Data 3 HW 5

Sean Duan

10/8/2020

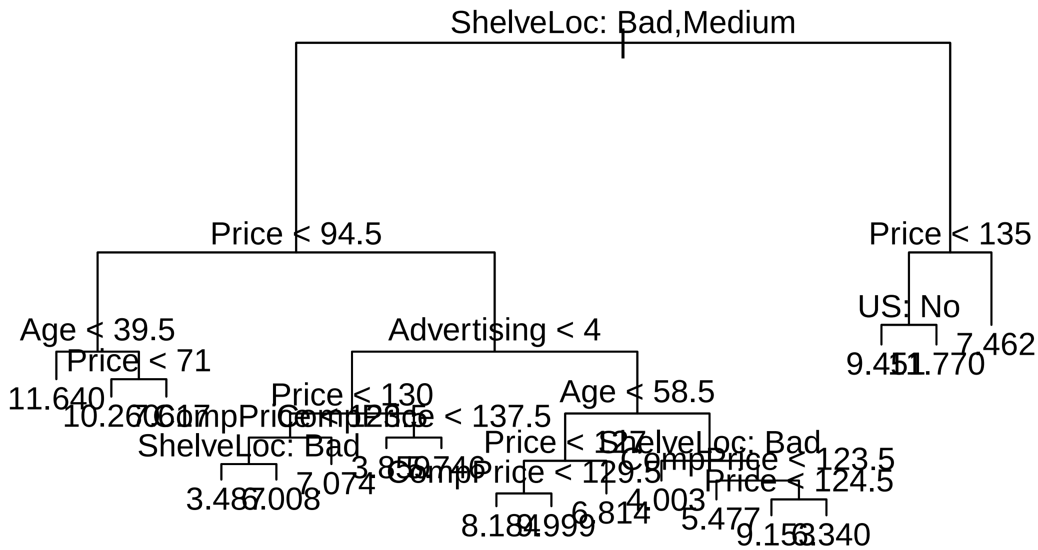
1.

A

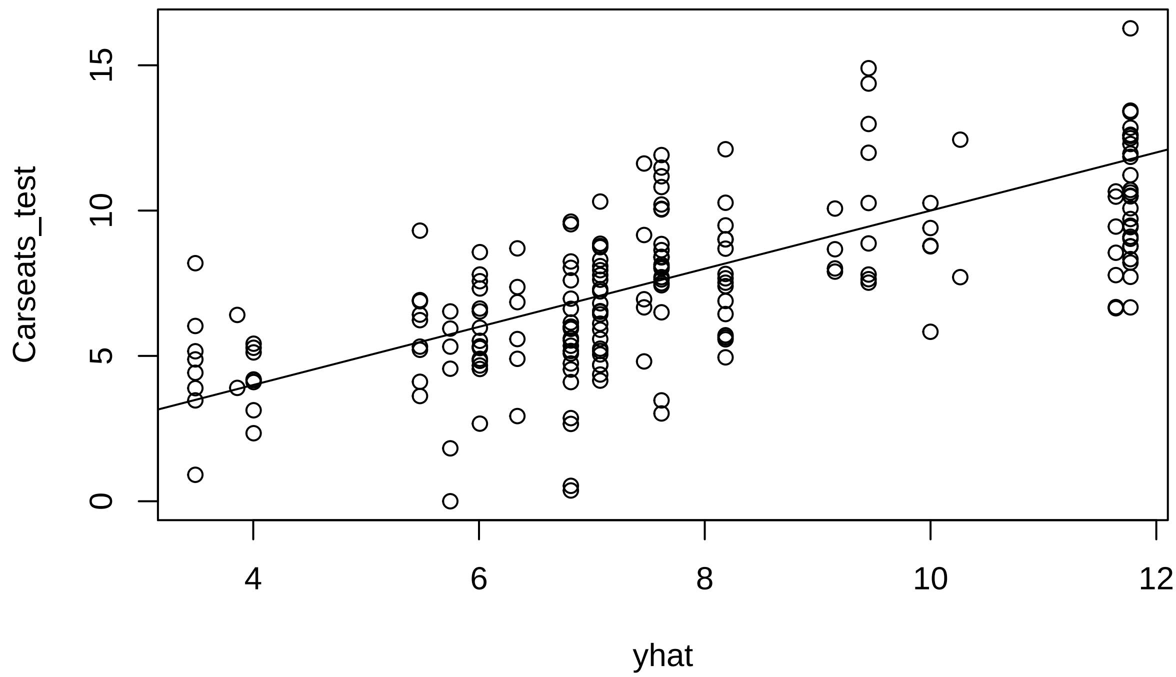
```r
train = sample (1:nrow(Carseats), nrow(Carseats)/2)
attach(Carseats)
data(Carseats)
#fitting a regression tree
carseat_t1=tree(Sales~.,Carseats , subset=train)
summary(carseat_t1)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats, subset = train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "Advertising" "CompPrice"
## [6] "US"
## Number of terminal nodes:  18
## Residual mean deviance:  2.167 = 394.3 / 182
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -3.88200 -0.88200 -0.08712  0.00000  0.89590  4.09900
```

```r
#only 6 vars are used, shelveloc,price,age,advertising, comprice,and us
#SSE is 2.16
plot(carseat_t1)
text(carseat_t1, pretty=0)
```




```
#get test/train mse
Carseats_test=Carseats [-train ,"Sales"]
yhat=predict(carseat_t1,newdata=Carseats[-train,])
plot(yhat ,Carseats_test)
abline (0,1)
```




```r
mean((yhat -Carseats_test)^2)
```

```
## [1] 4.922039
```

