GRADIENT BOOSTING METHOD

Another ensemble method is known as **boosting**. Boosting as opposed to bagging doesn't involve bootstrap sampling. Instead, models are generated sequentially and iteratively, meaning that it is necessary to have information from iteration i before conducting iteration $i+1$. Note that the boosting process cannot be parallelized (modeling cannot be done on several trees simultaneously), unlike bagging, which is straightforwardly parallelizable.

The method of boosting was introduced by Michael Kearns and Leslie Valiant in 1989. The question posed asked whether it was possible to combine, in some fashion, a selection of weak machine learning models (termed **weak learners**) to produce a single strong machine learning model (a **strong learner**). Weak, in this instance means a model that is only slightly better than chance at predicting a response. Correspondingly, a strong learner is well-correlated to the true response.

This motivated the concept of boosting. The idea is to build iteratively weak machine learning models on a continually-updated response variable in the training data set and then add them together to produce a final, strong learning model. This differs from bagging, which simply averages the models on separate bootstrapped samples.

A basic boosting algorithm proceeds as follows:

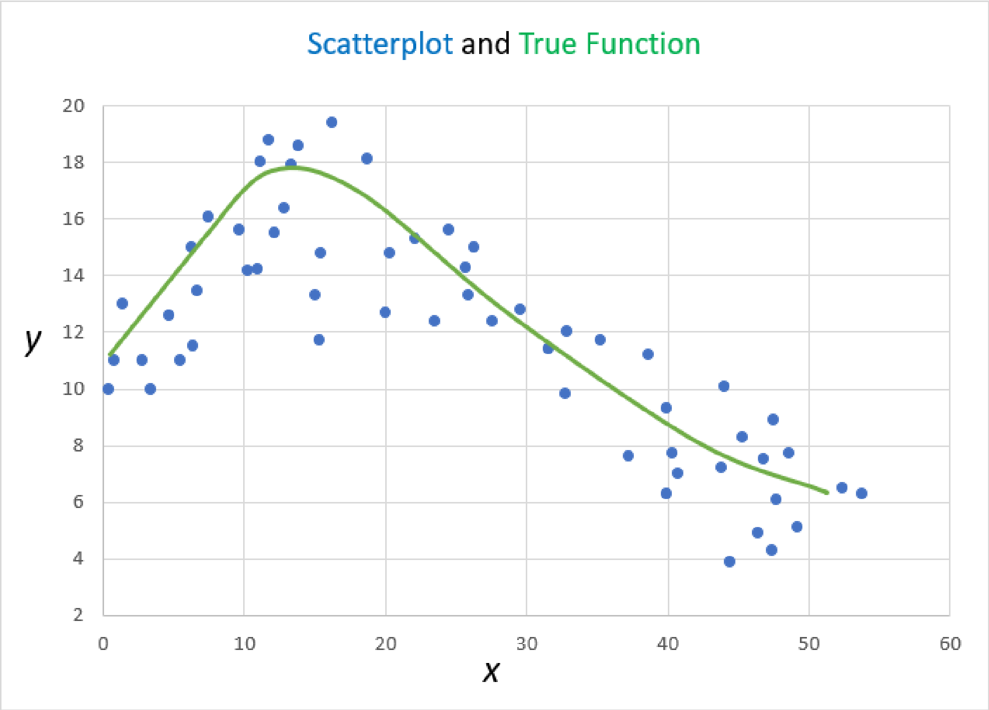
1. The initial estimator is set to zero, that is, $\hat{f}(x)=0$, and the residuals are set to current responses $r=y$, for all elements in the training set.
2. The number of boosting trees B is specified and then the loop over $b=1,...,B$ is run:
   Step 1. A weak-learning tree $\hat{f}^b$ with k splits is grown on the training data (x,r).
   Step 2. Estimator $\hat{f}$ is updated as $\hat{f}_{new}(x)=\hat{f}_{old}(x)+\lambda\,\hat{f}^b(x)$ for some scale parameter $\lambda,\ 0<\lambda<1$, called the **shrinkage rate** (or **learning rate**).
   Step 3. Residuals are updated as $r_{new}=r_{old}-\lambda\,\hat{f}^b(x)$.
3. The final boosted model is computed as the sum of individual weak learners, $\hat{f}(x)=\sum_{b=1}^{B}\ \lambda\,\hat{f}^b(x)$.

Notice that each subsequent tree is fitted to the residuals of the data. Hence each subsequent iteration is slowly improving the overall strong learner by improving its performance in poorly-performing regions of the feature space. It can be seen that this procedure is heavily dependent on the order in which the trees are grown. This process is said to **learn slowly**. Such **slow learning procedures** tend to produce well-performing machine learning models.

