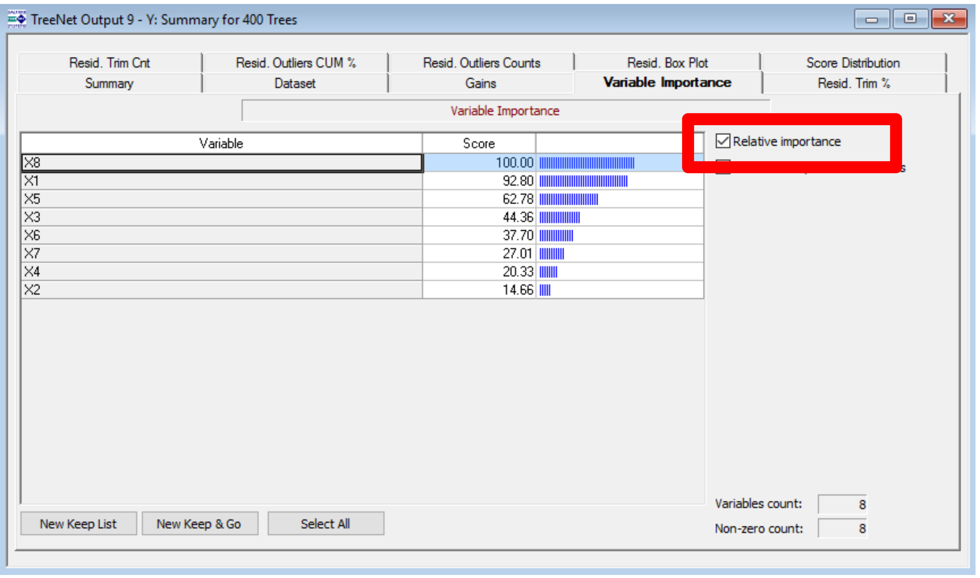
Variable Importance Measures



Variable	Score
X8	1.00
X1	0.93
X5	0.63
X3	0.44
X6	0.38
X7	0.27
X4	0.20
X2	0.15

☐ Relative importance
☐ Show zero importance variables

Variables count: 8
Non-zero count: 8



Variable	Score
X8	100.00
X1	92.80
X5	62.78
X3	44.36
X6	37.70
X7	27.01
X4	20.33
X2	14.66

☑ Relative importance

Variables count: 8
Non-zero count: 8

Relative Importance

Divide each of the raw importance scores by the largest score which means that the most important variable has a relative importance of 100 and the other variables have a relative importance values less than 100

Partial Dependency Plots

Partial Dependency Plots show how a variable (or set of variables) affect the average response of the model after <u>accounting for </u>(**not ignoring**) the other variables in the model

Partial dependency plots can be created for both categorical and numeric variables

Univariate Partial Dependency Plots

Assume that we have built a gradient boosted tree. Call it TN(x)

1. Sample a single record R

X1 = 400 X2= 300 X3= 140 …….

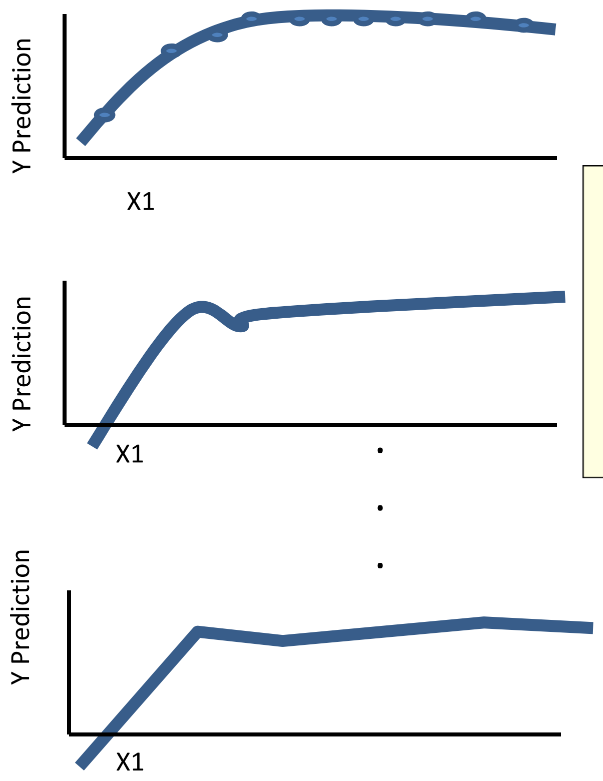
2. Hold every value in the record constant except for the value for the variable under consideration

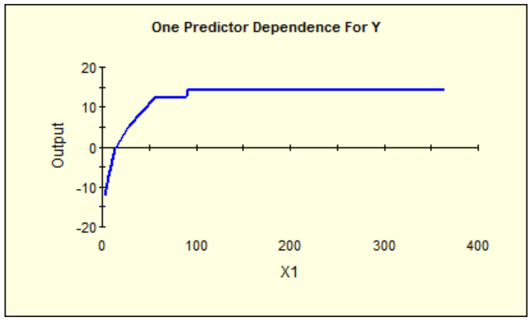
3. Plug the record R into the TreeNet model: TN(R); this generates one predicted value.