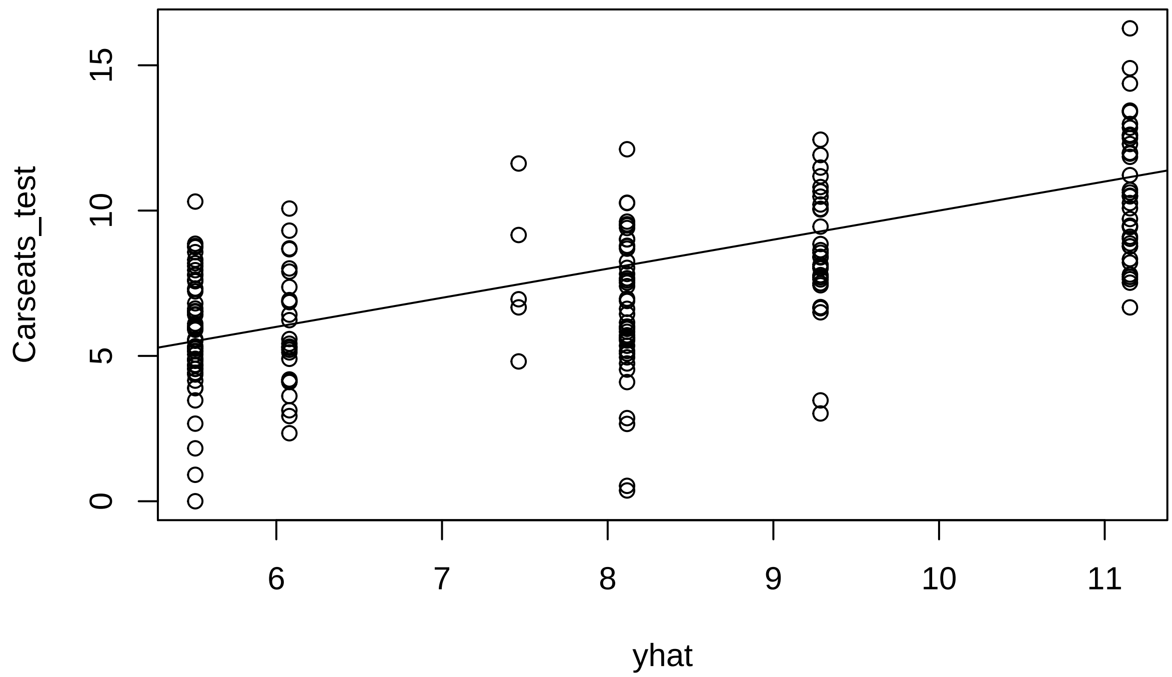
```r
#MSE is 4.922
```

Looking at our tree, it seems that the most important variables are shelf location and price, as evinced by which elements are the variables used as split criteria in our tree (Shelveloc, price, age, advertising, compprice, US). The test MSE we obtain is 4.922.

B

```r
#pruning
cv_carseat_t1=cv.tree(carseat_t1)
plot(cv_carseat_t1$size ,cv_carseat_t1$dev ,type="b")
```


```
#no point in pruning our tree, although getting it to 5 wouldn't be bad if we wanted to simplify it
#lets try w/ prune @ 6
carseat_prune<-prune.tree(carseat_t1, best =6)

yhat=predict(carseat_prune,newdata=Carseats[-train,])
plot(yhat ,Carseats_test)
abline (0,1)
```




```r
mean((yhat -Carseats_test)^2)
```

```
## [1] 5.318073
```

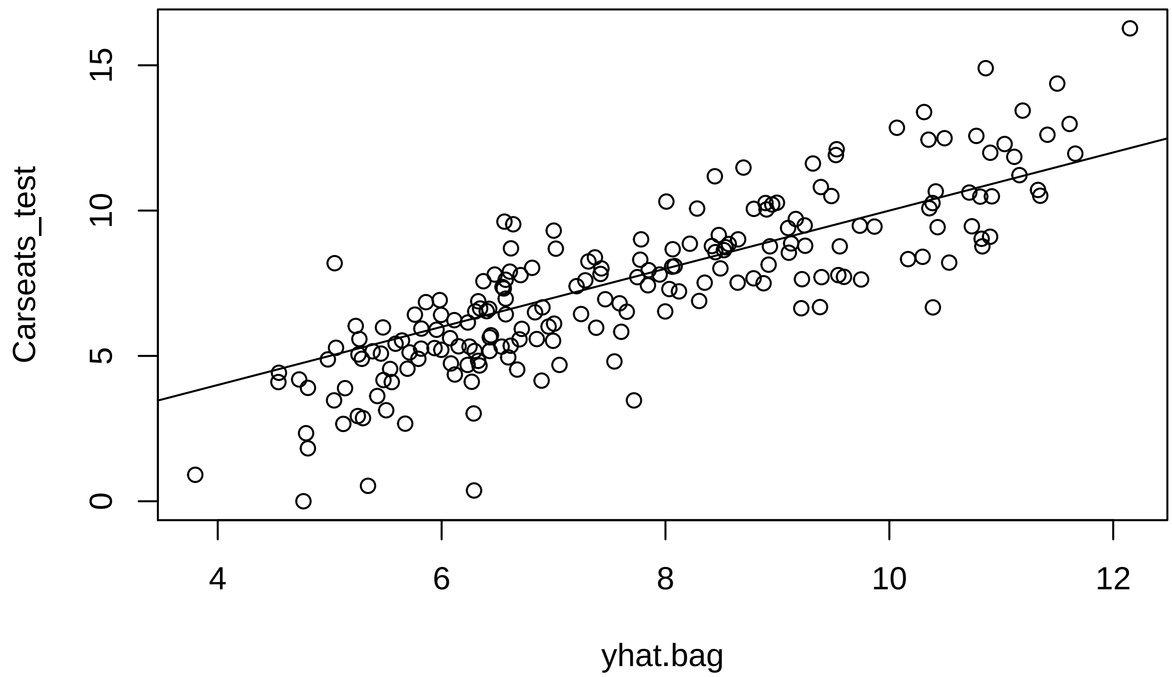
```r
#worse MSE 5.318
```

The optimal level of tree complexity is the full size tree with 18 terminal nodes. Pruning the tree does not improve test MSE, as evinced when we pruned the tree to a 6 node version. Our test MSE there was 5.318, which was noticably worse. ## C

```r
#lets do bagging!
carseat_bag=randomForest(Sales~.,data=Carseats , subset=train ,mtry=10,importance =TRUE)
carseat_bag
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = Carseats, mtry = 10,      importance = TRUE, subset = trai
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 2.931324
##                     % Var explained: 62.72
```

```r
#checking test MSE
yhat.bag = predict (carseat_bag , newdata=Carseats[-train ,])
plot(yhat.bag , Carseats_test)
abline (0,1)
```