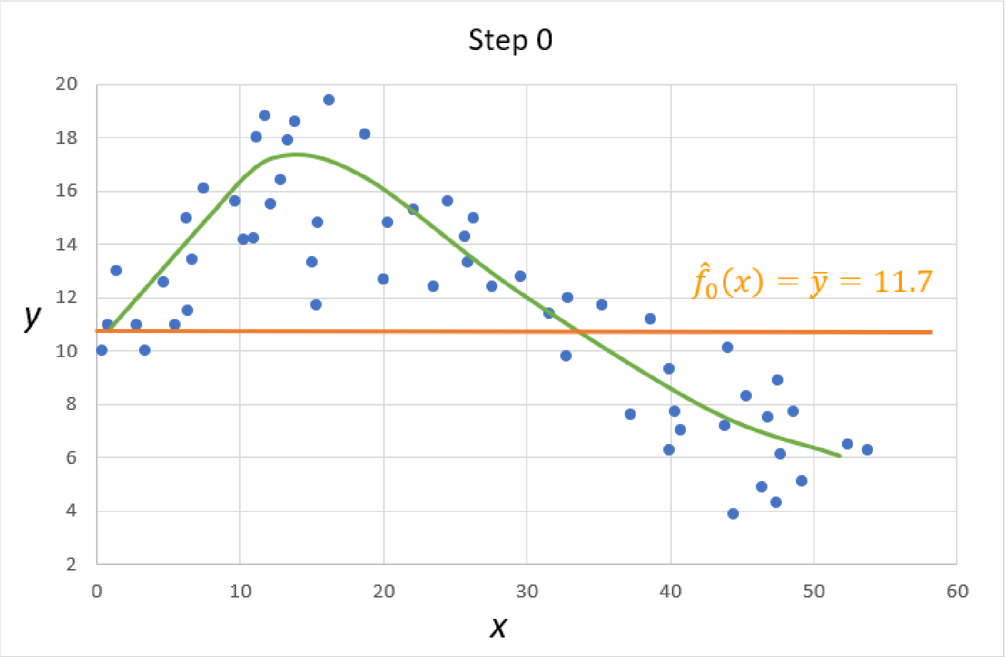
In the boosting algorithm, there are three hyperparameters: the number of boosted trees B, the number of splits k, and the shrinkage rate λ.

The **gradient boosting** method combines the **method of gradient descent** (or **steepest descent**) and the boosting algorithm. It was first introduced by Jerome Friedman in 1999.

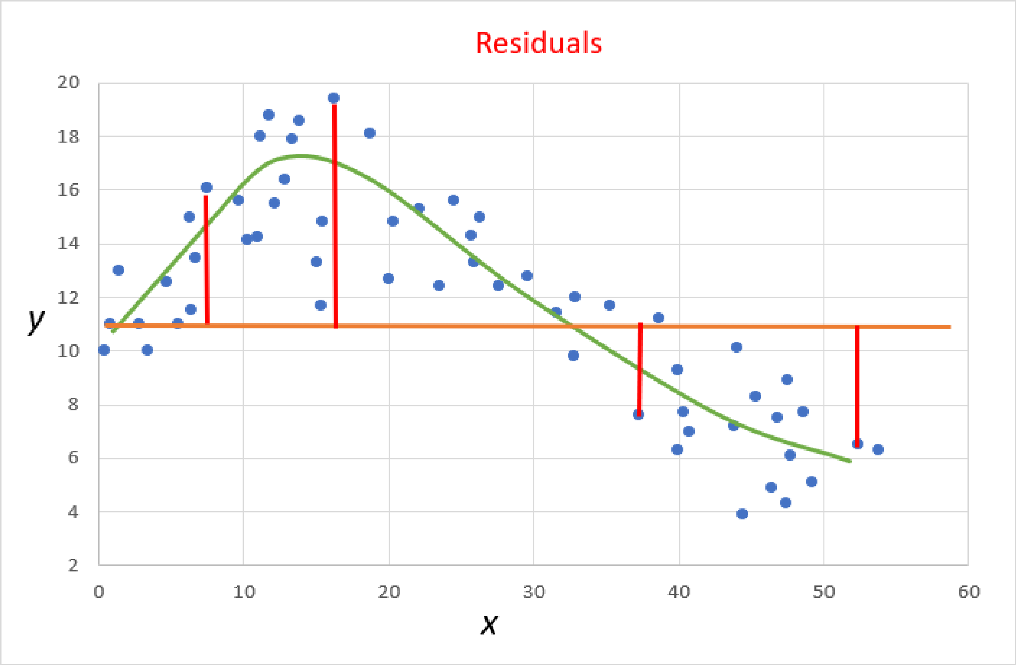
We present a simple example to explain how gradient boosting works. Suppose y depends on x through a non-linear relationship $y = f(x)$ depicted in the scatterplot below.