4. Now plug in the record again except this time change the value for the variable under consideration ; Do this multiple times which generates multiple points which are connected to form one curve

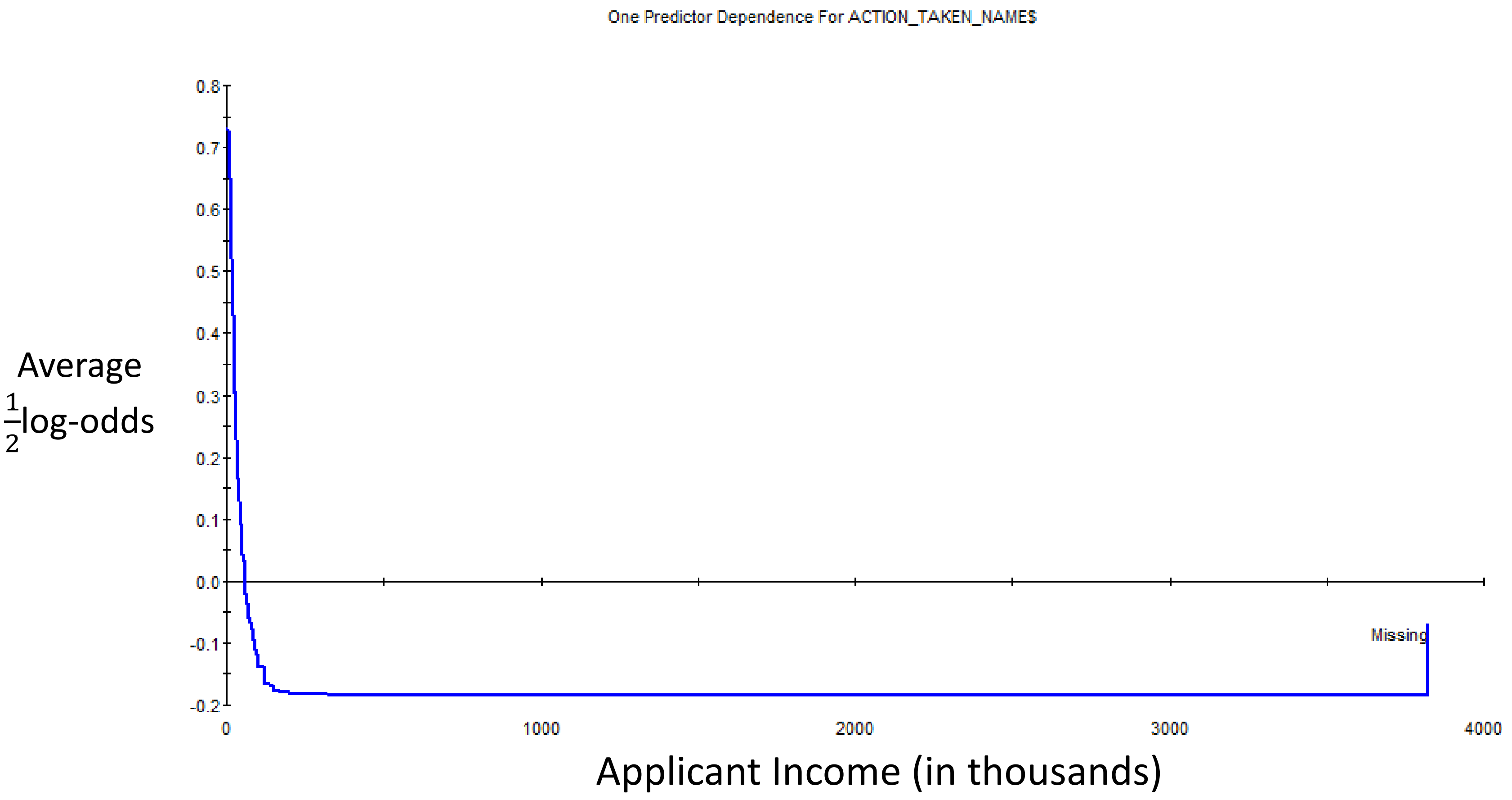
5. Repeat Steps 1-4 multiple times which results in multiple curves

