HW3

20989977 Zhang Mingtao

2023/11/10

0.

```
library(reticulate)
```

1.

```
#(1)
#1
# The cubic spline model for a set of data points (x1, y1), (x2, y2), ..., (xn, yn) and knots ξ1, ξ2, ..., ξK can be written
# as follows:
# For each interval [ξi, ξi+1], we fit a cubic polynomial of the form:
# f_i(x) = a_i0 + a_i1*x + a_i2*x^2 + a_i3*x^3
#
# We then apply the continuity and linearity constraints at each knot ξi:
#
# Continuity Constraints:
# f_i(ξi) = f_i+1(ξi)
# f_i'(ξi) = f_i+1'(ξi)
# f_i''(ξi) = f_i+1''(ξi)
#
# And also Linearity Constraints at endpoints (-∞, ξ1) and (ξK, ∞).
# The objective function can be a least squares error.
# Now the problem becomes a linear regression problem with equality constraints.
# We can solve the coefficients a_i, a_i1, a_i2, a_i3, subject to these equality constraints.

#2
# See in the picture 详见手写证明过程

#3
# See in the picture 详见手写证明过程

#4
# Piecewise polynomial regression divides the data set into multiple segments,
# and uses a polynomial function to fit each segment. The fitting of each segment is performed independently;
#
# Local polynomial regression is a non-parametric method that uses the neighbors near the point to fit the point,
# based on weights in the neighborhood, which is smoother but more complex.
#
# My understanding is: piecewise polynomial regression is suitable for situations
# where there are clear intervals between variables with different polynomial behavior;
# while local polynomial regression is more suitable for capturing smooth local changes in data that are not clearly segmented.

#5
# a. Lack of Global Information: Since local reference method regression only focuses on local data points,
# it usually cannot provide global information about the entire data set.
#
# b. Non-Parametric Nature: Local polynomial regression is a non-parametric regression method that
# does not rely on a specific functional form.
# Compared with traditional parametric models like linear regression, the parameters are less interpretable.
#
# c. Bandwidth Selection: Bandwidth Selection further complicates interpretation.

#6
# For a local demonstration regression model, assume that we use the least squares method for d
```

```
emonstration:
# y_hat(x) = Σ(1~n) w_i(x)*y_i
# Bias(x) = E[y_hat(x)] - y
# Variance(x) = E[(y_hat(x) - E[y_hat(x)])^2]

# When increasing the bandwidth h:
#
# Change in deviation:
# Increasing the bandwidth h makes the model smoother, it uses more neighbor data points to fi
t.
# This means that the predicted value y_hat(x) is more biased, reducing the model's ability to
capture local details.
#
# Change in variance:
# Increasing the bandwidth h reduces the model's sensitivity to neighbor data points, thus redu
cing the model's variance.
# When the bandwidth increases, the range of data points covered by the model's weight function
w_i(x) increases,
# and the model is relatively insensitive to changes in neighbor data points.
#
# Conclusion: h↑, Bias↑, Var↓

#7
# I think Regression tree is most similar to option (b) Piecewise constant regression:
# Regression tree and piecewise constant regression both achieve modeling of nonlinear relation
ships
# by dividing the feature space into multiple regions and using different constant values for f
itting.
#
# differences:
# Piecewise constant regression is a simple regression method that divides the feature space in
to several segments
# and fits a constant value in each segment. Its model is relatively simple, can only capture p
iecewise linear relationships,
# and cannot handle the complex structure of the feature space.
#
# Regression tree is a non-parametric supervised learning method that divides the feature space
# into a series of rectangular regions and fits a constant value in each region.
# Regression trees are more flexible and can capture nonlinear relationships and interactions
# by continuously dividing the feature space, so they perform well when dealing with nonlinear
data and complex relationships.
# And it can automatically determine the location of the division based on the distribution of
the data,
# while piecewise constant regression requires manually specifying the location of the divisio
n.

#8
# Increasing complexity reduces bias: the model is better able to capture approximate features
and relationships
# in the training data and can more accurately adapt to the complexity of the data, thus reduci
ng bias.
#
# Increasing complexity increases variance: Model complexity causes the model to be very sensit
ive to small changes
# in training data, and the difference in predictions on different training data sets increase
s, thus increasing variance.
```

```
#9
# Linear regression:
# If the true relationship between the predictor and response variables is indeed linear,
# linear regression can accurately capture this relationship, with less bias, and provide a goo
d fit to the true relationship.
#
# Regression tree:
# Regression trees are more suitable for handling non-linear and piecewise relationships.
# If the true relationship is linear, the regression tree may have difficulty capturing it accu
rately,
# so the deviation can be large. In this case, the regression tree is often not as accurate as
linear regression.
# Even though regression trees can capture complex non-linear relationships, there can be high
bias since the
# decision boundaries created by a single tree structure may not exactly match the true underly
ing relationships.
#
# Natural splines:
# Natural splines is a form of nonlinear regression that captures nonlinear patterns in data.
# The deviation of natural splines depends on the complexity of the spline function and the num
ber of nodes used.
# When the number of nodes is limited or the complexity of the spline function is low, especial
ly when the true relationship
# is close to linear, natural splines may have higher deviation.
#
# To sum up, in general, regression tree has relatively the smallest bias if data nonlinear, or
linear regression is better.

#10
# The variable importance measurement mainly uses "average decrease in accuracy" and "average d
ecrease of Gini index".
#
# "The average reduction in accuracy" means that for each tree model: predict the cases outside
the bag and
# obtain the accuracy; then randomly arrange the values of a certain variable on the cases outs
ide the bag,
# and then bring them in the model to obtains the accuracy, and obtains the difference between
the two accuracy rates.
# Average the accuracy differences of all trees on this variable to get the "average decrease i
n accuracy".
#
# For each tree, IM_Gini(Rm) = ∑k!=k'(p^mk*p^mk') = K∑k=1(p^mk(1 ??? p^mk)) = 1 ??? K∑k=1(p^
mk^2)
#
# The average decrease in the Gini index refers to, for a tree, obtaining the decrease in the G
ini index
# caused by a certain variable acting as a splitting variable on each node;
# summing up the decreases in the Gini index of the variable in all trees and dividing Based on
the number of trees,
# then the "average reduction of Gini index" is obtained.
#
# In the variable section/model selection part, we prefer variables with higher importance.
```

2.