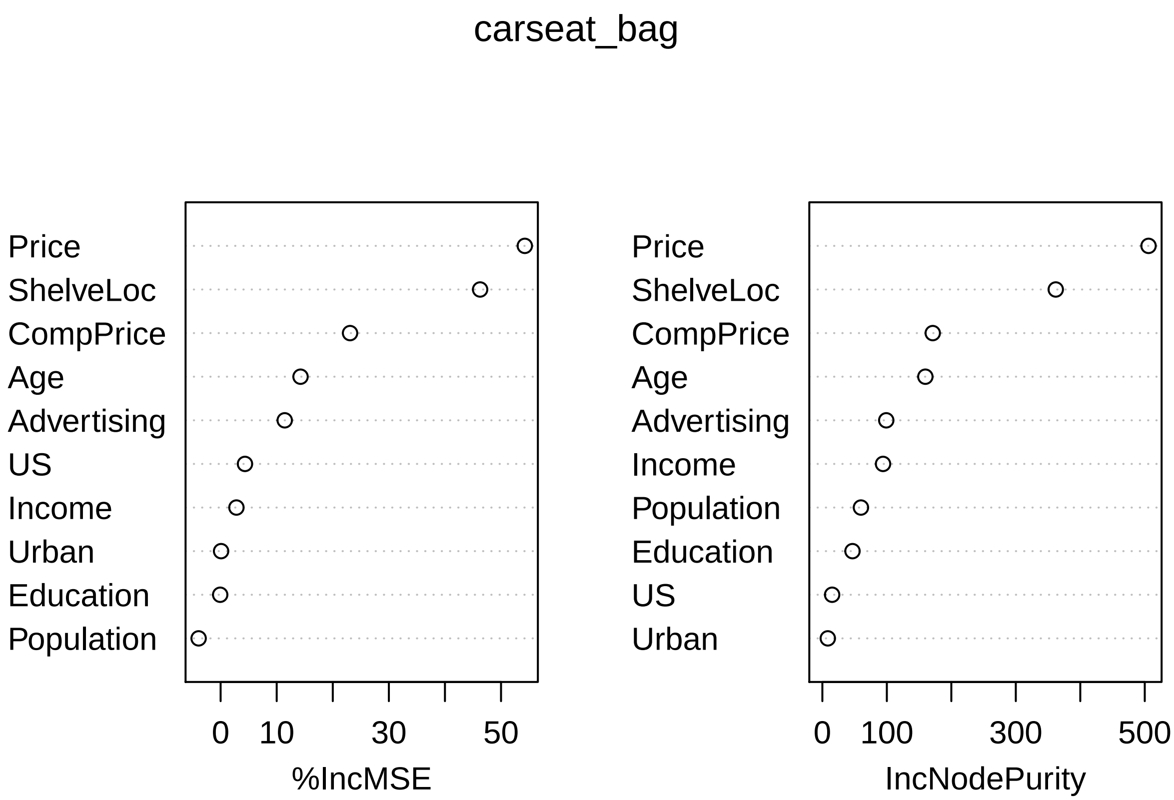

```r
mean((yhat.bag -Carseats_test)^2)
```

```
## [1] 2.657296
```

```r
#test MSE is 2.622! even bestter
#we can check importance of each variable by using the importance() fxn
importance(carseat_bag)
```

```
##                 %IncMSE IncNodePurity
## CompPrice   23.07909904    171.185734
## Income       2.82081527     94.079825
## Advertising 11.43295625     99.098941
## Population  -3.92119532     59.818905
## Price       54.24314632    505.887016
## ShelveLoc   46.26912996    361.962753
## Age         14.24992212    159.740422
## Education   -0.07662320     46.738585
## Urban        0.08530119      8.453749
## US           4.34349223     15.157608
```

```r
#we can plot these using this code
varImpPlot(carseat_bag)
```




```
#price and shelveloc most important, comp price not bad either
```

The test MSE we obtain with bagging is 2.622, which is the best we have had thus far! The most important variables seem to be price, shelf location, and comp price.