6. Take the average of all curves generated to yield the final partial dependency plot



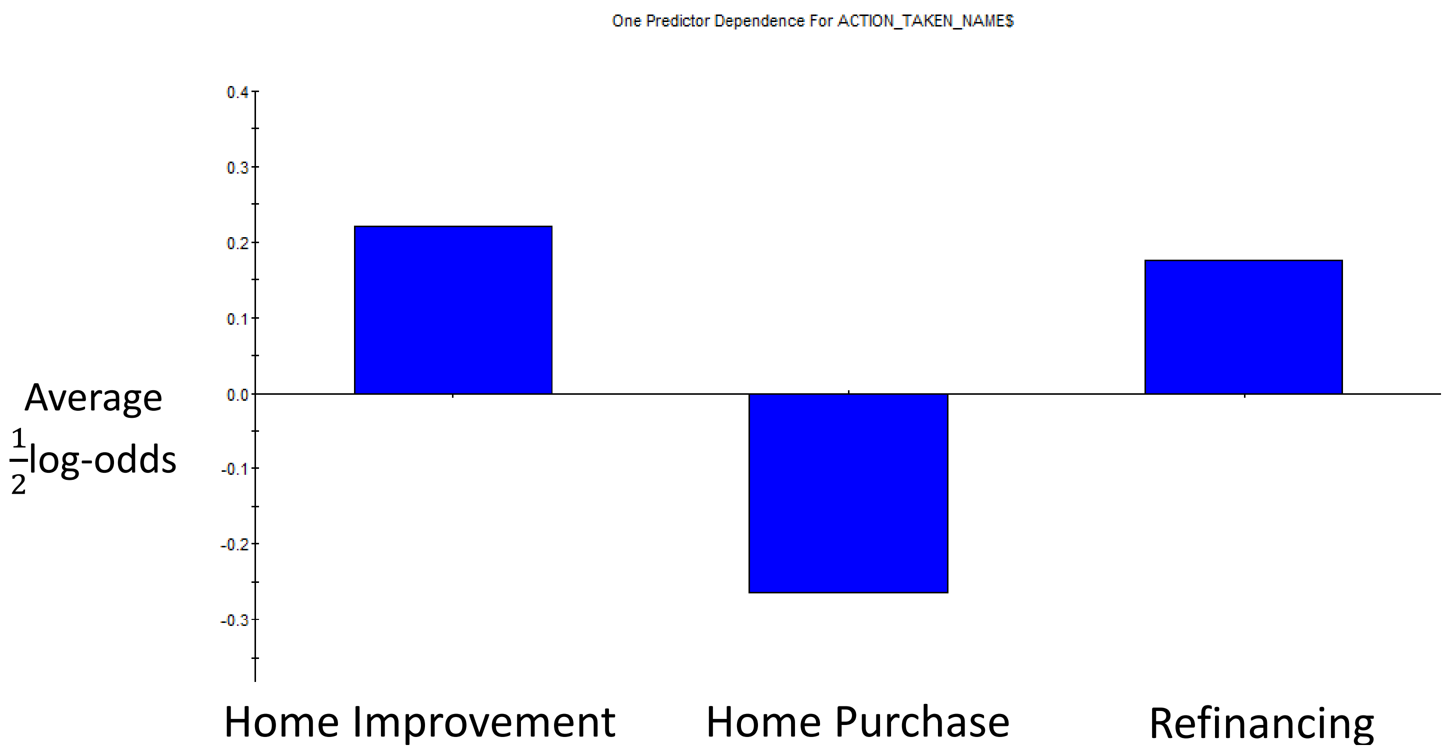


Univariate Partial Dependency Plot



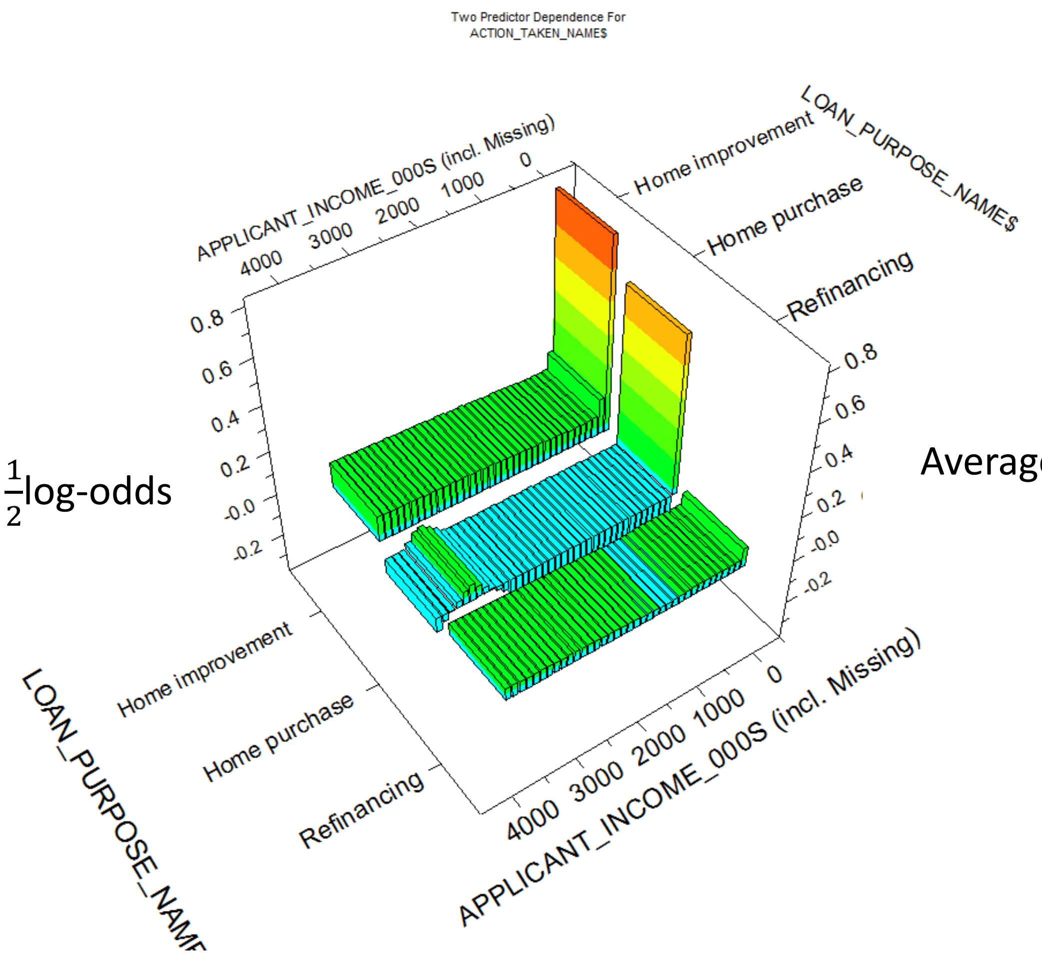
As income increases, the likelihood of being rejected falls. Note: 1,000 here is actually 1,000,000