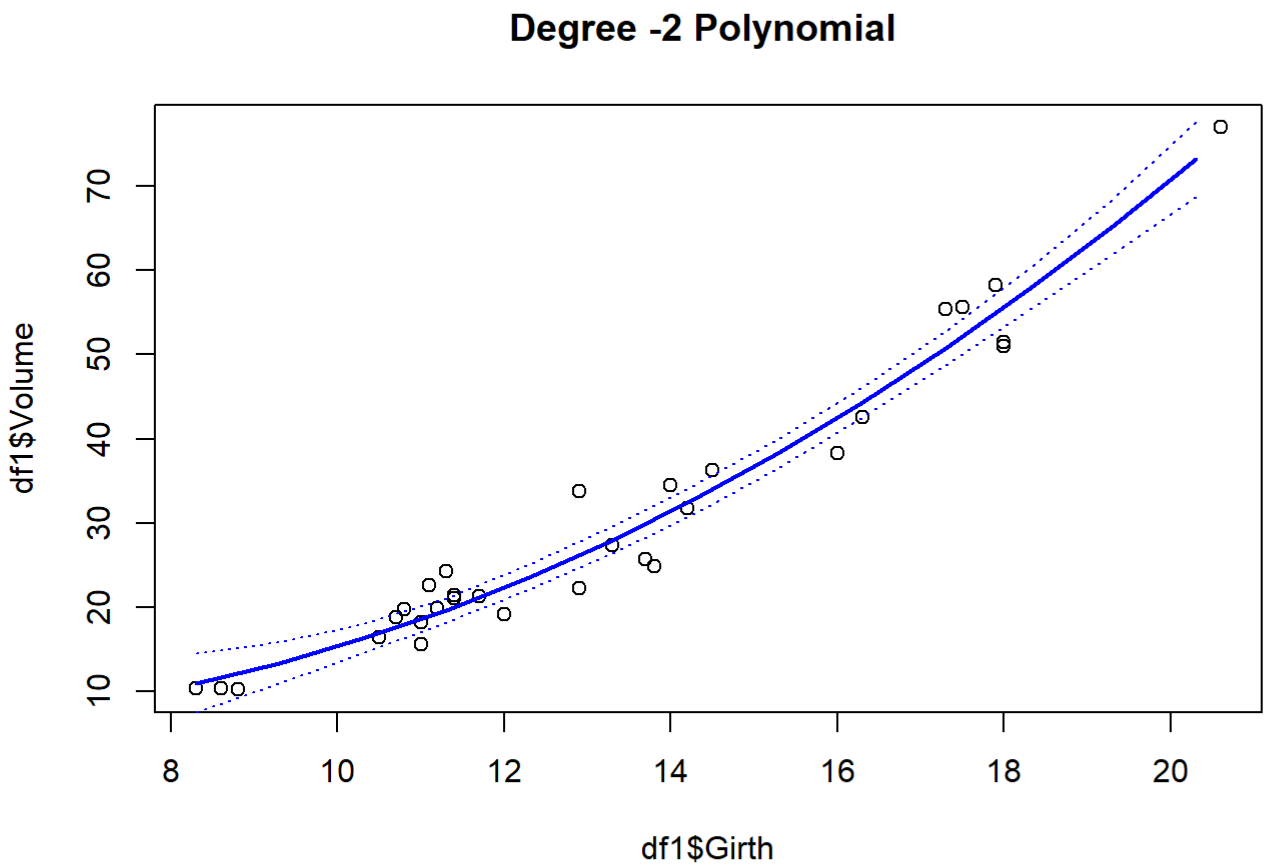
```
#(2)
#1
df1=read.csv("C://Users//张铭韬//Desktop//学业//港科大//MSDM5054机器学习//作业//hw3//trees.csv")
fit.1=lm(Volume ~ poly(Girth, 1, raw = TRUE), data = df1)
fit.2=lm(Volume ~ poly(Girth, 2, raw = TRUE), data = df1)
fit.3=lm(Volume ~ poly(Girth, 3, raw = TRUE), data = df1)
fit.4=lm(Volume ~ poly(Girth, 4, raw = TRUE), data = df1)

adj_rsq1=summary(fit.1)$adj.r.squared
adj_rsq2=summary(fit.2)$adj.r.squared
adj_rsq3=summary(fit.3)$adj.r.squared
adj_rsq4=summary(fit.4)$adj.r.squared

max(c(adj_rsq1, adj_rsq2, adj_rsq3, adj_rsq4))
```

```
## [1] 0.9588428
```

```
which.max(c(adj_rsq1, adj_rsq2, adj_rsq3, adj_rsq4))
```

```
## [1] 2
```

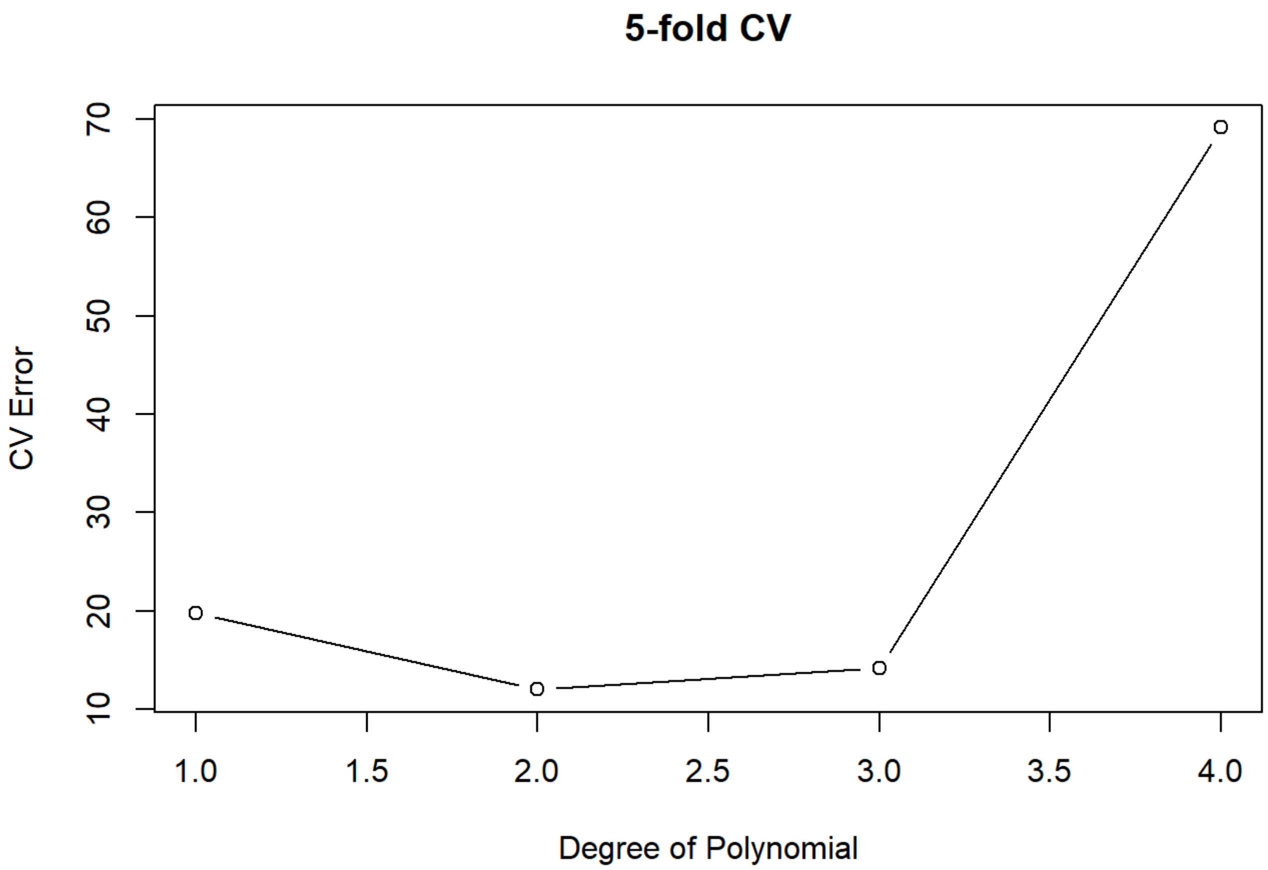
```
# deg = 2

## prediction on all range of age, and confidence bands
Girthlims=range(df1$Girth)
Girth.grid=seq(from=Girthlims[1],to=Girthlims[2])
preds=predict(fit.2,newdata=list(Girth=Girth.grid),se=TRUE)
se.bands=cbind(preds$fit+2*preds$se.fit,preds$fit-2*preds$se.fit)

par(mfrow=c(1,1))
plot(df1$Girth,df1$Volume,xlim=Girthlims ,cex=1,col="black")
title("Degree -2 Polynomial ",outer=F)
lines(Girth.grid,preds$fit,lwd=2,col="blue")
matlines(Girth.grid,se.bands,lwd=1,col="blue",lty=3)
```



