Challenge: Piecewise linear Regression Tree

Lingtian Bu*
WISE 2017

Update: December 22, 2019

Abstract

In this assignment, I first review the original regression tree based on piecewise constant model. Second, I try to introduce the idea of piecewise linear regression tree with the fundation of Shi et al.(2019)'s paper. Third, simulation with R to compare the performance of the original tree and piecewise linear tree are presented.

Keywords: Decision Tree Piecewise Linear Tree

1 Review of Decision Tree - Regression Tree

Suppose $D = \{(x_i, y_i)\}_{i=1}^N$ is the training data. A regression tree of M leaves can be thought of as dividing the predictor space into M regions $R_1, \ldots, R_M$, each corresponding to a leaf of the tree. For each $R_m, m = 1, \ldots, M$, let

$$\bar{y}_m = \frac{1}{n_m} \sum_{x_i \in R_m} y_i \tag{1}$$

, where n_m is the number of observations in R_m. And the estimated regression function is:

$$\hat{f}(x) = \sum_{m=1}^{M} \bar{y}_m \cdot I\{x \in R_m\} \tag{2}$$

, that means for every observation that falls into the region R_m we make the same prediction, which is simply the mean of the response values for the training observations in R_m. Therefore the result is a piecewise constant model.

To build a regression tree, the goal is to find $R_1, \ldots, R_M$ that minimize E_{in}. In the regression setting, we use the L2 loss:

$$\min RSS = \sum_{m=1}^{M} \sum_{x_i \in R_m} (y_i - \bar{y}_m)^2 \tag{3}$$

*卜令天 15220172202239

To achieve this kind of goal, there's the CART algorithm which is a recursive partitioning algorithm that divides the predictor space by making successive binary splits. At each step, it chooses the split to achieve the biggest drop in E_{in}. It's a greedy approach since the optimal split is made at that particular step rather than looking ahead and picking a picking a split that will lead to a better tree in some future step. For regression trees, this means choosing (x_j, s) such that

$$\min \sum_{x_i \in R_{left}} \left(y_i - \bar{y}_{left}\right)^2 + \sum_{x_i \in R_{right}} \left(y_i - \bar{y}_{right}\right)^2 \tag{4}$$

, where

$$R_{left} = \left\{x \in R : x_j < s\right\} \ and \ R_{right} = \left\{x \in R : x_j \geqslant s\right\} \tag{5}$$

In this way, we can generate a large tree T_0. Then we are going to prune the tree by minimizing

$$\sum_{m=1}^{|T|} \sum_{x_i \in R_m} (y_i - \bar{y}_m)^2 + \alpha |T| \tag{6}$$

, where $|T|$ is the size of T and α ia the tuning parameter that controla the tradeoff between the subtree's complexity and its fit to the training data.

Therefore, the algorithm for building a regression tree is:

Alogorithm 1 *Building a Regression Tree*

Stage 1 Use recursive binary splitting to grow a large tree on the training data until the stop criteria.

Stage 2 Apply cost complexity pruning to the large tree in order to obtain a sequence of best subtrees, as a function of α.

Stage 3 Use cross-validation to choose α .

Stage 4 Return the subtree from Stage 2 that corresponds to the chosen value of α.

2 Piecewise Linear Regression Tree

Comparing with "normal" regression tree, there are two basic components of our piecewise linear trees (PL Trees):

- **Splits**: A split associated with an internal node is a condition used to partition the data in the node to its two child nodes. PL Trees use univariate splits in the form $x_{i,j} \leqslant c$, where $x_{i,j}$ is the jth feature value of data point $x_i \in R_m$. The feature j is called the $splitfeature$.
- **Linear Models**: On each leaf s, there ia a linear model $f_s(x_i) = b_s + \sum_{j=1}^{m_s} \beta_{s,j} x_{i,k_{s,j}}$, where $\left\{x_{i,k_{s,j}}\right\}_{j=1}^{m}$ is a subset of $\left\{x_{i,j}\right\}_{j=1}^{m_s}$. We call features $\left\{k_{s,j}\right\}_{j=1}^{m_s}$ the $regressors$ for leaf s.

Corespondingly, there are two operations:

1. *SplitEval(s,j,c).* For a leaf s in tree T_k, a variable j and a real value with $x_{i,j} \leqslant c$ and fitting data in both child nodes.

2. *FitNode* fits a linear function on data in leaf s. The parameters of the function are calculated analytically to minimize the in-sample error.