Univariate Partial Dependency Plot



Those applying for a home improvement loan and refinancing are more likely to be rejected than those applying for a home purchase

Bivariate Partial Dependency Plot



Average $\frac{1}{2}$log-odds

Average $\frac{1}{2}$log-odds

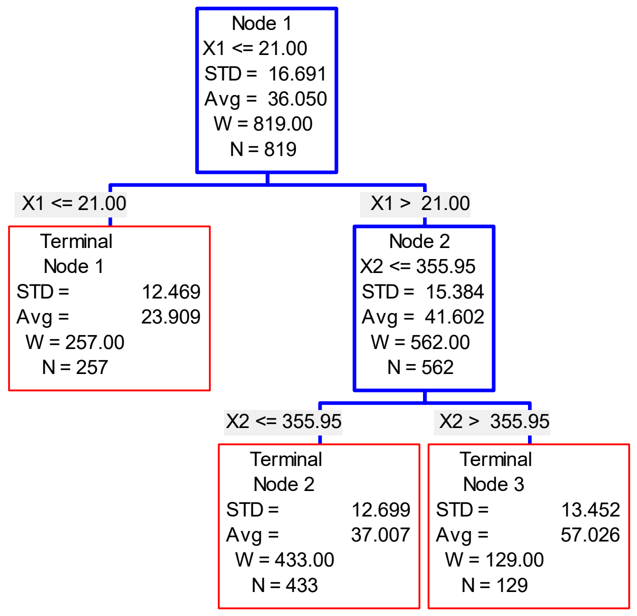
CART Advantages

In practice, **you can build CART models with dirty data** (i.e. missing values, lots of variables, nonlinear relationships, outliers, and numerous local effects)

This is due to CART's desirable properties:

1. Automatic handling of the following:
   a) Variable selection
   b) Variable interaction modeling (see **video links** for more information)
   c) Local effect modeling (see **video links** for more information)
   d) Nonlinear relationship modeling
   e) Missing values
   f) Outliers
2. Not affected by monotonic transformations of variables

CART: Interpretation and Automatic Variable Selection



Variable Selection:

All variables will be considered for each split but not all variables will be used. Some variables will be used more than others.

Only one variable is used for each split

The variables chosen are those that reduce the error the most

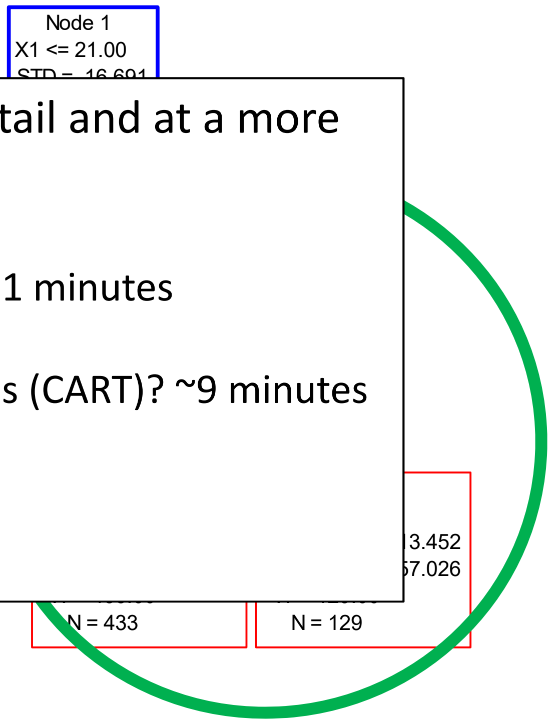
CART: Automatic Variable Interactions and Local Effects

In regression interaction terms modeled globally in
the forı
the inte

In CART
over ce
you do
terms c

Example
amounts of cement given that the Age is over 21 days (i.e. this is the interaction)

1. If X1 > 21 and X2 <= 355. 95 then the predicted value is 37.007
2. If X1 > 21 and X2 > 355.95 then the predicted value is 57.026



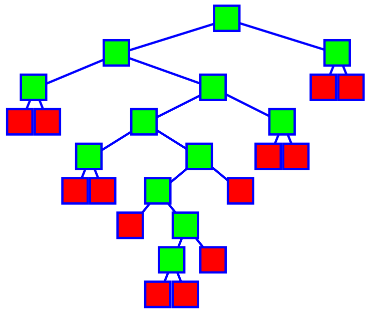
There are two videos that explain this concept in more detail and at a more convenient pace (watch them in order):