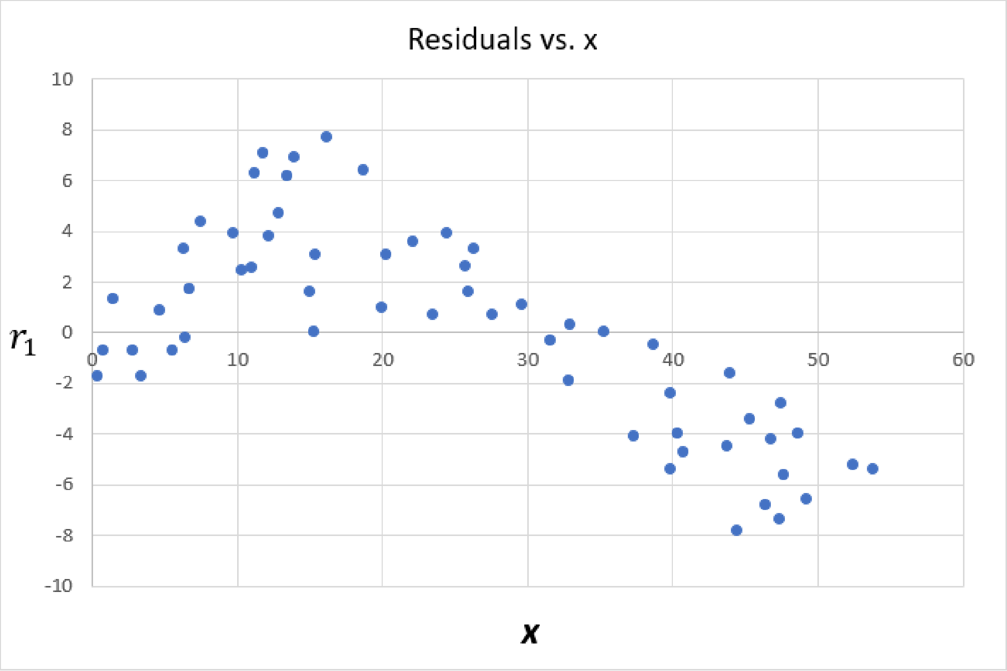
We initially predict the response y by the sample mean $\bar{y}$, that is, we let $\hat{f}_0(x) = \bar{y}$.



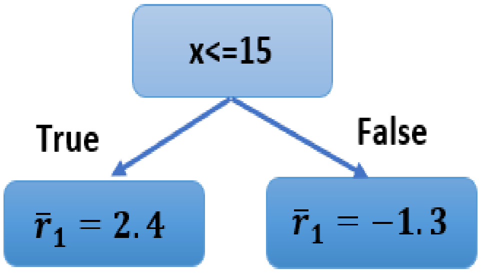
To improve our prediction, we will focus on the residuals (i.e., the vertical distances between the observed y's and the prediction $\bar{y}$). The residuals $r_1 = y - \bar{y}$ are shown as the vertical red lines in the figure below.



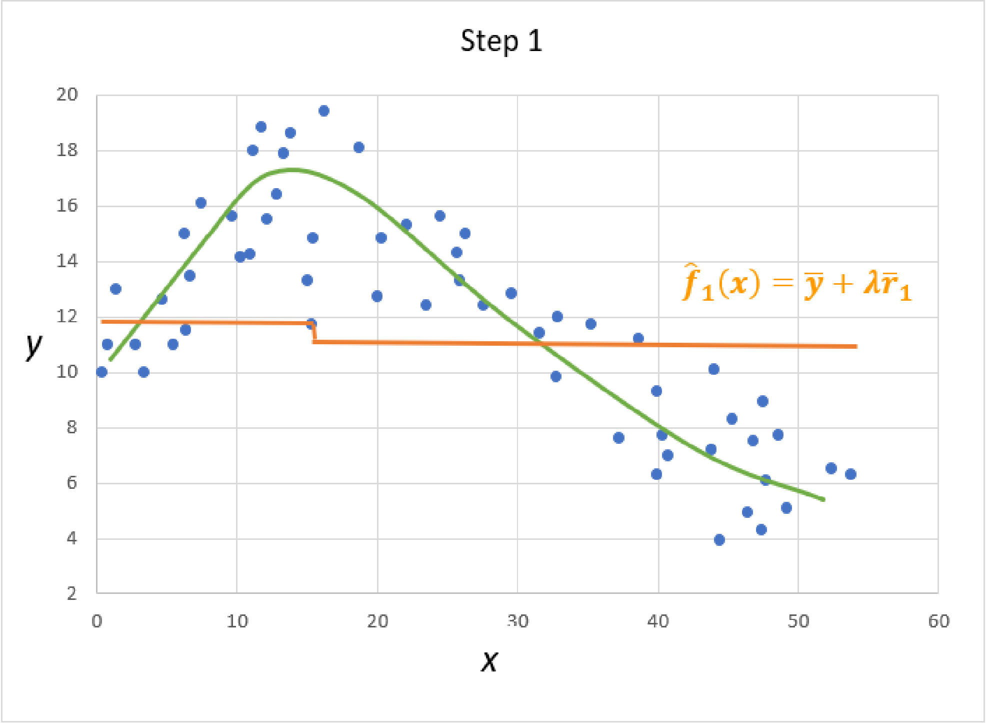
Next, we plot the residuals against x.