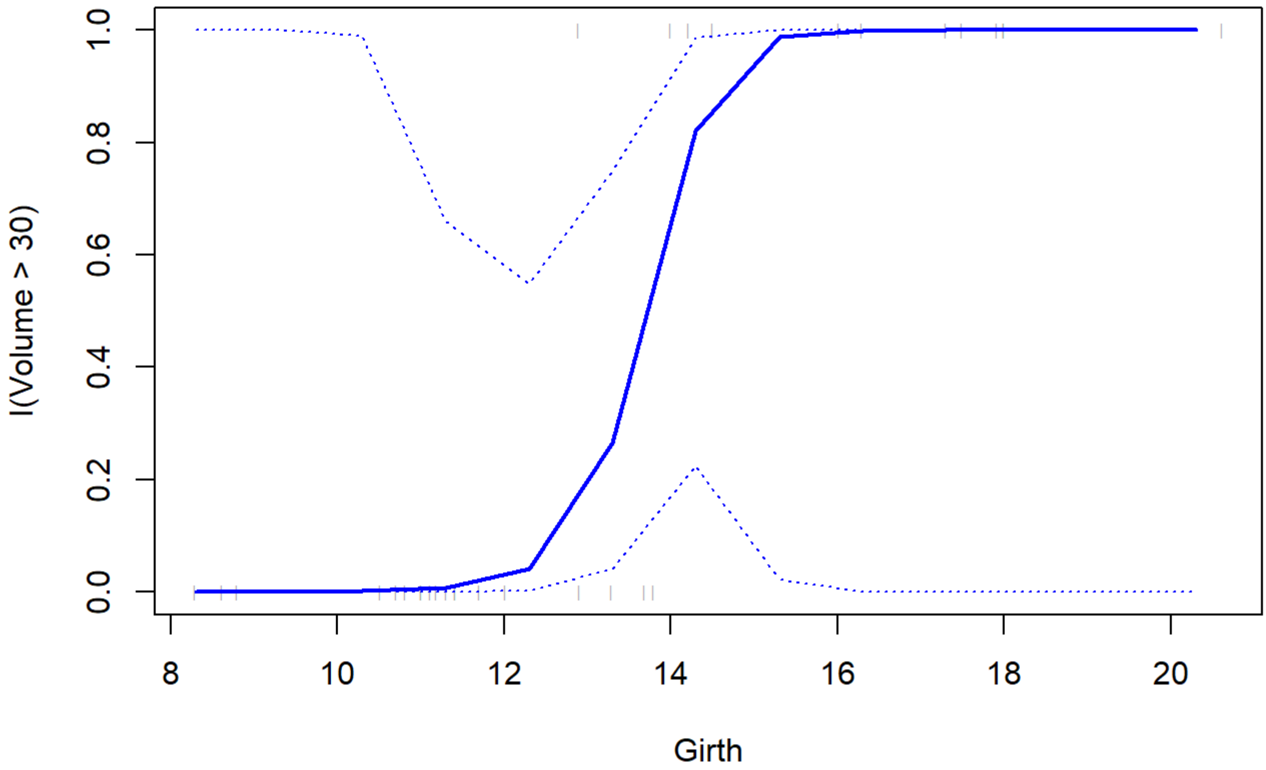

```
#### choosing the model using 5-CV error:
library(boot)

cv.error=rep(0,4)
for (i in 1:4){
  set.seed(1)
  glm.fit=glm(Volume ~ poly(Girth,i, raw = TRUE),data=df1)
  cv.error[i]=cv.glm(df1,glm.fit,K=5)$delta[1]
}
cv.error
```

```
## [1] 19.76372 12.05697 14.21523 69.10362
```

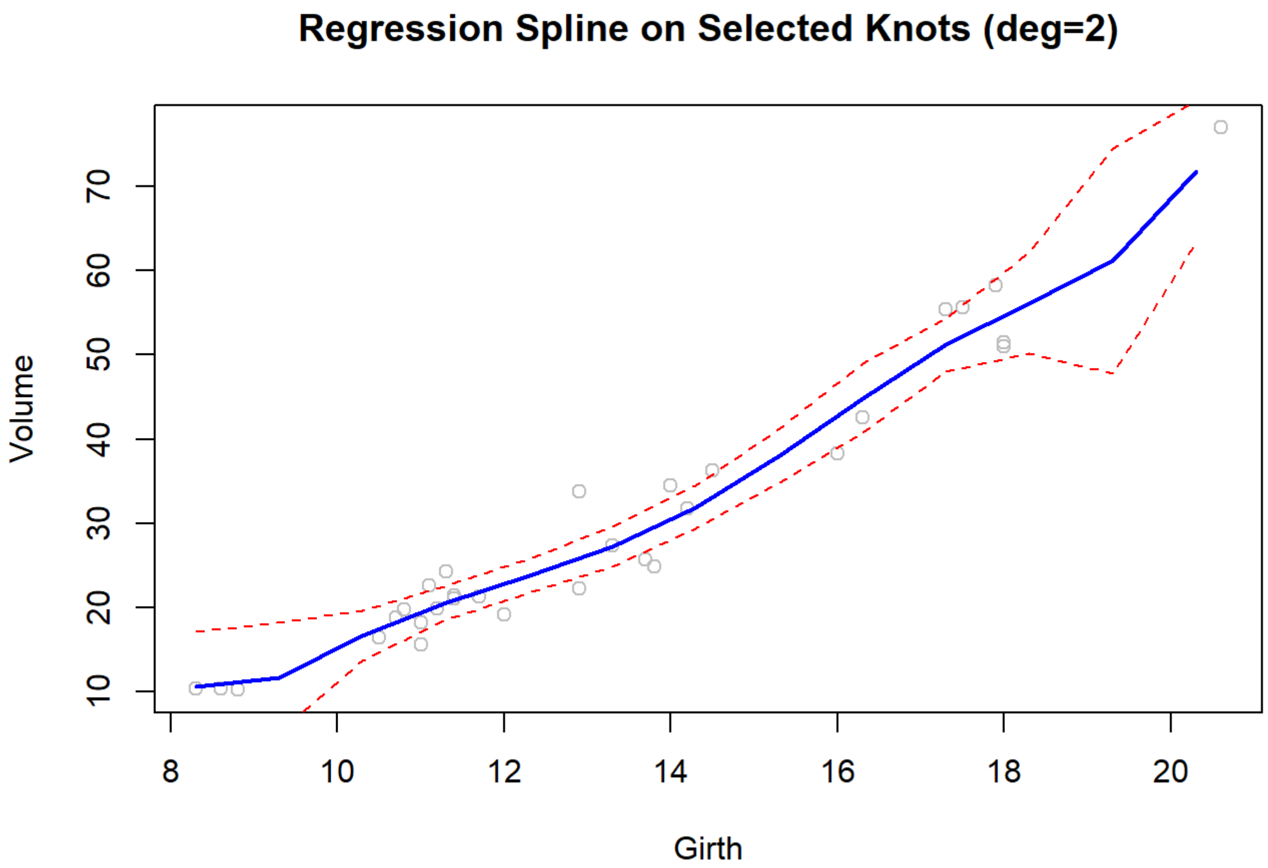
```
plot(1:4,cv.error,type='b',xlab="Degree of Polynomial",ylab="CV Error",main="5-fold CV")
```




```
#We still choose the i=2 model.

#2
Girth=df1$Girth
Volume=df1$Volume

fit=glm(I(Volume>30)~poly(Girth,2),data=df1,family=binomial)
preds=predict(fit,newdata=list(Girth=Girth.grid),se=T)          ## predict on all the age v
alues
pfit=exp(preds$fit)/(1+exp(preds$fit))
se.bands.logit = cbind(preds$fit+2*preds$se.fit, preds$fit-2*preds$se.fit)
se.bands = exp(se.bands.logit)/(1+exp(se.bands.logit))

preds=predict(fit,newdata=list(Girth=Girth.grid),type="response", se=T)

##### plot the confidence bands
plot(Girth,I(Volume>30),xlim=Girthlims ,type="n")
points(jitter(Girth), I(Volume>30),cex=.5,pch="|",col =" darkgrey ")
lines(Girth.grid,pfit,lwd=2, col="blue")
matlines(Girth.grid,se.bands,lwd=1,col="blue",lty=3)
```




```
#3
library(splines)
fit2=lm(Volume~bs(Girth,knots=c(10,14,18),df=2),data=df1)
pred2=predict(fit2,newdata=list(Girth=Girth.grid),se=T)
plot(Girth,Volume,col="gray",main="Regression Spline on Selected Knots (deg=2)")
lines(Girth.grid,pred2$fit,lwd=2,col="blue")
lines(Girth.grid,pred2$fit+2*pred2$se ,lty="dashed",col="red")
lines(Girth.grid,pred2$fit-2*pred2$se ,lty="dashed",col="red")
```




```
#4
library(caret)
```

```
## 载入需要的程辑包：ggplot2
```

```
## 载入需要的程辑包：lattice
```