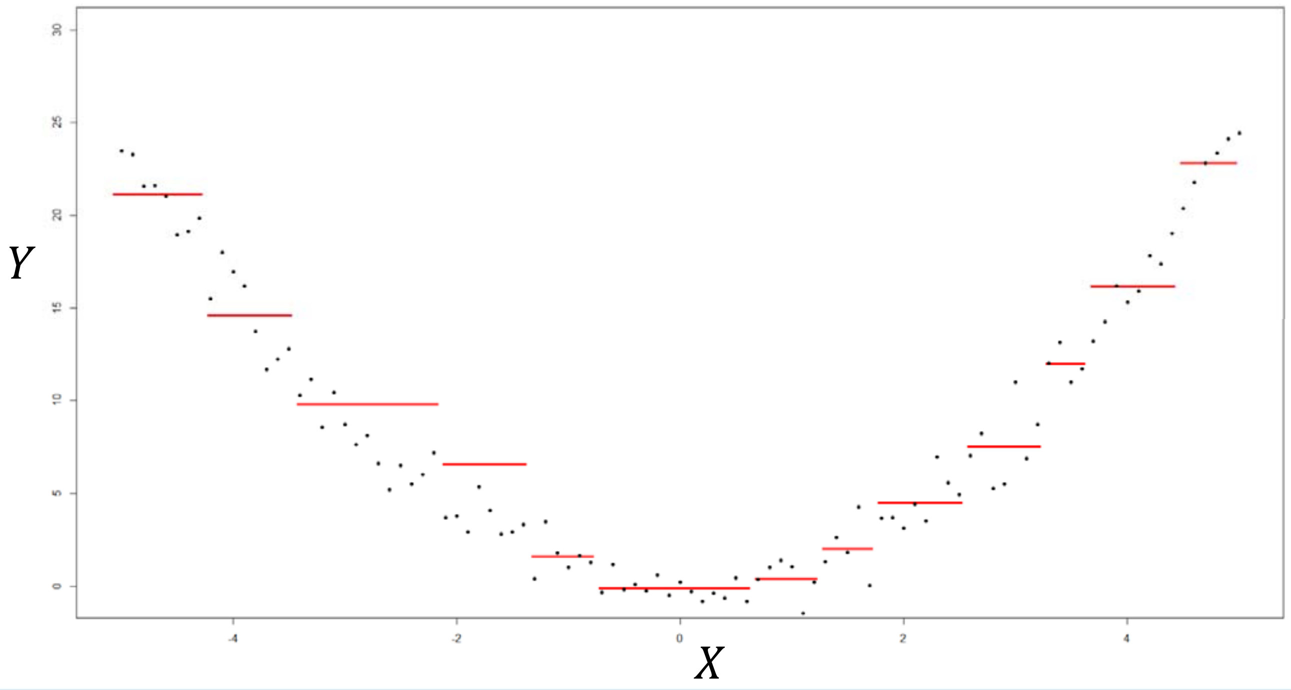
1. How to Write the Equation of a Decision Tree (CART) ~11 minutes
2. How are Variable Interactions Modeled in Decision Trees (CART)? ~9 minutes

The links are provided at the end of this presentation

CART: Automatic Nonlinear Modeling

Nonlinear functions (and linear) are approximated via step functions, **so in practice you do not need to worry about adding terms like x^2 *or* $ln(x)$ to capture nonlinear relationships.** The picture below is the CART fit to $Y = X^2$ + noise. CART modeled this data automatically. No data pre-processing. Just CART.





CART: Automatic Missing Value Handling

CART automatically handles missing values while building the model, so **you do not need to impute missing values yourself**

The missing values are handled using a *surrogate split*

- ***Surrogate Split-*** find another variable whose split is "similar" to the variable with the missing values and split on the variable that does not have missing values

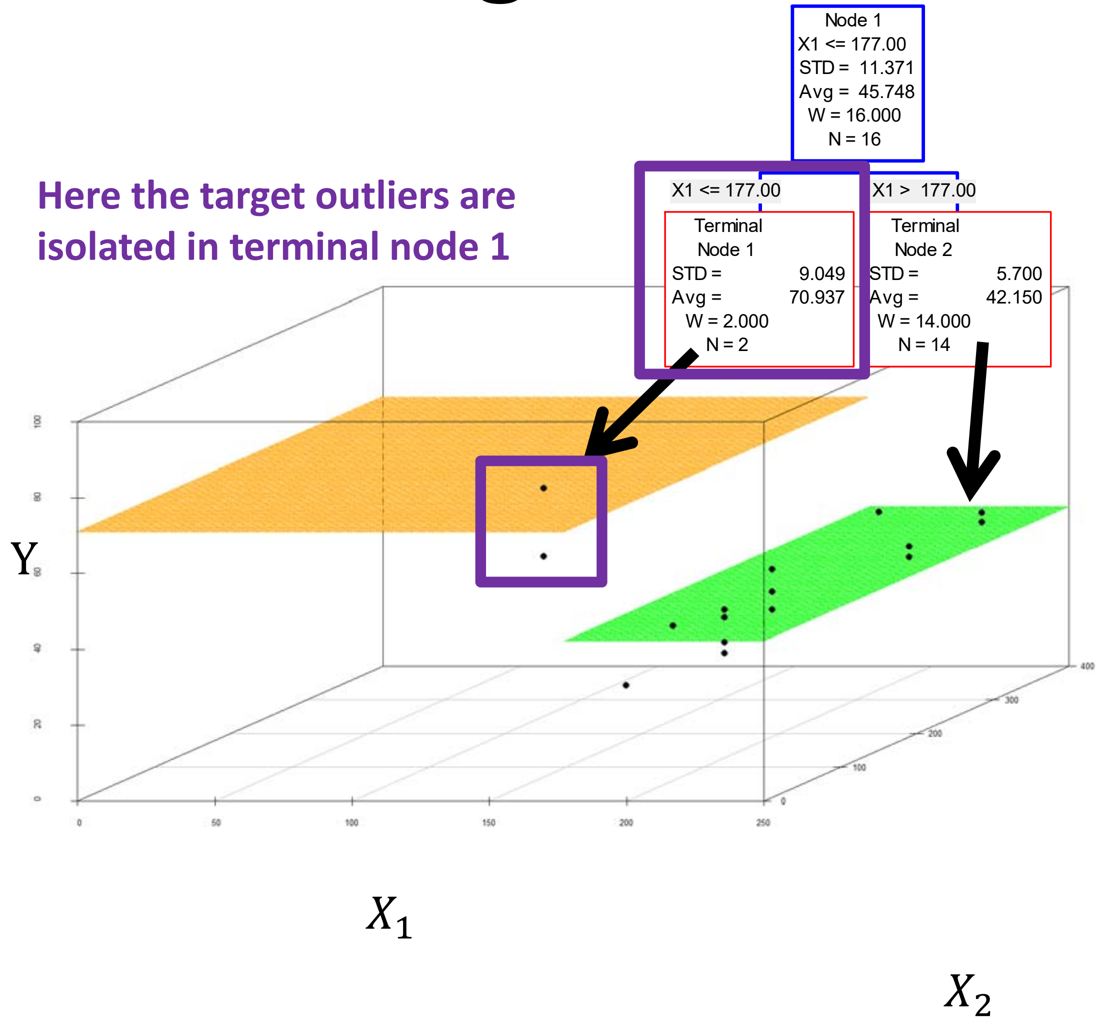
Reference: see Section 5.3 in Breiman, Friedman, Olshen, and Stone for more information