D

```
##D
#boosting!
set.seed(1)
carseat_boost=gbm(Sales~.,data=Carseats[train ,], distribution="gaussian",n.trees=5000, interaction.dep
#choose gaussian b/c i want sq err loss, not anything else
yhat.boost=predict (carseat_boost ,newdata =Carseats[-train ,],n.trees=5000)
mean((yhat.boost - Carseats_test)^2)
```

```
## [1] 1.806206
```

```
#2.44 mse
#code for our own k-fold cv
car_train<-Carseats[train,]
K=5
folds = sample(1:K,nrow(car_train),replace=T)
modelfits<-list(NA)
errorlist<-list(NA)
bestmodel<-list(NA)
moderror<-list(NA)
yhat.boost<-list(NA)
```

```
carseat_boost<-list(NA)
testn<-seq(from=100, to=10000, length.out = 10)
test_mse<-list(NA)

for(k in 1:K){
  CV.train = car_train[folds != k,]
  CV.test = car_train[folds == k,]
  CV.ts_y = CV.test$Sales
  for (i in 1:10){
    carseat_boost[[i]]=gbm(Sales~.,data=CV.train, distribution="gaussian",n.trees=testn[[i]], interacti
    yhat.boost[[i]]=predict(carseat_boost[[i]] ,newdata =CV.test,n.trees=testn[[i]])
    test_mse[[i]]<-mean((yhat.boost[[i]] - CV.ts_y)^2)
  }
  moderror[[k]]<-test_mse[[which.min(test_mse)]]
  bestmodel[[k]]<-which.min(test_mse)}
moderror
```

```
## [[1]]
## [1] 1.711042
##
## [[2]]
## [1] 1.807045
##
## [[3]]
## [1] 2.18769
##
## [[4]]
## [1] 2.307298
##
## [[5]]
## [1] 1.499894
```

```
bestmodel
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 2
##
## [[4]]
## [1] 2
##
## [[5]]
## [1] 2
```

```
#ntrees here is 1200

car_train<-Carseats[train,]
K=5
folds = sample(1:K,nrow(car_train),replace=T)
modelfits<-list(NA)
```

```
errorlist<-list(NA)
bestmodel<-list(NA)
moderror<-list(NA)
yhat.boost<-list(NA)
carseat_boost<-list(NA)
testn<-seq(from=100, to=10000, length.out = 10)
test_mse<-list(NA)

for(k in 1:K){
  CV.train = car_train[folds != k,]
  CV.test = car_train[folds == k,]
  CV.ts_y = CV.test$Sales
  for (i in 1:10){
    carseat_boost[[i]]=gbm(Sales~.,data=CV.train, distribution="gaussian",n.trees=testn[[i]], interactio
    yhat.boost[[i]]=predict(carseat_boost[[i]] ,newdata =CV.test,n.trees=testn[[i]])
    test_mse[[i]]<-mean((yhat.boost[[i]] - CV.ts_y)^2)
  }
  moderror[[k]]<-test_mse[[which.min(test_mse)]]
  bestmodel[[k]]<-which.min(test_mse)}

#ntrees here is 1200

carseat_boost=gbm(Sales~.,data=Carseats[train,], distribution="gaussian",n.trees=1200, interaction.dept
yhat.boost=predict(carseat_boost,newdata =Carseats[-train,],n.trees=1200)
mean((yhat.boost[[i]] - CV.ts_y)^2)
```

```
## [1] 6.53197
```

```
#looking at the code, it seems like the best ntrees is 1200, which is guarding against overfitting w/ l
#tried to fit w/ a stump first, but could not get better results w/ a larger interaction depth.
```

The distribution we used was Gaussian, because we were looking at what we believed to be normally distributed continous data. We chose the values for n-tree and interaction depth by comparing several 5-fold cross validated models on our training data set. We then used our cross validated values and predicted test MSE using our hold-out data. Our test MSE was 9.2653. Ntrees was set to 1200, and we used a 'stump' for our interaction depth (depth = 1).

E

```
#Random Forest

#we start w/ a loop to find the best m value!

K=5
folds = sample(1:K,nrow(car_train),replace=T)
modelfits<-list(NA)
errorlist<-list(NA)
bestmodel<-list(NA)
moderror<-list(NA)
mvec<-seq(from=1, to=10)
yhat.bag<-list(NA)
carseat_rf<-list(NA)
testn<-seq(from=100, to=10000, length.out = 10)
test_mse<-list(NA)
```

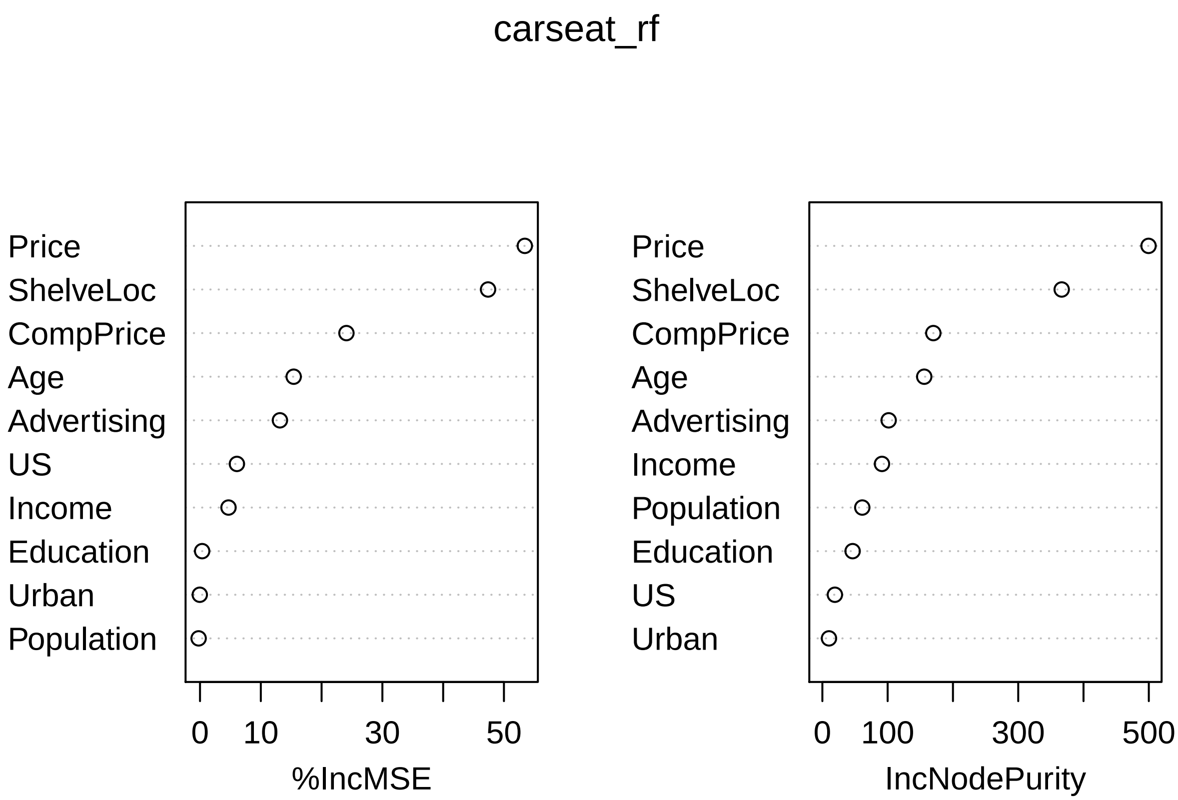
```r
for(k in 1:K){
  CV.train = car_train[folds != k,]
  CV.test = car_train[folds == k,]
  CV.ts_y = CV.test$Sales
  for(i in 1:10){
    carseat_rf[[i]]=randomForest(Sales~.,data=CV.train,mtry=mvec[i],importance =TRUE)
    yhat.bag[[i]]<-predict(carseat_rf[[i]] , newdata=CV.test)
    test_mse[[i]]<-mean((yhat.bag[[i]] -CV.ts_y)^2)
    }
  moderror[[k]]<-test_mse[[which.min(test_mse)]]
  bestmodel[[k]]<-which.min(test_mse)}

#setting m to 8 gets us the lowest cv MSE
carseat_rf=randomForest(Sales~.,data=Carseats[train,],mtry=8,importance =TRUE)
yhat.bag<-predict(carseat_rf , newdata=Carseats[-train,])
test_mse<-mean((yhat.bag -Carseats$Sales[-train])^2)
test_mse
```

```
## [1] 2.603308
```

```r
importance(carseat_rf)
```

```
##                 %IncMSE IncNodePurity
## CompPrice   24.08540996     169.98617
## Income       4.68460540      91.31148
## Advertising 13.14470071     101.55552
## Population  -0.23122125      61.13341
## Price       53.43480252     499.61689
## ShelveLoc   47.38654599     366.57422
## Age         15.40979563     156.01850
## Education    0.34958024      46.33423
## Urban       -0.04167871      10.13373
## US           6.06723168      19.19490
```

```r
varImpPlot(carseat_rf)
```