In the next step, we use the residuals r_1 as the target variable. Suppose for simplicity, we build a very simple regression tree, with one split and two terminal nodes (trees like this are called **decision stumps**). Suppose we split at 15, and so the decision stump looks this:

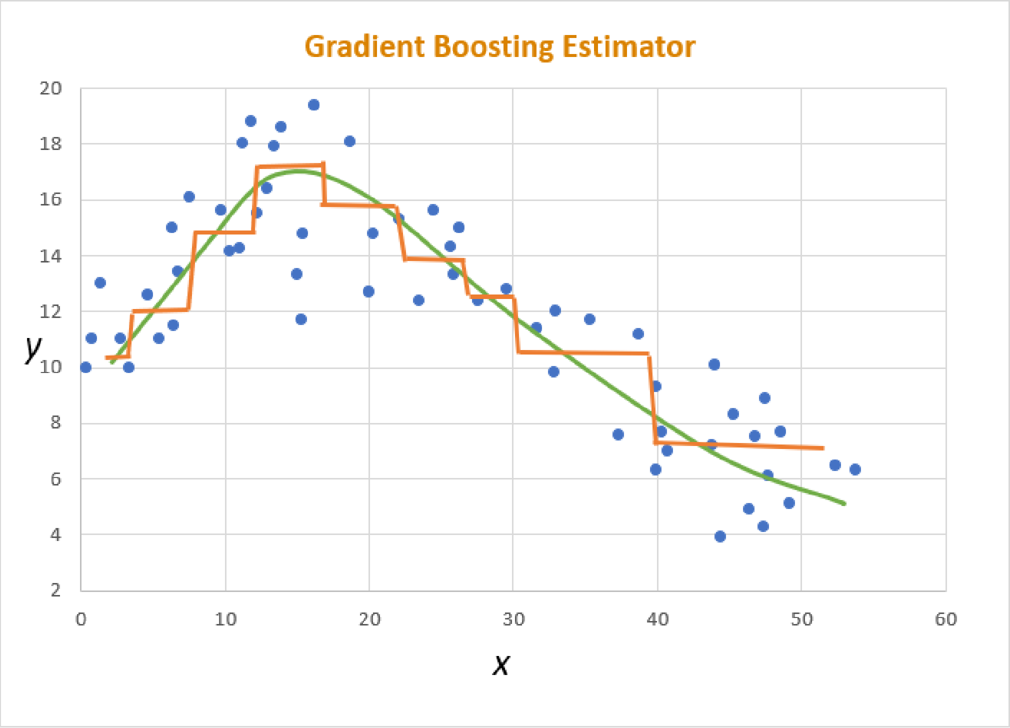


We then add the predicted $\bar{r}_1$ to the initial prediction $\bar{y}$ to reduce residuals. However, the gradient boosting algorithm does not simply add $\bar{r}_1$ to $\bar{y}$ as it overfits the model to the training data. Instead, $\bar{r}_1$ is scaled down by the shrinkage rate (or learning rate) λ, and then added to $\bar{y}$. For instance, we can take $\lambda = 0.2$. Then for $x \leq 15$, $\hat{f}_1(x) = \hat{f}_0(x) + \lambda\,\bar{r}_1 = \bar{y} + \lambda\bar{r}_1 = 11.7 + (0.2)(2.4) = 11.8$, and for $x > 15$, $\hat{f}_1(x) = 11.7 + (0.2)(-1.3) = 11.0$. We plot this fitted function in the figure below.



Now, in the next step, we update the residuals to $r_2 = y - \hat{f}_1(x)$ and build a regression tree, which will give us another split and another pair of estimates $\bar{r}_2$. We then update the fitted function $\hat{f}_2(x) = \hat{f}_1(x) + \lambda \bar{r}_2$.