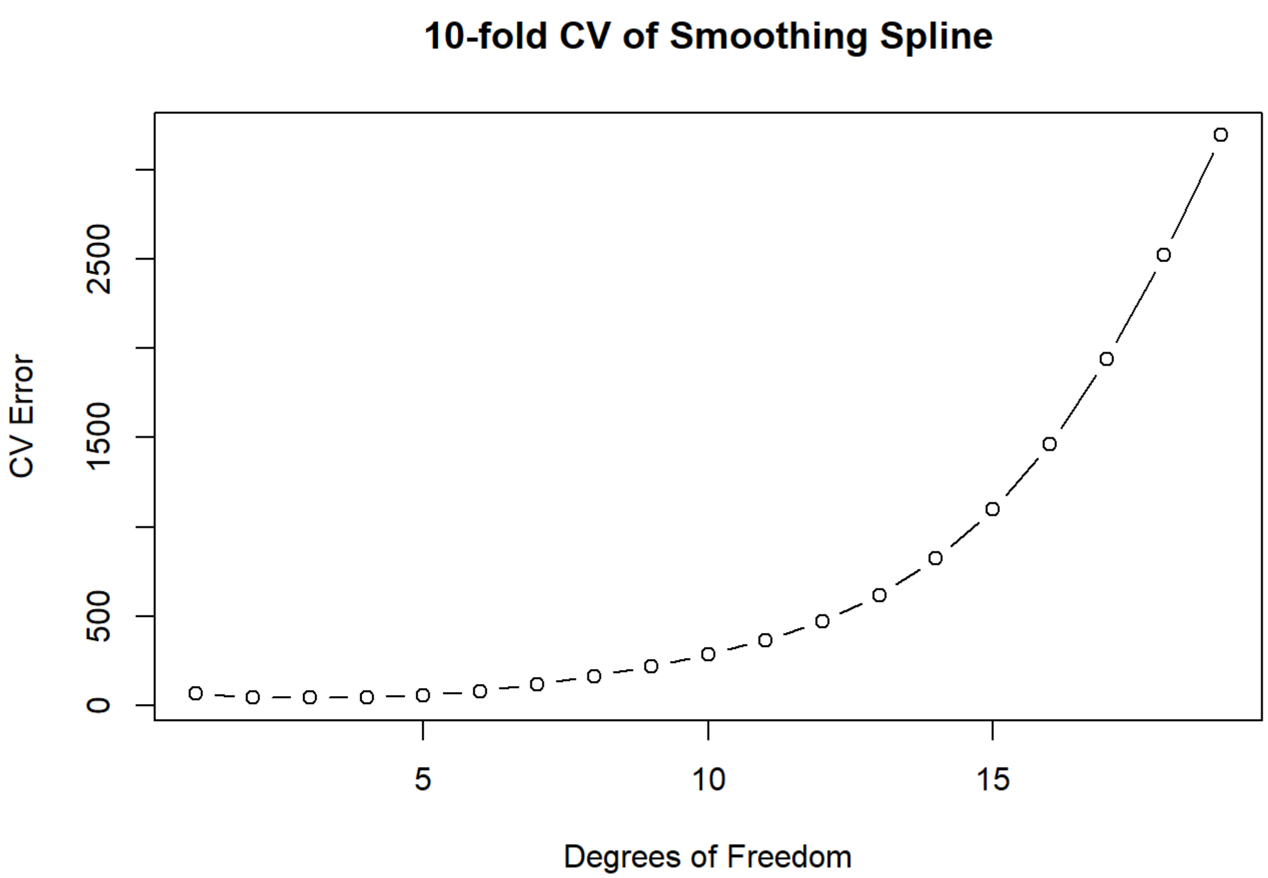
```
##
## 载入程辑包：'lattice'
```

```
## The following object is masked from 'package:boot':
##
##     melanoma
```

```
#########################   10-fold CV of Smoothing Spline
X=data.frame(Girth=df1$Girth)
y=df1$Volume
flds=createFolds(1:31, k = 10, list = TRUE, returnTrain = FALSE)    ### random split into 10 fo
lds
CVErr=rep(0,19)
for(k in 2:20){
  err=0
  for(fold in flds){
    fit=smooth.spline(X[-fold,], y[-fold],df=k)
    test_y=predict(fit,X[fold,])
    err=err+sum((test_y$y-y[fold])^2)
  }
  CVErr[k-1]=err/10
}
CVErr
```

```
##  [1]   65.07661   44.76987   42.22764   45.71116   56.28404   79.03316
##  [7]  115.86790  163.77374  219.29211  283.91232  364.41421  471.42887
## [13]  618.49793  821.25485 1097.35025 1463.95697 1939.48570 2523.73131
## [19] 3195.24379
```

```
plot(1:19,CVErr,type='b',xlab="Degrees of Freedom",ylab="CV Error",main="10-fold CV of Smoothin
g Spline")
```



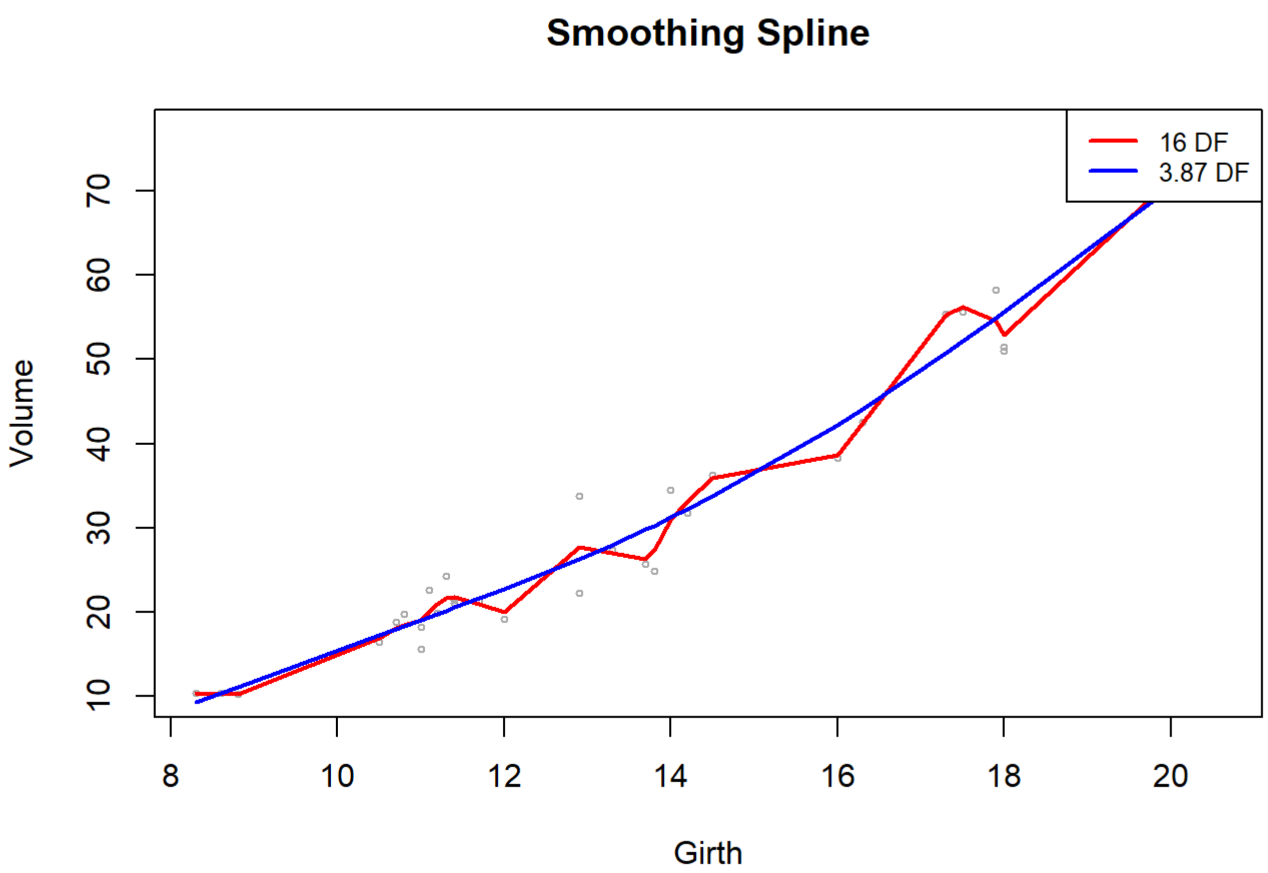
```
plot(Girth,Volume,xlim=Girthlims,cex=.5,col="darkgrey",main=" Smoothing Spline ")
fit=smooth.spline(Girth,Volume,df=16)
fit2=smooth.spline(Girth,Volume,cv=TRUE)
```

```
## Warning in smooth.spline(Girth, Volume, cv = TRUE): cross-validation with non-
## unique 'x' values seems doubtful
```

```
fit2$df # degree=3.87138
```

```
## [1] 3.87138
```

```
lines(fit,col="red",lwd=2)
lines(fit2,col="blue",lwd=2)
legend("topright",legend=c("16 DF","3.87 DF"),col=c("red","blue"),lty=1,lwd=2,cex=.8)
```