Test MSE was 2.6003. The variables that were the most important seemed to be price, shelvelocation, and compprice.We were able to obtain our lowest error through 5-fold cross validation when m was set to 8 on our training data. Other values of m were inferior to 8 with regards to obtaining the lowest error.

F

```r
#mars!~
#used tutorial from
#http://uc-r.github.io/mars
carseat_mars<-earth(Sales~.,data=Carseats, subset = train)
summary(carseat_mars)
```

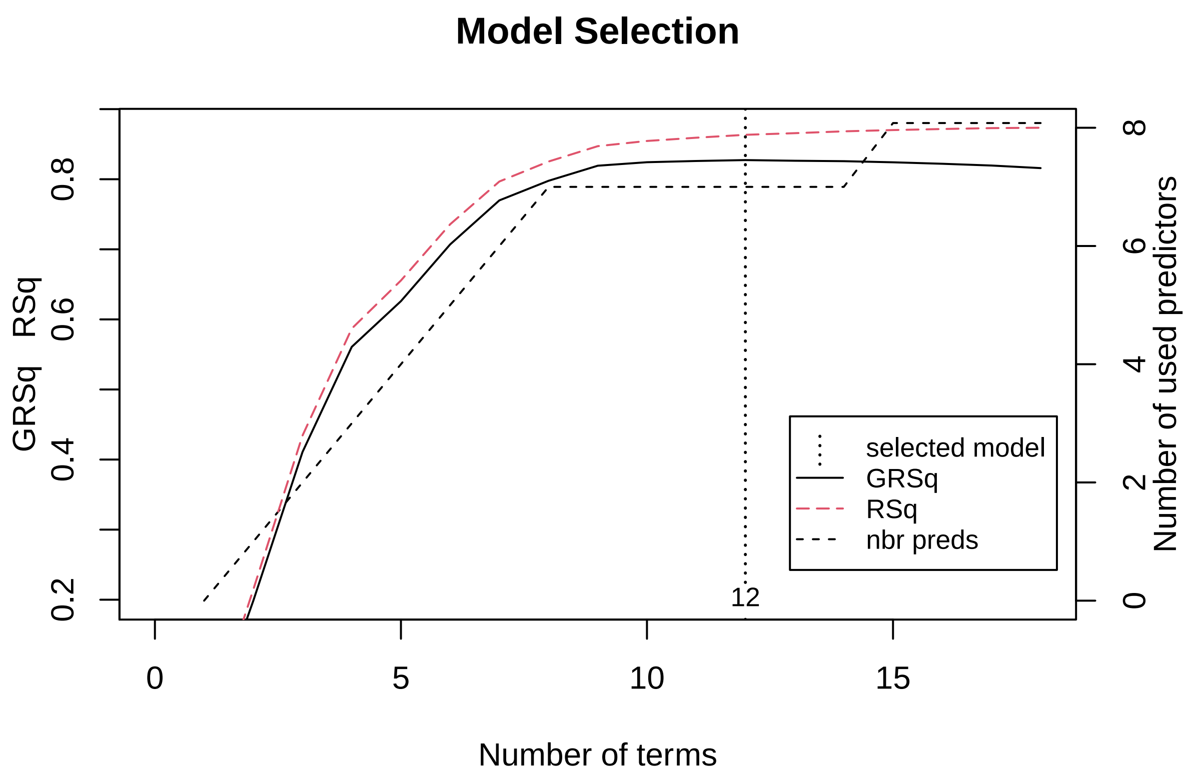
```
## Call: earth(formula=Sales~., data=Carseats, subset=train)
##
##                     coefficients
## (Intercept)           12.7567709
## ShelveLocGood          4.8181598
## ShelveLocMedium        2.1665778
## h(CompPrice-147)       0.0857445
## h(149-CompPrice)      -0.0922506
## h(109-Income)         -0.0173033
## h(10-Advertising)     -0.1722032
## h(Price-74)           -0.0886712
## h(97-Price)            0.0509660
## h(55-Age)              0.0479284
## h(Age-55)             -0.0990830
## h(Age-66)              0.1210698
```

```
##
## Selected 12 of 18 terms, and 7 of 11 predictors
## Termination condition: Reached nk 23
## Importance: ShelveLocGood, Price, CompPrice, Age, ShelveLocMedium, ...
## Number of terms at each degree of interaction: 1 11 (additive model)
## GCV 1.371816    RSS 214.8882    GRSq 0.827285    RSq 0.8633624
```

```r
print(carseat_mars)
```

```
## Selected 12 of 18 terms, and 7 of 11 predictors
## Termination condition: Reached nk 23
## Importance: ShelveLocGood, Price, CompPrice, Age, ShelveLocMedium, ...
## Number of terms at each degree of interaction: 1 11 (additive model)
## GCV 1.371816    RSS 214.8882    GRSq 0.827285    RSq 0.8633624
```

```r
#model selection code for mars
plot(carseat_mars, which=1)
```




```r
#mars models scale invariant

#test mse code
yhat=predict(carseat_mars,newdata=Carseats[-train,])
mean((yhat -Carseats_test)^2)
```

```
## [1] 1.231644
```

```r
#1.17
```