We iterate these steps until the model prediction stops improving. The figures below schematically show the boosting process for some number of iterations.



Further, putting the gradient boosting algorithm into rigorous mathematical terms, we can write:

1. We initialize the model with a constant value $\hat{f}_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$ where by L we denote a pre-specified loss function.

Example. For the squared loss function $L = (y_i - \gamma)^2$, the value of γ that minimizes $\sum_{i=1}^n L(y_i, \gamma)$ solves

$$\frac{\partial}{\partial \gamma} \sum_{i=1}^n (y_i - \gamma)^2 = -2 \sum_{i=1}^n (y_i - \gamma) = -2 \sum_{i=1}^n y_i + 2n\gamma = 0.$$

Solving we get $\gamma = \bar{y}$. Thus, for the squared loss function, we initialize the model with $\hat{f}_0(x) = \bar{y}$. □

2. We define b as our iteration counter that will range between 1 and B. For each iteration b, we conduct the following steps:

Step 1. We compute **residuals** according to the formula

$$r_{ib} = -\Big[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\Big]_{f(x_i)=\hat{f}_{b-1}(x_i)}, \quad i = 1, \dots, n.$$

Example. For the squared loss function, we compute

$$r_{ib} = -\Big[\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\Big]_{f(x_i)=\hat{f}_{b-1}(x_i)} = -\Big[\frac{\partial \big(y_i - f(x_i)\big)^2}{\partial f(x_i)}\Big]_{f(x_i)=\hat{f}_{b-1}(x_i)} = 2\big(y_i - \hat{f}_{b-1}(x_i)\big).$$

Ignoring the multiplicative constant 2, we see that the residuals r_{ib} are the distances between the observed y_i and fitted $\hat{f}_{b-1}(x_i)$ (so, these are residuals in the regular sense). □

Step 2. We train a regression tree with target r_{ib} and feature x. This tree defines terminal node regions R_{jb}, $j = 1, \dots, J_b$.

Step 3. For each region, we compute optimal estimators

$$\gamma_{jb} = \arg\min_{\gamma} \sum_{x_i \in R_{jb}} L\big(y_i, \hat{f}_{b-1}(x_i) + \gamma\big), \quad j = 1, \dots, J_b.$$

Example. For the squared loss function, we compute

$$\gamma_{jb} = \arg\min_{\gamma} \sum_{x_i \in R_{jb}} L\big(y_i, \hat{f}_{b-1}(x_i) + \gamma\big) = \arg\min_{\gamma} \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i) - \gamma\big)^2.$$

The value of γ that minimizes this sum is the solution of the equation:

$$\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i) - \gamma\big)^2 = 0,$$

$$-2 \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i) - \gamma\big) = 0,$$

$$\sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i)\big) = \sum_{x_i \in R_{jb}} \gamma = \gamma\, n_{jb},$$

where n_{jb} is the number of data points in the region R_{jb}. Finally, we get

$$\gamma = \frac{1}{n_{jb}} \sum_{x_i \in R_{jb}} \big(y_i - \hat{f}_{b-1}(x_i)\big) = \frac{r_{jb}}{2n_{jb}} = \bar{r}_{jb}/2.$$

Ignoring the 2 in the denominator, we see that the estimator γ is the average of the residuals. □

Step 4. We update the model as

$$\hat{f}_b(x) = \hat{f}_{b-1}(x) + \lambda \sum_{j=1}^{J_b} \gamma_{jb}\mathbb{I}(x \in R_{jb})$$

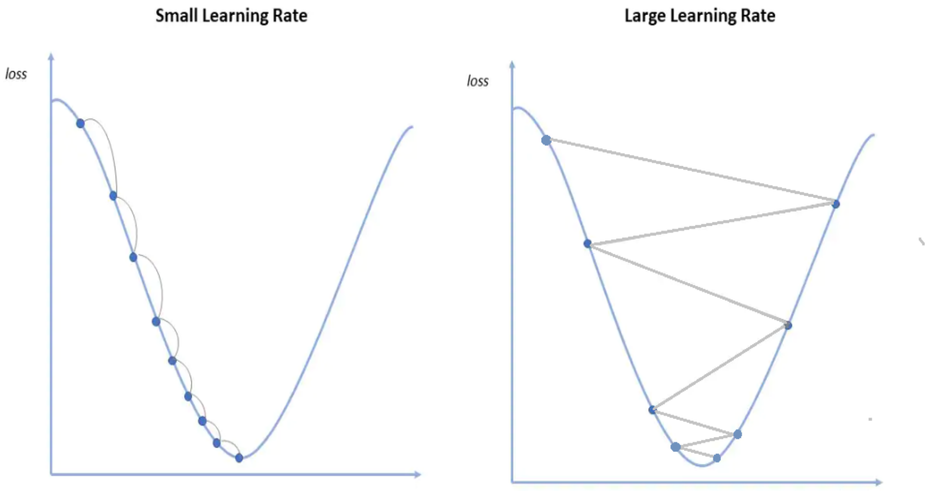
where the shrinkage (or learning) rate λ is a pre-defined constant between 0 and 1 (typically, 0.1 or smaller), and $\mathbb{I}(\cdot)$ is the indicator function (1 if true, and 0, otherwise).