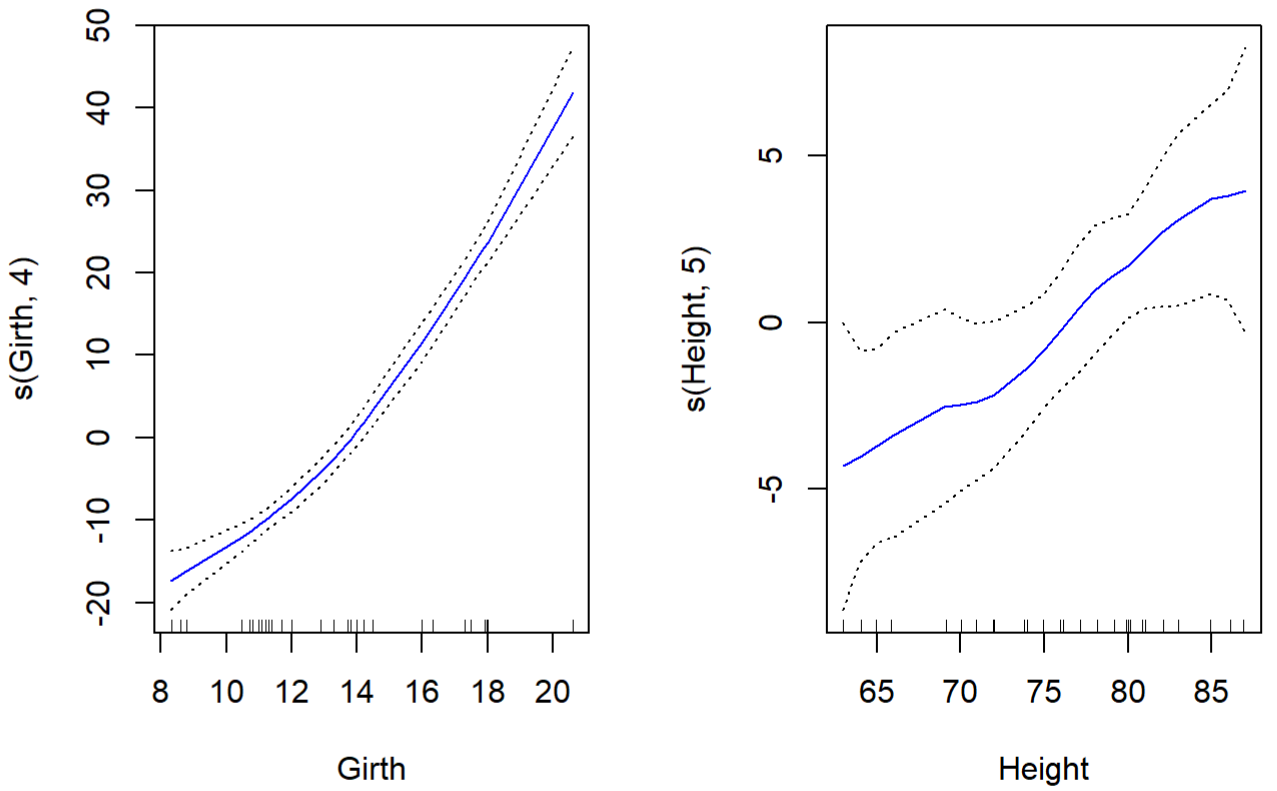



```
#5
library(gam)
```

```
## 载入需要的程辑包：foreach
```

```
## Loaded gam 1.20.1
```

```
gam.m3=gam(Volume~s(Girth,4)+s(Height,5),data=df1)
par(mfrow=c(1,2))
plot(gam.m3, se=TRUE,col="blue")
```



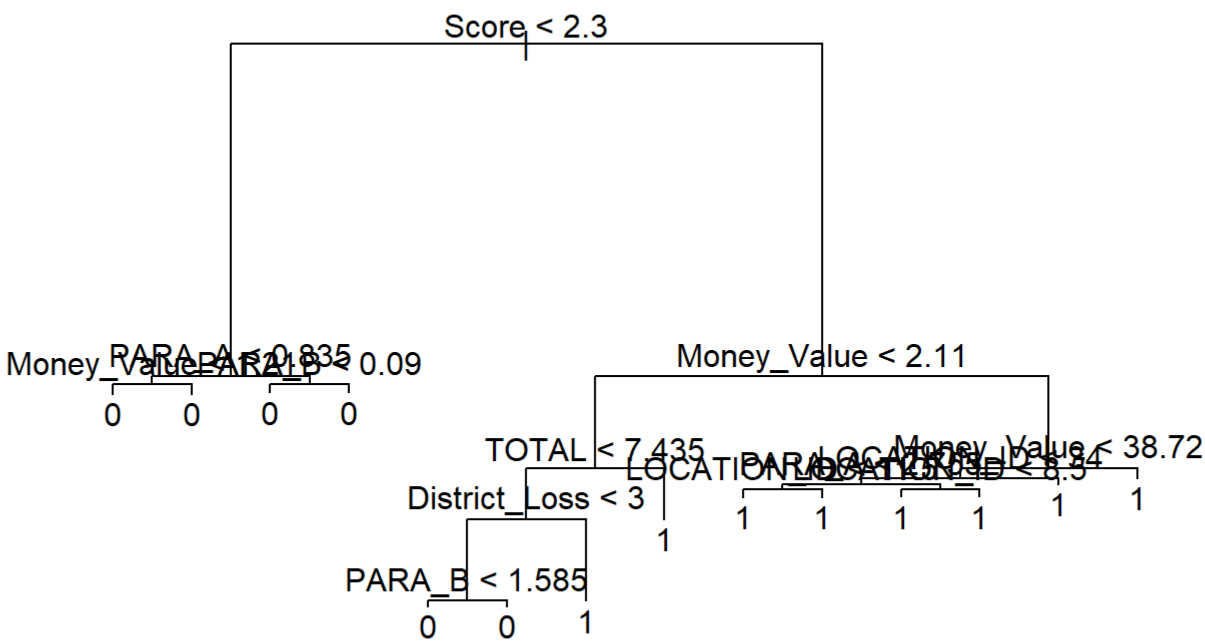

```
par(mfrow=c(1,1))
```

3.

```
#(3)
#1
traindf=read.csv("C://Users//张铭韬//Desktop//学业//港科大//MSDM5054机器学习//作业//hw3//audit_train.csv")
testdf=read.csv("C://Users//张铭韬//Desktop//学业//港科大//MSDM5054机器学习//作业//hw3//audit_test.csv")

traindf=na.omit(traindf)
testdf=na.omit(testdf)

traindf$Risk = as.factor(traindf$Risk)
testdf$Risk = as.factor(testdf$Risk)

library(tree)
audittree=tree(Risk~.,traindf,control =tree.control(dim(traindf)[1],mindev=0.005,minsize=40))
summary(audittree)
```

```
##
## Classification tree:
## tree(formula = Risk ~ ., data = traindf, control = tree.control(dim(traindf)[1],
##     mindev = 0.005, minsize = 40))
## Variables actually used in tree construction:
## [1] "Score"         "PARA_A"        "Money_Value"   "PARA_B"
## [5] "TOTAL"         "District_Loss" "LOCATION_ID"
## Number of terminal nodes:  14
## Residual mean deviance:  0.4045 = 226.9 / 561
## Misclassification error rate: 0.06957 = 40 / 575
```

```
plot(audittree)
text(audittree,pretty=0)
```




```
# Misclassification error rate: 0.06957 = 40 / 575

predp=predict(audittree,testdf)

pred=rep(0,dim(testdf)[1])
pred[predp[,2]>0.5]=1

table(pred,testdf$Risk)
```

```
##
## pred   0   1
##    0 103   5
##    1   6  83
```

```
length(which(pred==testdf$Risk))/dim(testdf)[1] # accuracy = 0.9441624
```

```
## [1] 0.9441624
```