Almost all the things the same except the estimating function. And the algorithm for PL Trees is shown as follows:

Alogorithm 2 *Building a Regression Tree*

Step 1 Initialize the tree with a single root node

Step 2 Put all the sample points in root node

Step 3 **while** number of leaves fewer than a preset value **do**:

for each leaf s **do**: $j_s^*, c_s^* \leftarrow \arg\max_{j,c} SplitEval\,(s, j, c)$;

$\hat{s} \leftarrow \arg\max_{s} SplitEval\left(s, j_s^*, c_s^*\right)$;

split $\hat{s}$ with condition $x_{i,j_s^*} \leqslant c_{\hat{s}}^*$ into s_1 and s_2

Stage 4 *FitNode(s_1),FitNode(s_2)*

The Comparision of piecewise constant model and piecewise linear model is shown in Figure 1

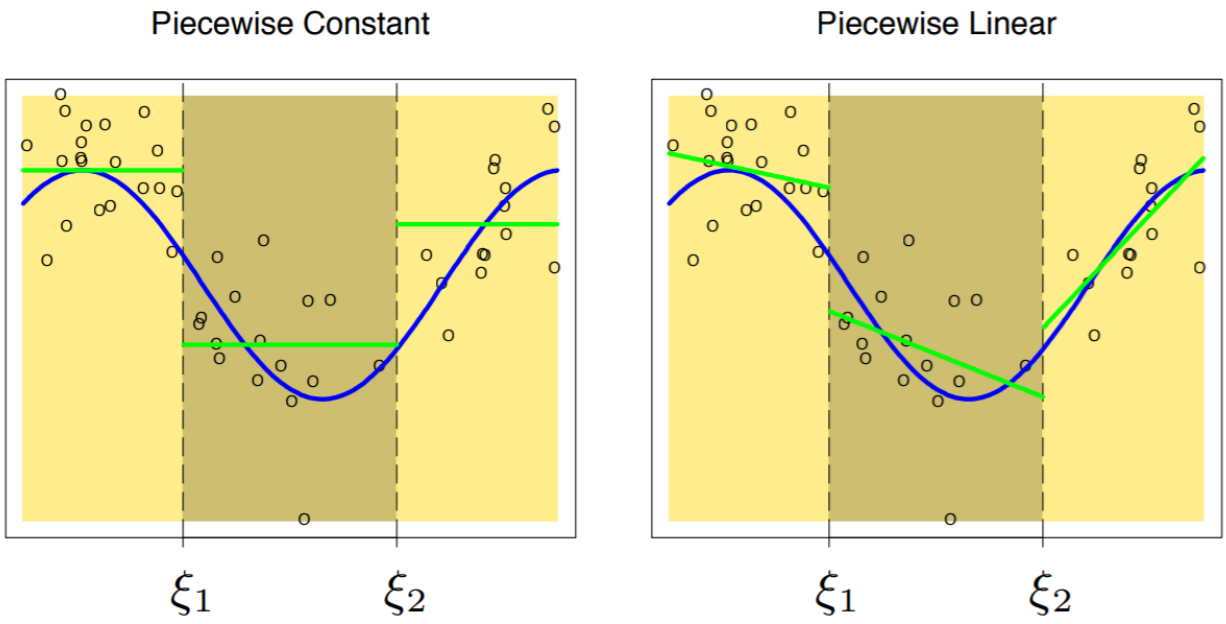


Figure 1: Comparision between PC and PL

3 References

[1] Regression. Jiaming Mao

[2] Decision Trees and Ensemble Methods. Jiaming Mao

[3] Yu Shi and Jian Li and Zhize Li. 2018. “Gradient Boosting With Piece-Wise Linear Regression Trees” arXiv, 1802.05640.

Trees

LingtianBu

2019.12.20

```
library(rpart)
```

```
## Warning: package 'rpart' was built under R version 3.6.1
```

```
library(readr)
```

```
## Warning: package 'readr' was built under R version 3.6.1
```

```
library(caTools)
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.1
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(party)
```

```
## Warning: package 'party' was built under R version 3.6.1
```

```
## Loading required package: grid
```

```
## Loading required package: mvtnorm
```

```
## Loading required package: modeltools
```

```
## Loading required package: stats4
```

```
## Loading required package: strucchange
```

```
## Warning: package 'strucchange' was built under R version 3.6.1
```

```
## Loading required package: zoo
```

```
## Warning: package 'zoo' was built under R version 3.6.1
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
## Warning: package 'sandwich' was built under R version 3.6.1
```

```
library(partykit)
```

```
## Warning: package 'partykit' was built under R version 3.6.1
```

```
## Loading required package: libcoin
```

```
## Warning: package 'libcoin' was built under R version 3.6.1
```

```
##
## Attaching package: 'partykit'
```

```
## The following objects are masked from 'package:party':
##
##     cforest, ctree, ctree_control, edge_simple, mob, mob_control,
##     node_barplot, node_bivplot, node_boxplot, node_inner,
##     node_surv, node_terminal, varimp
```