I first followed a tutorial "http://uc-r.github.io/mars" as material on MARS was not covered in class or our text books. The R package I used was earth. I created our simple MARS model by using the earth function, prediction sales from all our variables, on our training data. Then I used that model to predict our outcomes from our testing set, and calculated the MSE. Test MSE was 1.17, the lowest out of all of our methods thus far.

2

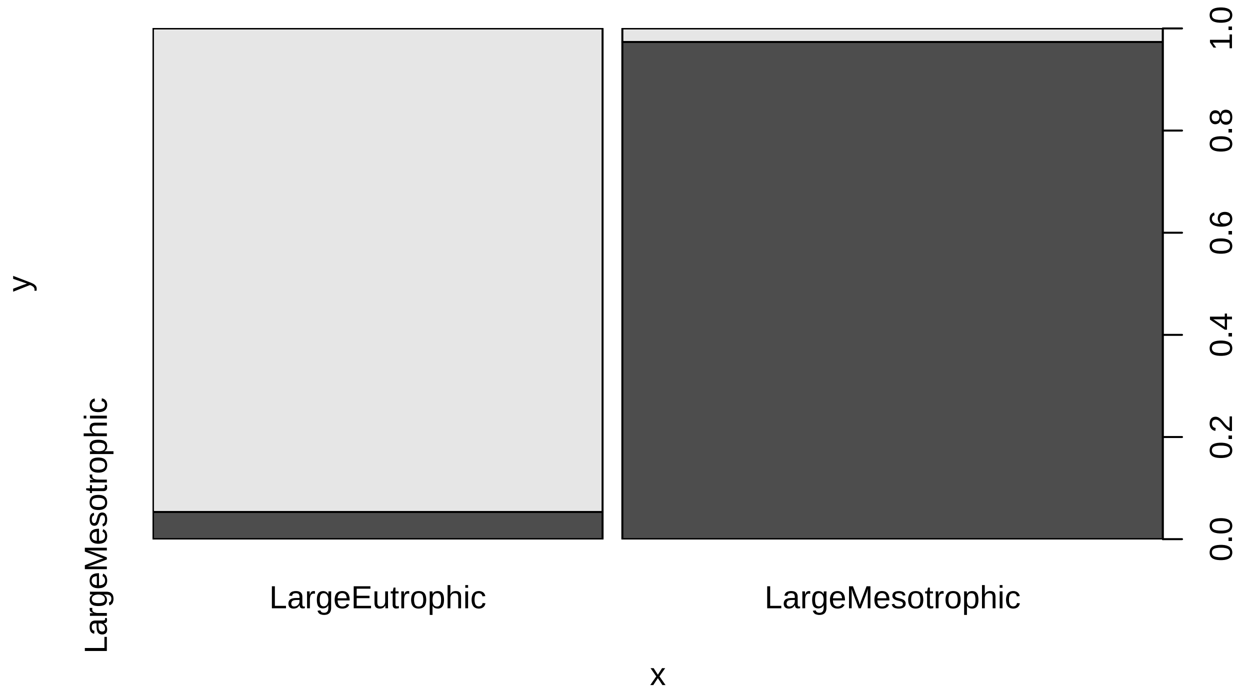
```
load("fish_data3.RData")
fish<-fish_data3
fish$LSH7class = droplevels(fish$LSH7class)

#split into test/training
set.seed(1)
training.set=sample(1:nrow(fish),400)
fish.test=fish[-training.set,]
LSH7class.test=fish.test[,12]
#goal - lowest classif error (use pred to find that out)
```

CART bagging

```
#find best CART bagging
fish_bag=randomForest(LSH7class~.,data=fish , subset=training.set ,mtry=11,importance =TRUE)
fish_bag
```

```
##
## Call:
##  randomForest(formula = LSH7class ~ ., data = fish, mtry = 11,      importance = TRUE, subset = trai
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 11
##
##         OOB estimate of  error rate: 4.75%
## Confusion matrix:
##                   LargeEutrophic LargeMesotrophic class.error
## LargeEutrophic               169               11  0.06111111
## LargeMesotrophic               8              212  0.03636364
```

```
#error rate 4.75
#checking test MSE
yhat.bag = predict (fish_bag , newdata=fish[-training.set ,])
plot(yhat.bag , LSH7class.test)
```