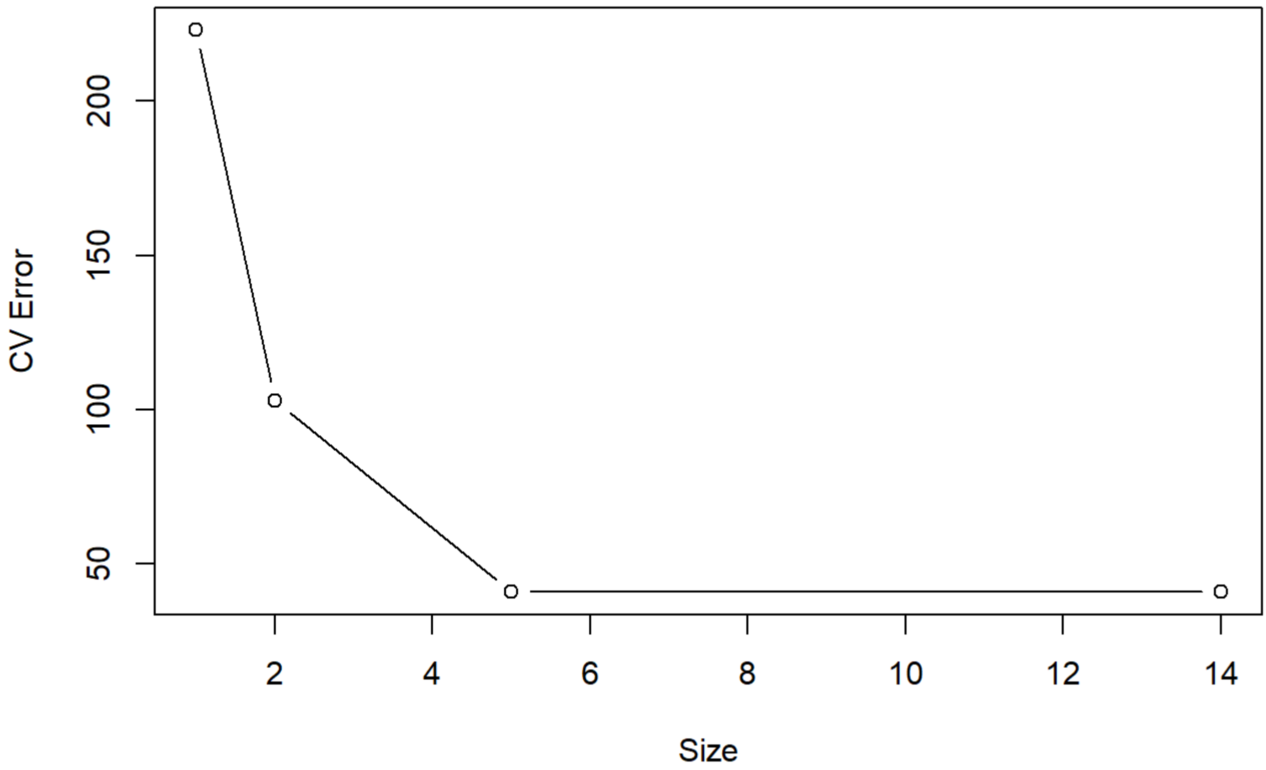
```
#2
cv.audittree = cv.tree(audittree,FUN=prune.misclass)
names(cv.audittree)
```

```
## [1] "size"   "dev"    "k"      "method"
```

```
cv.audittree
```

```
## $size
## [1] 14  5  2  1
##
## $dev
## [1]  41  41 103 223
##
## $k
## [1] -Inf    0   20  123
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

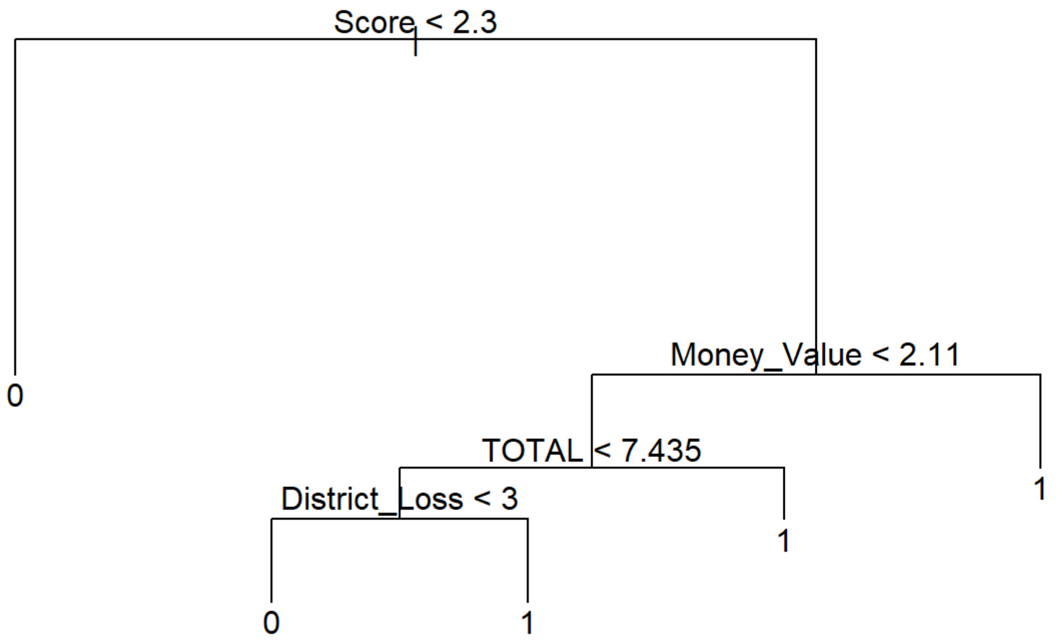
```
plot(cv.audittree$size ,cv.audittree$dev ,type="b",xlab="Size",ylab="CV Error")
```




```
# size = 5

audittree_pruned=prune.tree(audittree,best=5)
summary(audittree_pruned)
```

```
##
## Classification tree:
## snip.tree(tree = audittree, nodes = c(24L, 2L, 7L))
## Variables actually used in tree construction:
## [1] "Score"         "Money_Value"   "TOTAL"         "District_Loss"
## Number of terminal nodes:  5
## Residual mean deviance:  0.5013 = 285.8 / 570
## Misclassification error rate: 0.06957 = 40 / 575
```