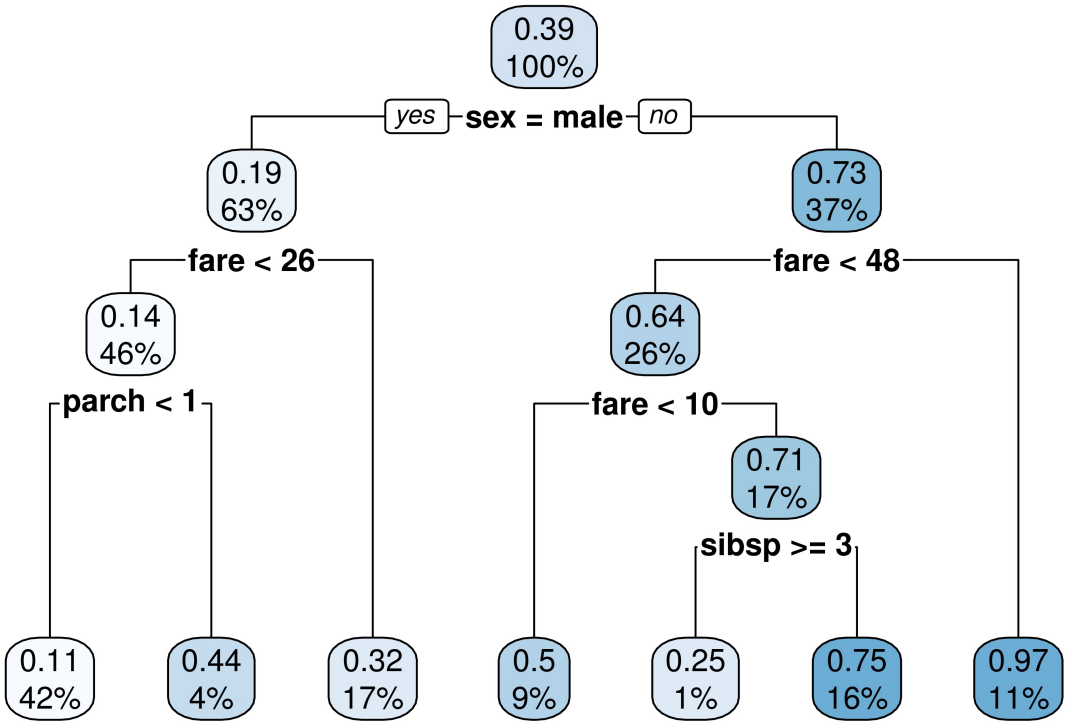
```
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.6.1
```