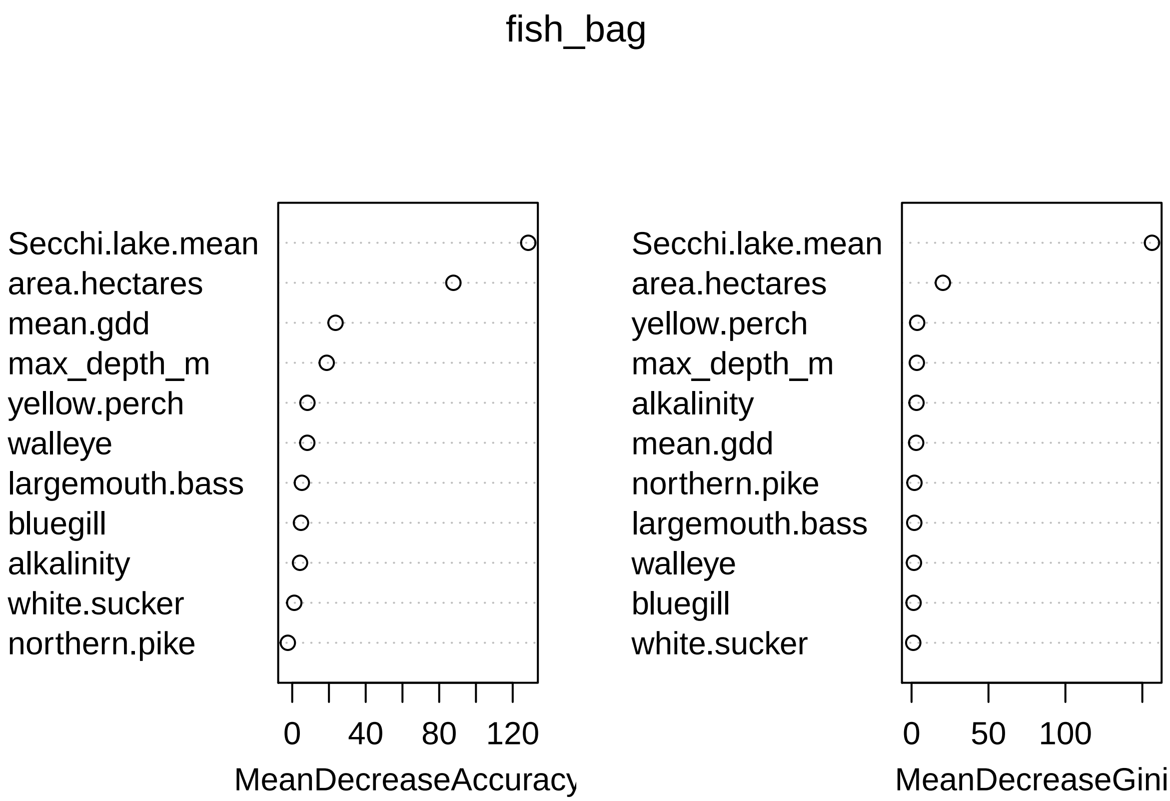



```r
#importance of variables
importance(fish_bag)
```

```
##                   LargeEutrophic LargeMesotrophic MeanDecreaseAccuracy
## bluegill             -0.82912162        5.2508410             4.772639
## northern.pike        -2.85699646       -0.6092413            -2.388441
## walleye               1.28136287        8.3437755             8.199436
## white.sucker         -0.35879852        1.8131908             1.109723
## yellow.perch          4.20492971        7.0812266             8.280914
## largemouth.bass      -0.01245867        5.9502533             5.259010
## area.hectares        23.34802436       88.9895215            87.733261
## max_depth_m           8.87375762       17.1324764            18.770539
## Secchi.lake.mean     65.08231908      120.0963122           128.466043
## mean.gdd             -0.10515857       23.2923160            23.572841
## alkalinity           -0.57471066        6.8080975             4.247411
##                   MeanDecreaseGini
## bluegill                  1.319191
## northern.pike             1.862354
## walleye                   1.553284
## white.sucker              1.109469
## yellow.perch              3.643456
## largemouth.bass           1.755862
## area.hectares            20.359902
## max_depth_m               3.420307
## Secchi.lake.mean        156.280401
## mean.gdd                  2.999120
```

```
## alkalinity                  3.210195
```

```r
varImpPlot(fish_bag)
```




```r
#secchi lake mean, area hectacres most important, hectacres are good for accuracy, not for node purity!

table(yhat.bag,LSH7class.test)
```

```
##                     LSH7class.test
## yhat.bag             LargeEutrophic LargeMesotrophic
##   LargeEutrophic                 89                5
##   LargeMesotrophic                3              110
```

```r
1-mean(yhat.bag==LSH7class.test)
```

```
## [1] 0.03864734
```

```r
#3.86% class error!
```

Using bagging, we were able to get a 3.86% classification error.

Random Forest

```r
#Random Forest
#we start w/ a loop to find the best m value!
fish_train<-fish[training.set,]
K=5
folds = sample(1:K,nrow(fish_train),replace=T)
bestmodel<-list(NA)
```

```
moderror<-list(NA)
test_mse<-list(NA)
mvec<-seq(from=1, to=11)
yhat.bag<-list(NA)
fish_rf<-list(NA)
class_err<-list(NA)

for(k in 1:K){
  CV.train = fish_train[folds != k,]
  CV.test = fish_train[folds == k,]
  CV.ts_y = CV.test$LSH7class
 for(i in 1:11){
  fish_rf[[i]]=randomForest(LSH7class~.,data=CV.train ,mtry=mvec[i],importance =TRUE)
  yhat.bag[[i]]<-predict(fish_rf[[i]] , newdata=CV.test)
  class_err[[i]]<-(1-mean(yhat.bag[[i]]==CV.ts_y))
  }
  moderror[[k]]<-class_err[[which.min(class_err)]]
  bestmodel[[k]]<-which.min(class_err)}
moderror
```

```
## [[1]]
## [1] 0.05555556
##
## [[2]]
## [1] 0.04494382
##
## [[3]]
## [1] 0.01369863
##
## [[4]]
## [1] 0.05882353
##
## [[5]]
## [1] 0.03703704
```

```
bestmodel
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 2
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 2
```

```
#setting m to 2 gets us the lowest class error
#lets try ntree
fish_train<-fish[training.set,]
```

```r
K=5
folds = sample(1:K,nrow(fish_train),replace=T)
bestmodel<-list(NA)
moderror<-list(NA)
test_mse<-list(NA)
yhat.bag<-list(NA)
fish_rf<-list(NA)
class_err<-list(NA)
testn<-seq(from=300, to=3000, length.out = 10)
for(k in 1:K){
  CV.train = fish_train[folds != k,]
  CV.test = fish_train[folds == k,]
  CV.ts_y = CV.test$LSH7class
 for(i in 1:10){
  fish_rf[[i]]=randomForest(LSH7class~.,data=CV.train ,mtry=2, ntree=testn[[i]], importance =TRUE)
  yhat.bag[[i]]<-predict(fish_rf[[i]] , newdata=CV.test)
  class_err[[i]]<-(1-mean(yhat.bag[[i]]==CV.ts_y))
  }
  moderror[[k]]<-class_err[[which.min(class_err)]]
  bestmodel[[k]]<-which.min(class_err)}
#best ntrees is 300 across all folds
#test on our test data
fish_rf=randomForest(LSH7class~.,data=fish[training.set,] ,mtry=2, ntree=300, importance =TRUE)
yhat.bag<-predict(fish_rf , newdata=fish[-training.set,])
1-mean(yhat.bag==LSH7class.test)
```