Example. Consider the case of the squared loss function. Suppose x_i belongs to the region R_jb. We estimate $f_b(x_i)$ by

$$\hat{f}_b(x_i) = \hat{f}_{b-1}(x_i) + \lambda \sum_{j=1}^{J_b} \gamma_{jb}\mathbb{I}(x \in R_{jb}) = \hat{f}_{b-1}(x_i) + \lambda\,\bar{r}_{jb}/2 = f_{b-1}(x_i) + \lambda_1\,\bar{r}_{jb}.$$

Here λ_1 absorbed the constant 2, but still is a constant between 0 and 1, and can be considered the learning rate. □

Remark. The method of **gradient boosting** is closely related to the method of **gradient descent** (or **steepest descent**), which is a first-order iterative optimization algorithm for finding a local minimum of a differentiable function. The idea is to take repeated steps in the opposite direction of the gradient of the function at the current point because this is the direction of the steepest descent. The descent is depicted in the figure below. Note that if the learning rate (length of step) is small, one would descend slowly along one slope. However, one can choose to take large learning steps. Convergence is still guaranteed but it will take more time and computations to reach the minimum.



Remark. In the literature, the method of gradient boosting of a regression tree goes by a variety of names: gradient boosting machine (GBM), functional gradient boosting, multiple additive regression trees (MART), boosted regression trees (BRT), generalized boosting model, or tree net. In R, we will fit the extreme gradient boosting (XGBoost) method which is a popular modern implementation of the gradient boosting method with some extensions, like second-order optimization.

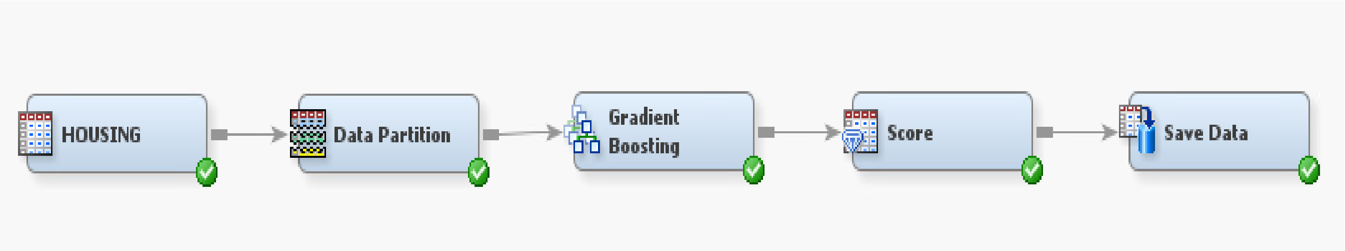
Example. We apply the gradient boosting algorithm to the data in the file "housing_data.csv". The codes below run the algorithm and output the list of features in the order of their importance, and also compute the proportion of correctly predicted median house prices within 10%, 15%, and 20% of the true values.

In SAS: Due to the large memory required to run the model, we have to resort to SAS Enterprise Miner Workstation 14.2 (in Student Virtual Lab). It runs on the SAS Viya platform that utilizes Cloud Analytics Service (CAS).

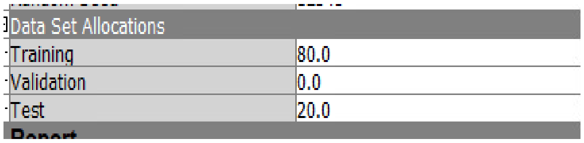
First, we go to Student Virtual Lab (SVL), open SAS, import the data set into SAS, and store it in the `sasuser` library. The code is:

```
proc import out=sasuser.housing datafile="./housing_data.csv" dbms=csv replace;
run;
```

Then we open SAS Enterprise Miner Workstation (EM), create a new project named, say, "XGBoostReg", and specify SAS Server Directory as the **desktop in SVL**. Then right-click "Data Sources" and extract the housing data set from the sasuser library. We also change the role of "median_house_value" to "target". Next, we right-click "Diagrams" and create a new process flow diagram depicted here:

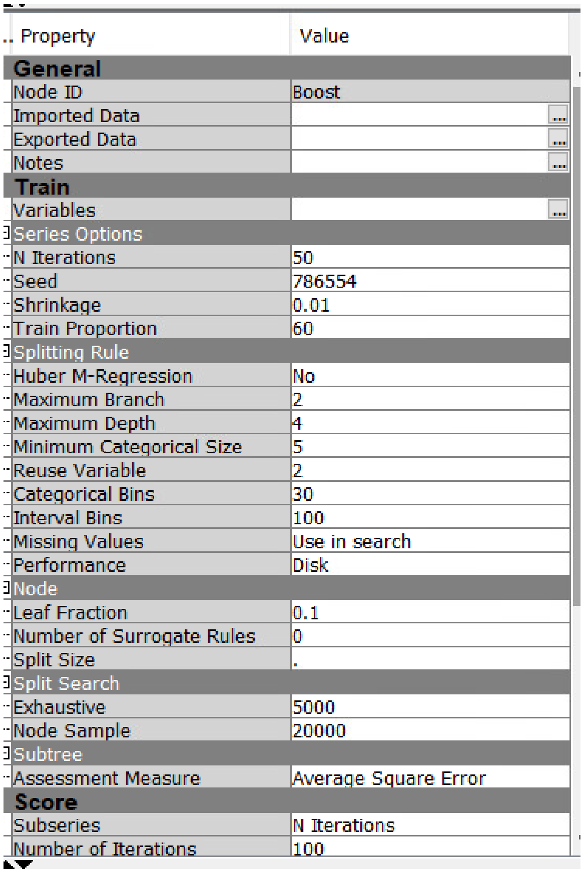


We can set specifications for data partition (click on the node "Data Partition") to 80% of training data, 0% of validation data, and 20% of testing data. See the snippet below.



Data Set Allocations	
Training	80.0
Validation	0.0
Test	20.0
Report	

Further, it is recommended to set the specifications for the "Gradient Boosting" node to the ones displayed here:



Property	Value
General	
Node ID	Boost
Imported Data	...
Exported Data	...
Notes	...
Train	
Variables	...
Series Options	
N Iterations	50
Seed	786554
Shrinkage	0.01
Train Proportion	60
Splitting Rule	
Huber M-Regression	No
Maximum Branch	2
Maximum Depth	4
Minimum Categorical Size	5
Reuse Variable	2
Categorical Bins	30
Interval Bins	100
Missing Values	Use in search
Performance	Disk
Node	
Leaf Fraction	0.1
Number of Surrogate Rules	0
Split Size	.
Split Search	
Exhaustive	5000
Node Sample	20000
Subtree	
Assessment Measure	Average Square Error
Score	
Subseries	N Iterations
Number of Iterations	100

The next step is to right-click on the "Save Data" node and run the path. The output is the scored testing data set that is located in the SAS data file that can be retrieved from the folder "XGBooostReg" on the desktop. The path to the file is

"XGBoostReg/Workspaces/EMWS1/EMSave/em_save_test.sas7bdat".

Once we locate this file, we open it in SAS (in the library "Tmp1") and compute the accuracy of prediction.

Gradient Boosting Method for Binary Classification

The gradient boosting algorithm for binary classification works as follows.
1. We initialize the model with a constant value $\hat{f}_0(x) = \arg\min_\gamma \sum_{i=1}^n L(y_i, \gamma)$ where L is the binary cross-entropy loss function given by $L(y_i, \gamma) = y_i \ln \gamma + (1 - y_i)ln(1 - \gamma)$. To minimize $\sum_{i=1}^n L(y_i, \gamma)$ with respect to γ, we solve

$$\frac{\partial}{\partial \gamma} \sum_{i=1}^n \big[y_i \ln \gamma + (1 - y_i)\ln(1 - \gamma)\big] = \sum_{i=1}^n \Big[\frac{y_i}{\gamma} - \frac{1 - y_i}{1 - \gamma}\Big] = \frac{n\bar{y}}{\gamma} - \frac{n - n\bar{y}}{1 - \gamma} = 0.$$

From here, $\gamma = \bar{y}$.
2. An iterative process is initiated with respect to a counter b, ranging between 1 and B, where B is a user-defined number of boosting trees. For each iteration b, we perform the following steps:

Step 1. The iterative process starts with the model residuals computed by the formula:

$$r_{ib} = -\Big[\frac{\partial\big(y_i \ln f(x_i) + (1 - y_i)\ln(1 - f(x_i))\big)}{\partial f(x_i)}\Big]_{f(x_i) = \hat{f}_{b-1}(x_i)}$$

$$= -\Big[\frac{y_i}{f(x_i)} - \frac{1 - y_i}{1 - f(x_i)}\Big]_{f(x_i) = \hat{f}_{b-1}(x_i)} = -\frac{y_i - \hat{f}b - 1(x_i)}{\hat{f}_{b-1}(x_i)(1 - \hat{f}_{b-1}(x_i))}, \quad r = 1, \dots, n.$$

These residuals become the target variable in the next iteration.