CART: Outliers in the Target Variable

Two types of outliers are
1. Outliers in the target variable (i.e. “Y”)
2. Outliers in the predictor variable (i.e. “x”)

CART is more sensitive to outliers with respect to the target variable
1. More severe in a regression context than a classification context
2. CART may treat target variable outliers by isolating them in small terminal nodes which can limit their effect

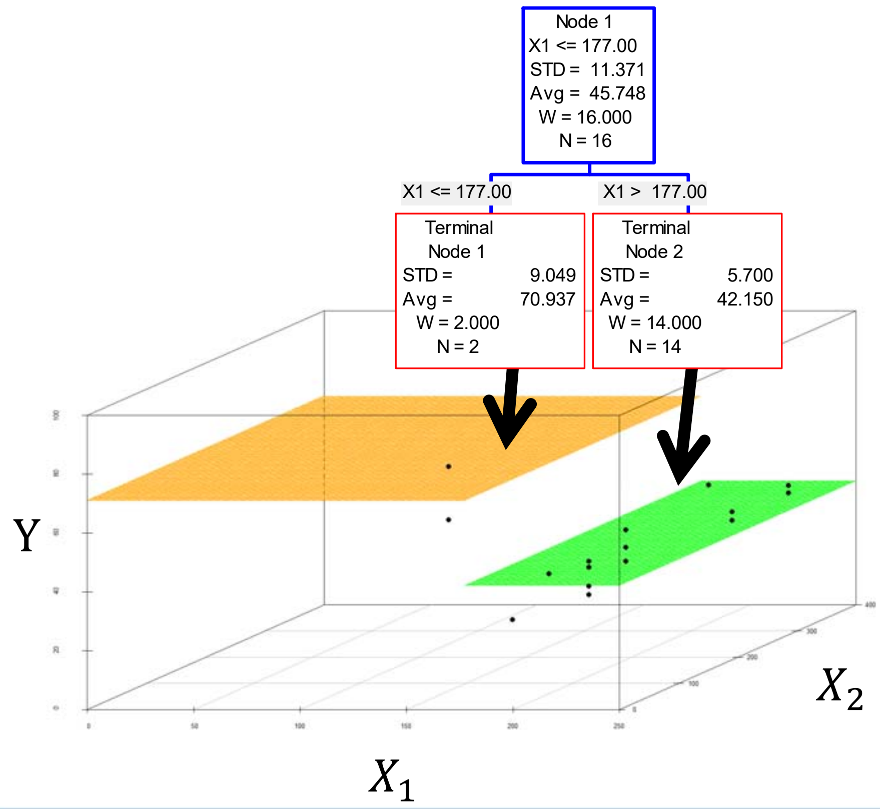
Reference: Pages 197-200 and 253 in Breiman, Friedman, Olshen, and Stone (1984)