```
plot(audittree_pruned)
text(audittree_pruned,pretty=0)
```




```
predp2=predict(audittree_pruned,testdf)

pred2=rep(0,dim(testdf)[1])
pred2[predp2[,2]>0.5]=1

table(pred2,testdf$Risk)
```

```
##
## pred2   0   1
##     0 103   5
##     1   6  83
```

```
length(which(pred2==testdf$Risk))/dim(testdf)[1] # accuracy = 0.9441624
```

```
## [1] 0.9441624
```

```
#3
library(randomForest)
```

```
## randomForest 4.7-1.1
```

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

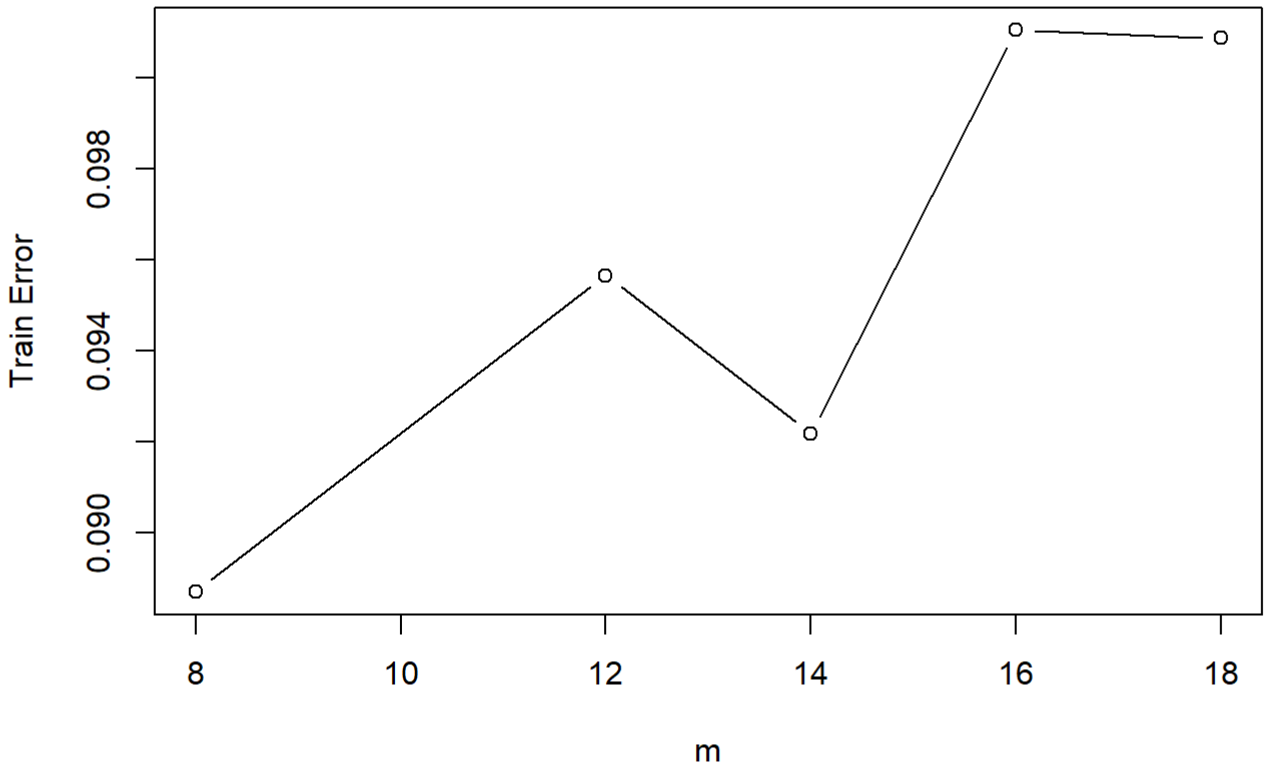
```
##
## 载入程辑包：'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(1)
rf.audit=randomForest(Risk~.,data=traindf,mtry=13,ntree=25,importance=T,proximity=T,na.action=na.omit)
rf.audit
```

```
##
## Call:
##  randomForest(formula = Risk ~ ., data = traindf, mtry = 13, ntree = 25,      importance = T, proximity = T, na.action = na.omit)
##                Type of random forest: classification
##                      Number of trees: 25
## No. of variables tried at each split: 13
##
##         OOB estimate of  error rate: 9.91%
## Confusion matrix:
##     0   1 class.error
## 0 326  26  0.07386364
## 1  31 192  0.13901345
```

```
# OOB estimate of  error rate: 9.91%
#     0   1 class.error
# 0 326  26  0.07386364
# 1  31 192  0.13901345




#4
error = c()

mlist = c(8,12,14,16,18)

for (m in mlist) {
  set.seed(1)
  rfmodel=randomForest(Risk~.,data=traindf,mtry=m,ntree=25,importance=T,proximity=T,na.action=na.omit)
  error=c(error,rfmodel$err.rate[25,1])
}

plot(mlist,error,type="b",xlab="m",ylab="Train Error")
```




```
error # m = 8 min
```

```
##        OOB        OOB        OOB        OOB        OOB
## 0.08869565 0.09565217 0.09217391 0.10104530 0.10086957
```

```
set.seed(1)
rfmodel2 = randomForest(Risk~.,data=traindf,mtry=8,ntree=25,importance=T,proximity=T,na.action=
na.omit)

yhat.rf = predict(rfmodel2,newdata=testdf,type="class")
table(yhat.rf,testdf$Risk)
```

```
##
## yhat.rf   0   1
##       0 106   3
##       1   3  85
```

```
length(which(yhat.rf==testdf$Risk))/dim(testdf)[1] # accuracy = 0.9695431
```

```
## [1] 0.9695431
```

```
rfmodel2$importance # Score, Risk_D, TOTAL, Money_Value
```

```
##                           0             1 MeanDecreaseAccuracy
## Sector_score   -0.0081305665 -0.0004347395        -0.0052540876
## LOCATION_ID    -0.0023425774 -0.0053865255        -0.0038657037
## PARA_A          0.0910842104  0.0427982612         0.0733307664
## Score_A         0.0020888573 -0.0001472557         0.0011496118
## Risk_A          0.0904383377  0.0273629273         0.0653068315
## PARA_B          0.0534399308  0.0052693335         0.0345709968
## Score_B         0.0025934066  0.0039339501         0.0032661212
## Risk_B          0.0370048783  0.0334081231         0.0355870428
## TOTAL           0.1081393987  0.0354221044         0.0801110173
## numbers         0.0012307692 -0.0010536263         0.0003756044
## Score_B.1       0.0000000000 -0.0009090909        -0.0003669725
## Risk_C          0.0000000000  0.0000000000         0.0000000000
## Money_Value     0.0375234096  0.0030790955         0.0239149941
## Score_MV        0.0361666840  0.0050456933         0.0237879280
## Risk_D          0.0575986924  0.0507693238         0.0543476774
## District_Loss   0.0090290982  0.0088654011         0.0090042431
## PROB            0.0000000000  0.0000000000         0.0000000000
## RiSk_E          0.0222600922  0.0194261222         0.0211090303
## History         0.0000000000 -0.0019883992        -0.0007634787
## Prob           -0.0003149606 -0.0009038662        -0.0005563926
## Risk_F          0.0000000000 -0.0009697326        -0.0003703704
## Score           0.0726820708  0.1318342378         0.0959364838
## CONTROL_RISK    0.0440693149  0.0337509536         0.0403278143
## Detection_Risk  0.0000000000  0.0000000000         0.0000000000
##                MeanDecreaseGini
## Sector_score          5.2855731
## LOCATION_ID           9.8626726
## PARA_A               17.3416683
## Score_A               0.9589384
## Risk_A               14.3612160
## PARA_B                9.9255645
## Score_B               0.6798054
## Risk_B                7.6124064
## TOTAL                32.1905116
## numbers               1.2937736
## Score_B.1             0.2736190
## Risk_C                0.0400000
## Money_Value          20.1516109
## Score_MV             17.4396505
## Risk_D               45.8337950
## District_Loss         3.4036572
## PROB                  0.0800000
## RiSk_E               10.4137235
## History               0.4920710
## Prob                  0.3291569
## Risk_F                0.5592363
## Score                53.1050277
## CONTROL_RISK         19.5085789
## Detection_Risk        0.0000000
```

```
#5
# single tree: accuracy = 0.9441624
# random forest: accuracy = 0.9695431 , which is better than a single tree. The model built on
decision trees is less likely
# to overfit after selecting appropriate parameters, which can reduce errors and deviations, an
d improve accuracy.
# Due to the combination of multiple decision trees, diversity is effectively considered, and t
he combined result is
# better than the result of a single tree。

# summary(audittree_pruned):    "Score" "Money_Value" "TOTAL"  "District_Loss"
# rfmodel2$importance:           Score,  Money_Value , TOTAL ,  Risk_D
# 3 of them are the same.
```

4.