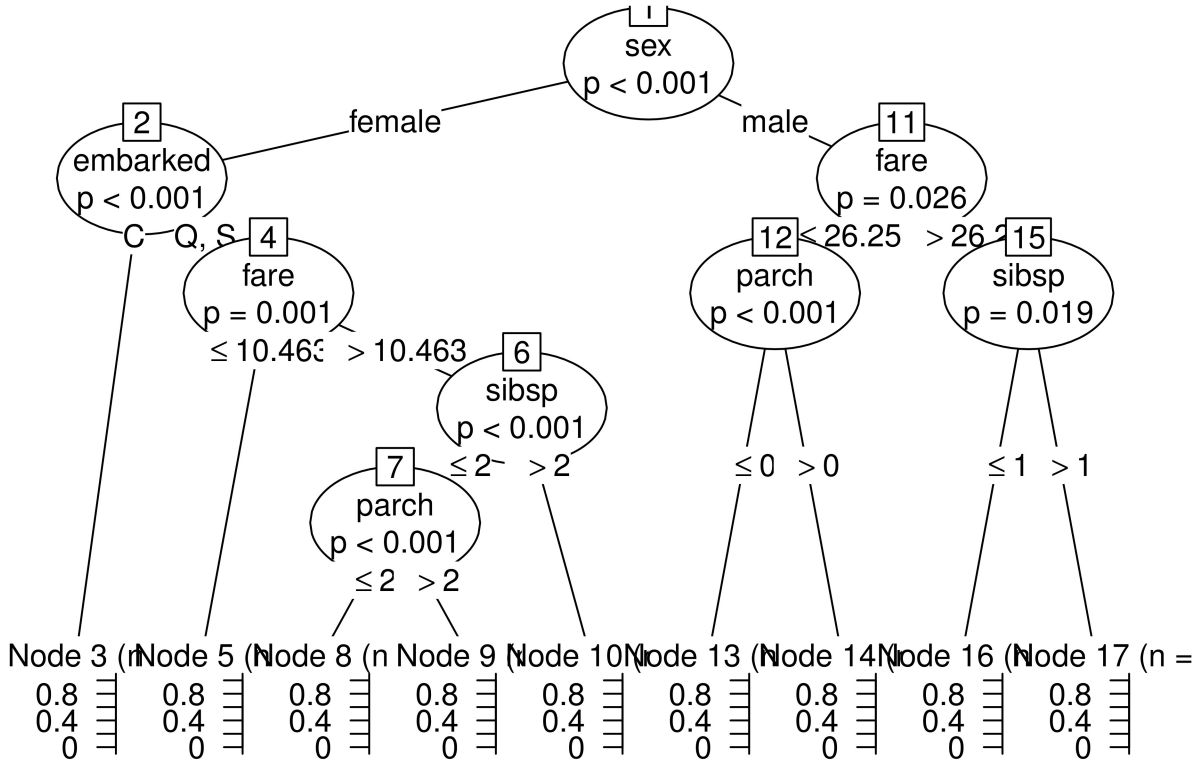
```
titanic_data <- "https://goo.gl/At238b" %>%  #DataFlair
  read.csv %>% # read in the data
  select(survived, embarked, sex,
         sibsp, parch, fare) %>%
  mutate(embarked = factor(embarked),
         sex = factor(sex))
```

```
set.seed(123)
sample_data = sample.split(titanic_data, SplitRatio = 0.75)
train_data <- subset(titanic_data, sample_data == TRUE)
test_data <- subset(titanic_data, sample_data == FALSE)
```

```
rtree <- rpart(survived ~ ., train_data)
rpart.plot(rtree)
```




```
ctree_ <- ctree(survived ~ ., train_data)
plot(ctree_)
```




```
rm(list=ls())
library(nnet)
library(rpart)
library(rpart.plot)
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 3.6.1
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
library(gbm)
```

```
## Warning: package 'gbm' was built under R version 3.6.1
```

```
## Loaded gbm 2.1.5
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.1
```

```
## Loading required package: lattice
```

```
## Loading required package: ggplot2
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
##   print.quosures rlang
```

```
##
## Attaching package: 'ggplot2'
```

```
## The following object is masked from 'package:randomForest':
##
##     margin
```

```
library(AER)
```

```
## Warning: package 'AER' was built under R version 3.6.1
```

```
## Loading required package: car
```

```
## Warning: package 'car' was built under R version 3.6.1
```

```
## Loading required package: carData
```

```
## Warning: package 'carData' was built under R version 3.6.1
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:modeltools':
##
##     Predict
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## Loading required package: lmtest
```

```
## Warning: package 'lmtest' was built under R version 3.6.1
```

```
## Loading required package: survival
```

```
##
## Attaching package: 'survival'
```

```
## The following object is masked from 'package:caret':
##
##     cluster
```

```
# Simulation
set.seed (10)
n = 1000
x = matrix(rnorm(n*2),ncol=2)
z = 1.6*x[,1] + 7.7*x[,2]
p = exp(z)/(1+exp(z)) #p(y=1)
u = runif(n)
y = (p>u)
```

```
# create training and test set
data = data.frame(x1=x[,1],x2=x[,2],y=as.factor(y))
train = sample(n,n*0.4)
```

```r
data_train = data[train,]
data_test = data[-train,]
ytrue = data_test[,"y"]
```

```r
########################
# Logistic Regression #
########################
set.seed(100)
fit = multinom(y~.,data=data_train)
```

```
## # weights:  4 (3 variable)
## initial  value 277.258872
## final  value 62.111635
## converged
```