Step 2. At every iteration, we build a decision tree. Denote its terminal nodes' regions by R_{jb}, $j = 1, \dots, J_b$.

Step 3. For each region, we compute optimal estimators

$$\gamma_{jb} = \arg\min_\gamma \sum_{x_i \in R_{jb}} \Big[y_i \ln(\hat{f}_{b-1}(x_i) + \gamma) + (1 - y_i)\ln(1 - \hat{f}_{b-1}(x_i) - \gamma)\Big].$$

The value of γ that minimizes this sum is a solution of the following equation:

$$\frac{\partial}{\partial \gamma} \sum_{x_i \in R_{jb}} \Big[y_i \ln(\hat{f}_{b-1}(x_i) + \gamma) + (1 - y_i)\ln(1 - \hat{f}_{b-1}(x_i) - \gamma)\Big] = 0,$$

or, equivalently,

$$\sum_{x_i \in R_{jb}} \Big[\frac{y_i}{\hat{f}_{b-1}(x_i) + \gamma} - \frac{1 - y_i}{1 - \hat{f}_{b-1}(x_i) - \gamma}\Big] = 0,$$

which simplifies to

$$\sum_{x_i \in R_{jb}} \Big[\frac{y_i - \hat{f}_{b-1}(x_i) - \gamma}{(\hat{f}_{b-1}(x_i) + \gamma)(1 - \hat{f}_{b-1}(x_i) - \gamma)}\Big] = 0.$$

This equation does not have a closed-form solution and has to be solved numerically.

Step 4. At the end of every iteration, we update the model as

$$\hat{f}_b(x) \,=\, \hat{f}_{b-1}(x) \,+\, \lambda \sum_{j=1}^{J_b} \gamma_{jb}\, \mathbb{I}(x \in R_{jb})$$

with some pre-specified shrinkage rate λ.

Gradient Boosting Method for Multinomial Classification

For multinomial classifiers, a multi-class cross-entropy loss function is applied. Suppose there are c classes, and n_j observations belong to class $j, j = 1, \dots, c$, where $n_1 + \dots, n_c = n$. Denote by t_{ij} an indicator of y_i belonging to class j. Note that $\sum_{i=1}^{n} t_{ij} = n_j, j = 1, \dots, c$. The loss function is given by $L(y_i, \gamma_1, \dots, \gamma_c) = t_{i1} \ln \gamma_1 + t_{i2} \ln \gamma_2 + \dots + t_{ic} \ln \gamma_c = t_{i1} \ln \gamma_1 + t_{i2} \ln \gamma_2 + \dots + t_{ic} \ln(1 - \gamma_1 - \gamma_2 - \dots - \gamma_{c-1})$. To minimize $\sum_{i=1}^{n} L(y_i, \gamma_1, \dots, \gamma_c)$ with respect to $\gamma_1, \dots, \gamma_c$, we solve for $j = 1, \dots, c-1$,

$$\frac{\partial}{\partial \gamma_j} \sum_{i=1}^{n} \big[t_{i1} \ln \gamma_1 + t_{i2} \ln \gamma_2 + \dots + t_{ic} \ln(1 - \gamma_1 - \gamma_2 - \dots - \gamma_{c-1})\big] = \sum_{i=1}^{n} \Big[\frac{t_{ij}}{\gamma_j} - \frac{t_{jc}}{1 - \gamma_1 - \gamma_2 - \dots - \gamma_{c-1}}\Big]$$

$$= \frac{n_j}{\gamma_j} - \frac{n_c}{1 - \gamma_1 - \gamma_2 - \dots - \gamma_{c-1}} = \frac{n_j}{\gamma_j} - \frac{n_c}{\gamma_c} = 0, \quad \text{or,} \quad \gamma_j = \frac{n_j}{n_c}\gamma_c.$$

Since $\gamma_1 + \gamma_2 + \dots + \gamma_c = 1$, we have that $\gamma_c\big(\frac{n_1}{n_c} + \dots + \frac{n_{c-1}}{n_c} + 1\big) = 1$, or $\gamma_c = \frac{n_c}{n_1 + \dots + n_c} = \frac{n_c}{n}$, and $\gamma_j = \frac{n_j}{n_c} \cdot \frac{n_c}{n} = \frac{n_j}{n}, \;\; j = 1, \dots, c. \quad \Box$.