```
#(4)
# see the python code since I'm not able to define a class in R but I can do that in python.
#(4)

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

train = pd.read_csv("C://Users//张铭韬//Desktop//学业//港科大//MSDM5054机器学习//作业//hw3//Hitters_train.csv")
test = pd.read_csv("C://Users//张铭韬//Desktop//学业//港科大//MSDM5054机器学习//作业//hw3//Hitters_test.csv")

train = train[["Years", "Hits", "RBI","Walks", "PutOuts", "Runs","Salary"]]
test = test[["Years", "Hits", "RBI","Walks", "PutOuts", "Runs","Salary"]]

train.dropna(inplace=True)
test.dropna(inplace=True)

import random
import collections
from sklearn.preprocessing import PolynomialFeatures
from sklearn.model_selection import train_test_split
from collections import deque

class TreeNode:
    def __init__(self,labels_idx=None,left=None,right=None,split_idx=None,is_discrete=None,split_value=None,father=None) -> None:

        self.labels_idx = labels_idx     # 训练集的label对应的下标
        self.left = left                 # 左子树
        self.right = right               # 右子树
        self.split_idx = split_idx       # 划分特征对应的下标
        self.is_discrete = is_discrete   # 是否离散
        self.split_value = split_value   # 划分点
        self.father = father             # 父节点

class RegressionTree:

    def __init__(self,data,labels,is_discrete,validate_ratio=0.1):

        self.data = np.array(data)
        self.labels=np.array(labels)
        self.feature_num = self.data.shape[1]
        self.is_discrete = is_discrete
        self.validate_ratio = validate_ratio
        self.leaves = []
        if validate_ratio>0:
            all_index = range(data.shape[0])
            self.train_idx,self.test_idx = train_test_split(all_index,test_size=validate_ratio)
            self.validate_data = self.data[self.test_idx,:]
            self.validate_label = self.labels[self.test_idx]
            self.train_data = self.data[self.train_idx,:]
            self.train_label = self.labels[self.train_idx]
```

```python
    def sum_std(self,x):

        return np.sum(np.abs(x-np.mean(x)))/len(x)

    def choose_feature(self,x,left_labels):

        std_list = []
        split_value_list = []
        for i in range(x.shape[1]):
            final_split_value,final_sum_std=self.calc_std(x[:,i],self.is_discrete[i],left_labels)

            std_list.append(final_sum_std)
            split_value_list.append(final_split_value)
        idx = np.argmin(std_list)
        return idx,split_value_list[idx]

    def calc_std(self,feature,is_discrete,labels):

        final_sum_std = float("inf")
        final_split_value = 0
        idx = range(len(feature))
        feature_with_idx = np.c_[idx,feature]
        labels = np.array(labels)
        if is_discrete:
            values = list(set(feature))
            idx_dict = {v:[] for v in values}
            for i,fea in feature_with_idx:
                idx_dict[fea].append(i)
            for v in values:
                anti_idx = [i for i in idx if i not in idx_dict[v]]
                left = labels[idx_dict[v]]
                right = labels[anti_idx]
                if left.shape[0]==0 or right.shape[0] == 0:
                    continue
                sum_std = self.sum_std(left)+self.sum_std(right)
                if sum_std<final_sum_std:
                    final_sum_std = sum_std
                    final_split_value = v
        else:
            feature_with_idx = feature_with_idx[feature_with_idx[:,1].argsort()]
            feature = feature_with_idx[:,1]
            idx = feature_with_idx[:,0]
            for i in range(len(feature)-1):
                if feature[i]==feature[i+1]:
                    continue
                split_value = (feature[i]+feature[i+1])/2
                idx_left = idx[:i+1]
                idx_right = idx[i+1:]
                sum_std = self.sum_std(labels[idx_left.astype('int64')])+self.sum_std(labels[idx_right.astype('int64')])
                if sum_std<final_sum_std:
                    final_sum_std = sum_std
                    final_split_value = split_value

        return final_split_value,final_sum_std
```

```
    def generate_tree(self,idxs,min_ratio):

        root = TreeNode(labels_idx=idxs)

        if len(idxs)/self.data.shape[0]<=min_ratio:
            return root

        idx,split_value = self.choose_feature(self.data[idxs,:],self.labels[idxs])
        root.split_value = split_value
        root.split_idx = idx
        left_idxs = []
        right_idxs = []

        if self.is_discrete[idx]:
            for i in idxs:
                if self.data[i,idx] != split_value:
                    right_idxs.append(i)
                else:
                    left_idxs.append(i)

        else:
            for i in idxs:
                if self.data[i,idx] <= split_value:
                    right_idxs.append(i)
                else:
                    left_idxs.append(i)

        left_idxs = np.array(left_idxs)
        right_idxs = np.array(right_idxs)
        root.left = self.generate_tree(left_idxs,min_ratio)

        if root.left:
            root.left.father = root

        root.right = self.generate_tree(right_idxs,min_ratio)

        if root.right:
            root.right.father = root

        return root

    def train(self,max_depth = 0,min_ratio=0.05):

        if self.validate_ratio>0:
            idx = self.train_idx
        else:
            idx = range(len(self.labels))

        self.tree = self.generate_tree(idx,min_ratio)

        # 后剪枝
        if self.validate_ratio>0:
            self.find_leaves(self.tree)
            nodes = deque(self.leaves)
            while len(nodes)>0:
                n=len(nodes)
```

```
            for _ in range(n):
                node = nodes.popleft()
                if not node.father:
                    nodes = []
                    break
                valid_pred = self.predict(self.validate_data)
                mse_before = self.get_mse(valid_pred,self.validate_label)
                backup_left = node.father.left
                backup_right= node.father.right
                node.father.left = None
                node.father.right = None
                valid_pred = self.predict(self.validate_data)
                mse_after = self.get_mse(valid_pred,self.validate_label)
                if mse_after>mse_before:
                    node.father.left = node.father.left
                    node.father.right = node.father.right
                else:
                    nodes.append(node.father)
    # 树深
    if max_depth>0:
        nodes = deque([self.tree])
        d=1
        while len(nodes)>0 and d<max_depth:
            n = len(nodes)
            for _ in range(n):
                node = nodes.popleft()
                if node.left:
                    nodes.append(node.left)
                if node.right:
                    nodes.append(node.right)
            d += 1
        if len(nodes)>0:
            for node in nodes:
                node.left=None
                node.right=None

def find_leaves(self,node):

    if not node.left and not node.right:
        self.leaves.append(node)
        return None
    else:
        if node.left:
            self.find_leaves(node.left)
        if node.right:
            self.find_leaves(node.right)

def predict_one(self,x,node=None):

    if node == None:
        node = self.tree
    while node.left and node.right:
        idx = node.split_idx
        if self.is_discrete[idx]:
            if x[idx]==node.split_value:
                node = node.left
```

```
                else:
                    node = node.right
            else:
                if x[idx]>node.split_value:
                    node = node.right
                else:
                    node = node.left

        res_idx = node.labels_idx
        return np.mean(self.labels[res_idx])

    def predict(self,x,node=None):

        x = np.array(x)
        predicts = []
        for i in range(x.shape[0]):
            res = self.predict_one(x[i,:],node)
            predicts.append(res)
        return predicts

    def get_mse(self,y_pred,y_true):

        y_pred = np.array(y_pred)
        y_true = np.array(y_true)

        return np.mean(np.square(y_pred-y_true))

x_train = train.iloc[:,0:6].values
x_test = test.iloc[:,0:6].values
y_train = train.iloc[:,6].values
y_test = test.iloc[:,6].values

rt = RegressionTree(x_train,y_train,is_discrete=[False,False,False,False,False,False],valid
ate_ratio=0.1)

rt.train(max_depth=15,min_ratio=0.05) # 限制深度和最小v_gain（实际函数已经进行剪枝）
res = rt.predict(x_test)
rt.get_mse(res,y_test)

# ############ 对比库 #############
# from sklearn.tree import DecisionTreeRegressor
#
# tes = DecisionTreeRegressor()
# tes.fit(x_train,y_train)
# res2 = tes.predict(x_test)
# rt.get_mse(res2,y_test)
```

```
## 0.8655267163965358
```