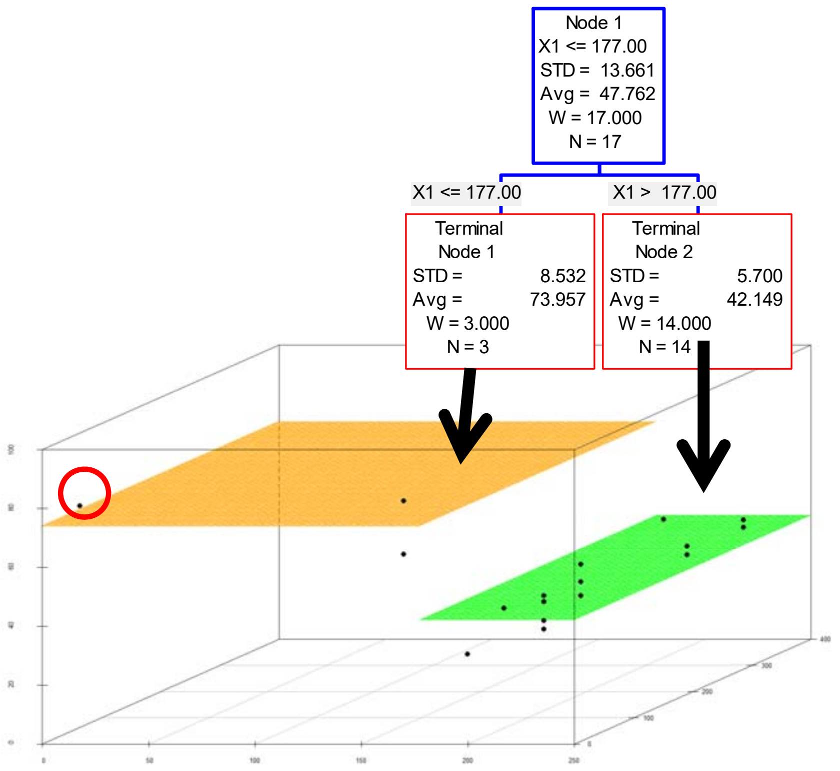


CART: Outliers in the Predictor Variables

CART is more robust to outliers in the predictor variables partly due to nature of the splitting process

Reference: Pages 197-200 and 253 in Breiman, Friedman, Olshen, and Stone (1984)





CART: Monotonic Transformations of Variables

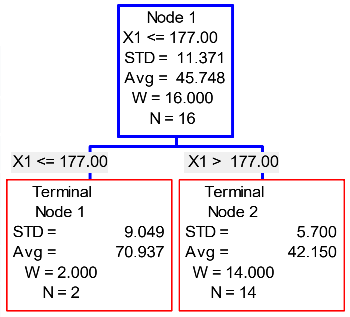
Monotonic transformation- is a transformation that does not change the order of a variable.

- CART, unlike linear regression, is not affected by this so **if a transformation does not affect the order of a variable then you do not need to worry about adding it to a CART model**
- **Example:** Our best first split in the example tree was $X_1 \leq 177$. What happens if we square X_1? The split point value changes, but nothing else does including the predicted values. This happens because the same Y values fall into the same partition (i.e. their order has not changed after we squared and sorted X_1)

Y	X_1	X_2
79.99	162	28
61.89	162	28
28.02	192	28
38.07	192	90
44.3	192	360
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365

Y	X_1	X_2
79.99	162	28
61.89	162	28

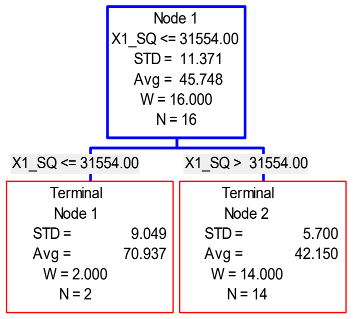
Y	X_1	X_2
28.02	192	28
38.07	192	90
44.3	192	360
36.45	228	28
45.85	228	28
39.29	228	28
47.81	228	28
47.03	228	90
42.33	228	90
52.91	228	90
40.27	228	270
43.01	228	270
41.05	228	365
43.7	228	365



Y	X_1	X_2
79.99	26,244	28
61.89	26,244	28
28.02	36,864	28
38.07	36,864	90
44.3	36,864	360
36.45	51,984	28
45.85	51,984	28
39.29	51,984	28
47.81	51,984	28
47.03	51,984	90
42.33	51,984	90
52.91	51,984	90
40.27	51,984	270
43.01	51,984	270
41.05	51,984	365
43.7	51,984	365

Y	X_1	X_2
79.99	26,244	28
61.89	26,244	28

Y	X_1	X_2
28.02	36,864	28
38.07	36,864	90
44.3	36,864	360
36.45	51,984	28
45.85	51,984	28
39.29	51,984	28
47.81	51,984	28
47.03	51,984	90
42.33	51,984	90
52.91	51,984	90
40.27	51,984	270
43.01	51,984	270
41.05	51,984	365
43.7	51,984	365