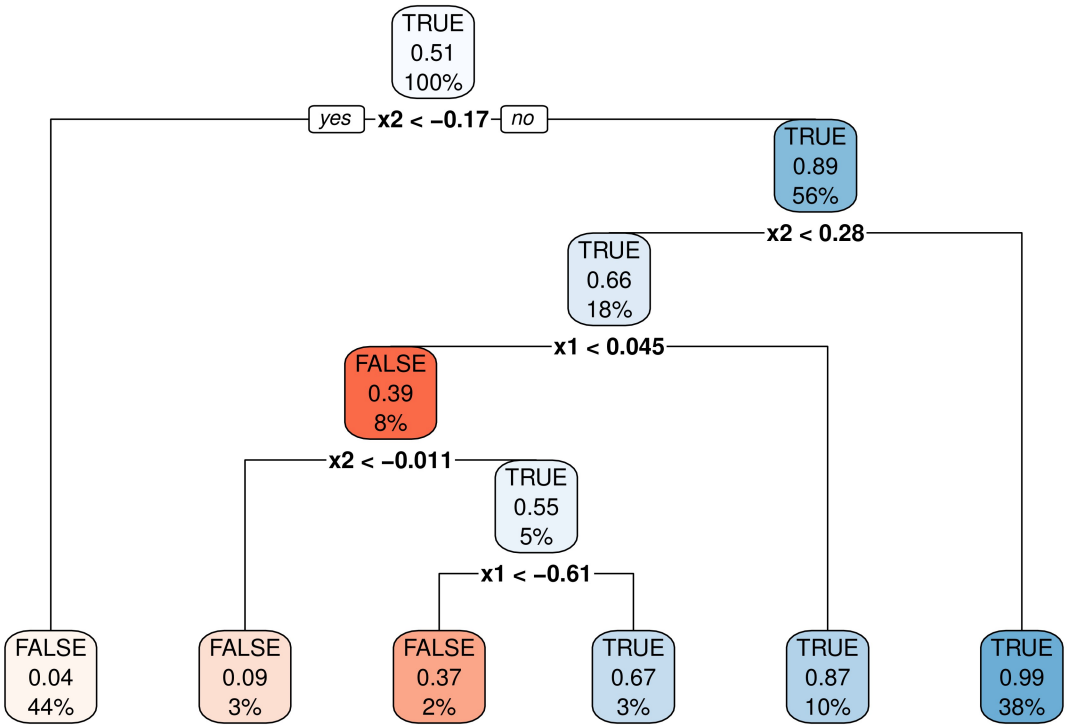
```r
coeftest(fit)
```

```
##
## z test of coefficients:
##
##             Estimate Std. Error z value  Pr(>|z|)
## (Intercept)  0.25748    0.23567  1.0925    0.2746
## x1           1.27408    0.28270  4.5068 6.580e-06 ***
## x2           7.62612    0.99892  7.6344 2.269e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
yhat = predict(fit,data_test)
err = 1-mean(ytrue==yhat)
err
```

```
## [1] 0.06833333
```

```r
########################
# Classification Tree #
########################
set.seed(100)
fit0 = rpart(y~.,data_train,control=rpart.control(cp=0))
fit = prune(fit0,cp=fit0$cptable[which.min(fit0$cptable[,"xerror"]),"CP"])
rpart.plot(fit,box.palette=c("Reds","Blues"))
```




```
# test err
yhat = predict(fit,data_test,type="class")
err = 1-mean(ytrue==yhat)
err
```

```
## [1] 0.06333333
```

```
############
# Bagging #
############
set.seed(100)
fit = randomForest(y~.,data=data_train,mtry=2,maxnodes=8)
fit
```

```
##
## Call:
##  randomForest(formula = y ~ ., data = data_train, mtry = 2, maxnodes = 8)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 2
##
##         OOB estimate of  error rate: 8.5%
## Confusion matrix:
##       FALSE TRUE class.error
## FALSE   179   16  0.08205128
## TRUE     18  187  0.08780488
```

```r
# test err
yhat = predict(fit,data_test)
err = 1-mean(ytrue==yhat)
err
```

```
## [1] 0.05666667
```

```r
#############
# Boosting #
#############
set.seed(100)
ntree = 4000
data_boost = transform(data_train, y = as.numeric(y) - 1)
fit = gbm(y~.,data=data_boost,distribution="adaboost",
          n.trees=ntree,
          interaction.depth = 2,
          shrinkage = 0.001)
fit
```

```
## gbm(formula = y ~ ., distribution = "adaboost", data = data_boost,
##     n.trees = ntree, interaction.depth = 2, shrinkage = 0.001)
## A gradient boosted model with adaboost loss function.
## 4000 iterations were performed.
## There were 2 predictors of which 2 had non-zero influence.
```

```r
# test error
phat = predict(fit,data_test,n.trees=ntree,type="response")
yhat = (phat>0.5)
err = 1-mean(yhat==ytrue)
err
```

```
## [1] 0.07166667
```

```r
#############
# Piecewise Linear Tree#
#############
#library(RWeka)
#fit <- M5P(y~., data=data_train)
#summary(fit)
#predictions <- predict(fit, data_test)
#mse <- mean((ytrue - predictions)^2)
#print(mse)
```