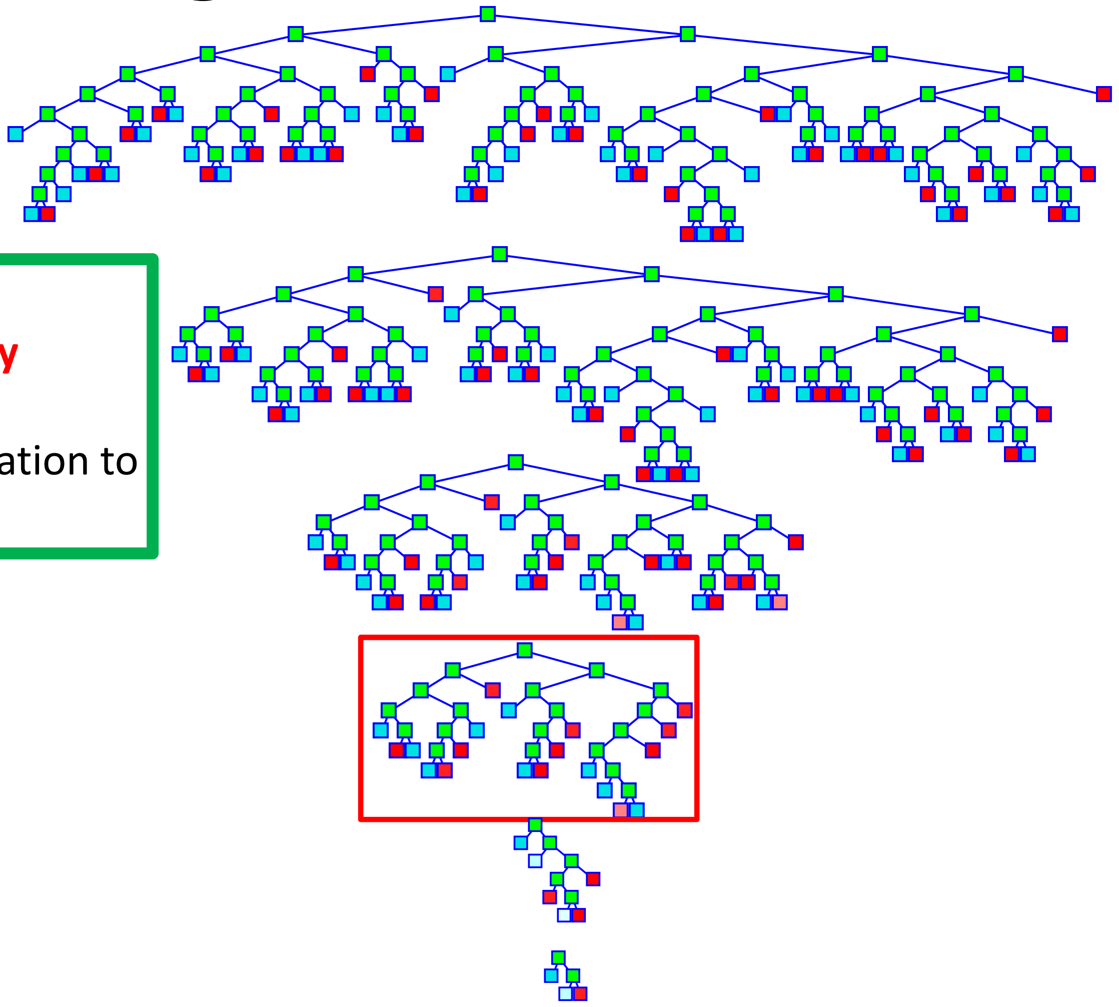
CART: Algorithm

Step 1: Grow a large tree

Step2: Prune the large tree

This is also done for you automatically

Use either a test sample or cross validation to prune subtrees

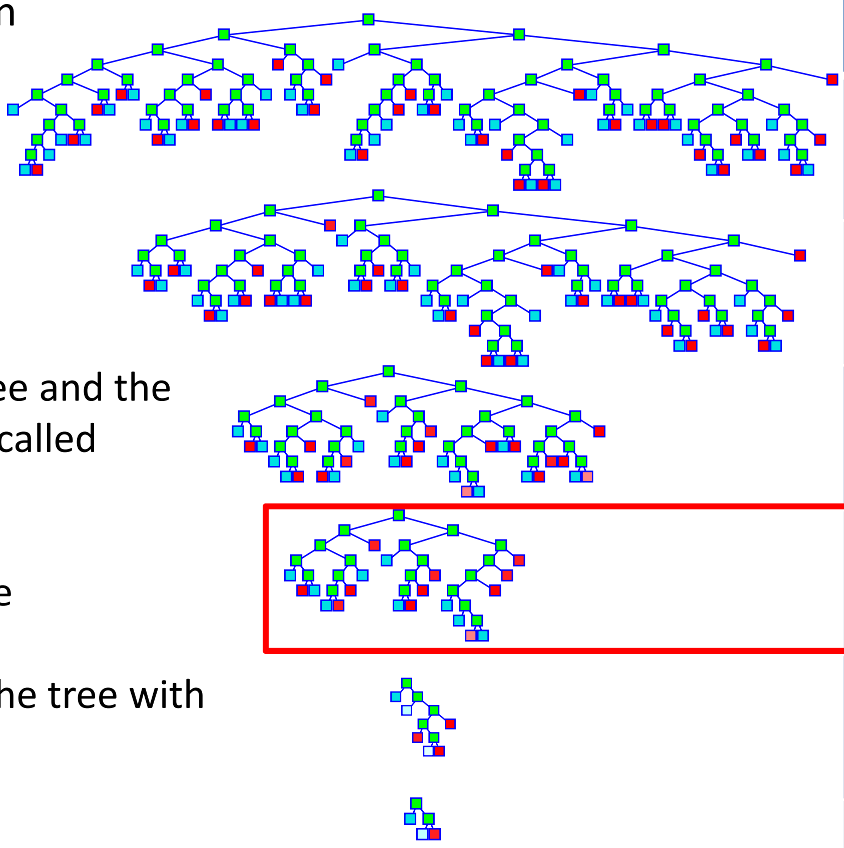


CART: Pruning with a Test Sample

Test sample- randomly select a certain percentage
of the data (often ~20%-30%)to be used to assess the model error

Prune the CART tree

1. Run the test data down the large tree and the smaller trees (the smaller trees are called "*subtrees*")
2. Compute the test error for each tree
3. The final tree shown to the user is the tree with the smallest test error



Subtree	Error
1	200
2	125
3	100
4	83
5	113
6	137