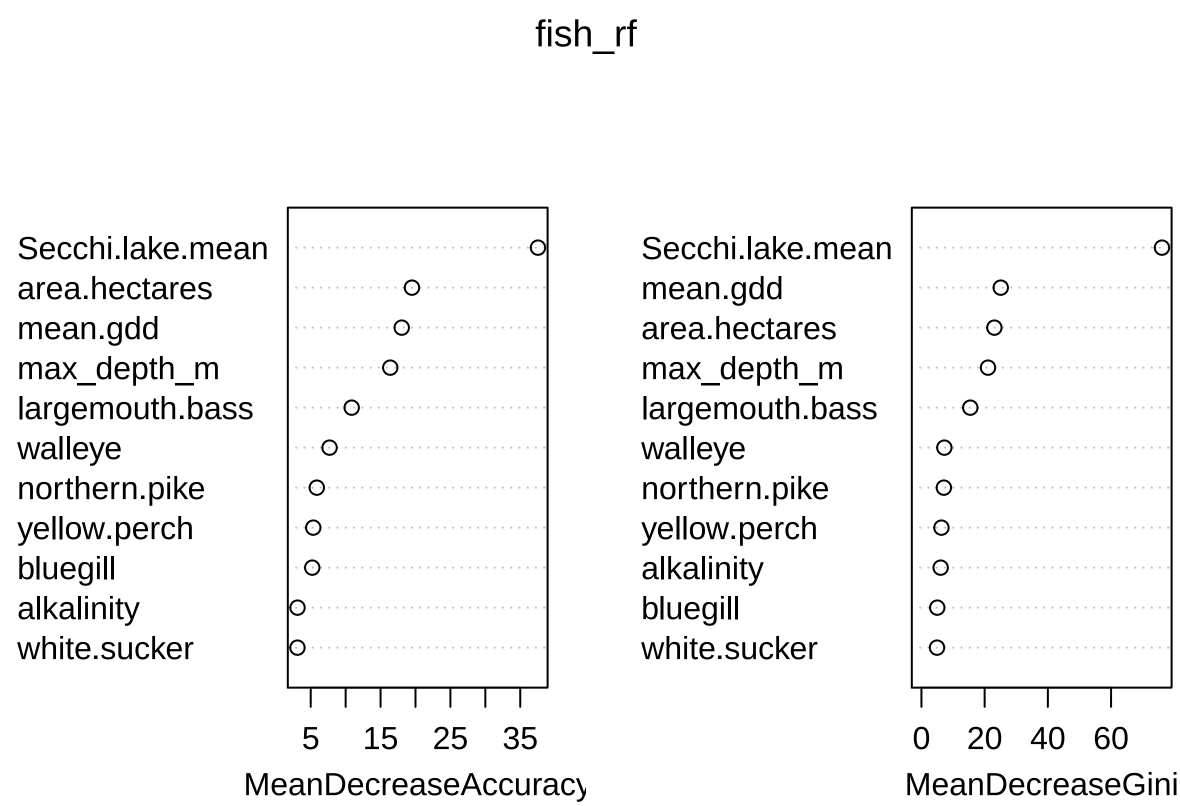
```
## [1] 0.01449275
```

```r
#2.41% class err
importance(fish_rf)
```

```
##                 LargeEutrophic LargeMesotrophic MeanDecreaseAccuracy
## bluegill             1.5818992         4.962150             5.215712
## northern.pike        3.7115781         4.551053             5.865495
## walleye              1.6754816         9.091366             7.697122
## white.sucker         0.2489845         3.776059             3.097779
## yellow.perch         4.9013950         3.072281             5.359472
## largemouth.bass      8.9144440         7.944439            10.865559
## area.hectares       14.4860868        18.555245            19.492887
## max_depth_m         10.5617017        14.124754            16.387261
## Secchi.lake.mean    29.6056319        33.621926            37.539994
## mean.gdd             9.4441186        16.326457            18.030367
## alkalinity           1.7088996         2.626790             3.107658
##                 MeanDecreaseGini
## bluegill                4.999438
## northern.pike           7.094356
## walleye                 7.243697
## white.sucker            4.928832
## yellow.perch            6.323404
## largemouth.bass        15.453453
## area.hectares          23.075739
## max_depth_m            21.061704
## Secchi.lake.mean       76.112530
## mean.gdd               25.090622
```

```
## alkalinity                  6.081357
```

```r
varImpPlot(fish_rf)
```




```r
#secchi lake mean, area hectacres, mean gdd, max depth most important
```

MARS

```r
#mars
fish_mars<-earth(LSH7class~.,data=fish, subset = training.set)
summary(fish_mars)
```

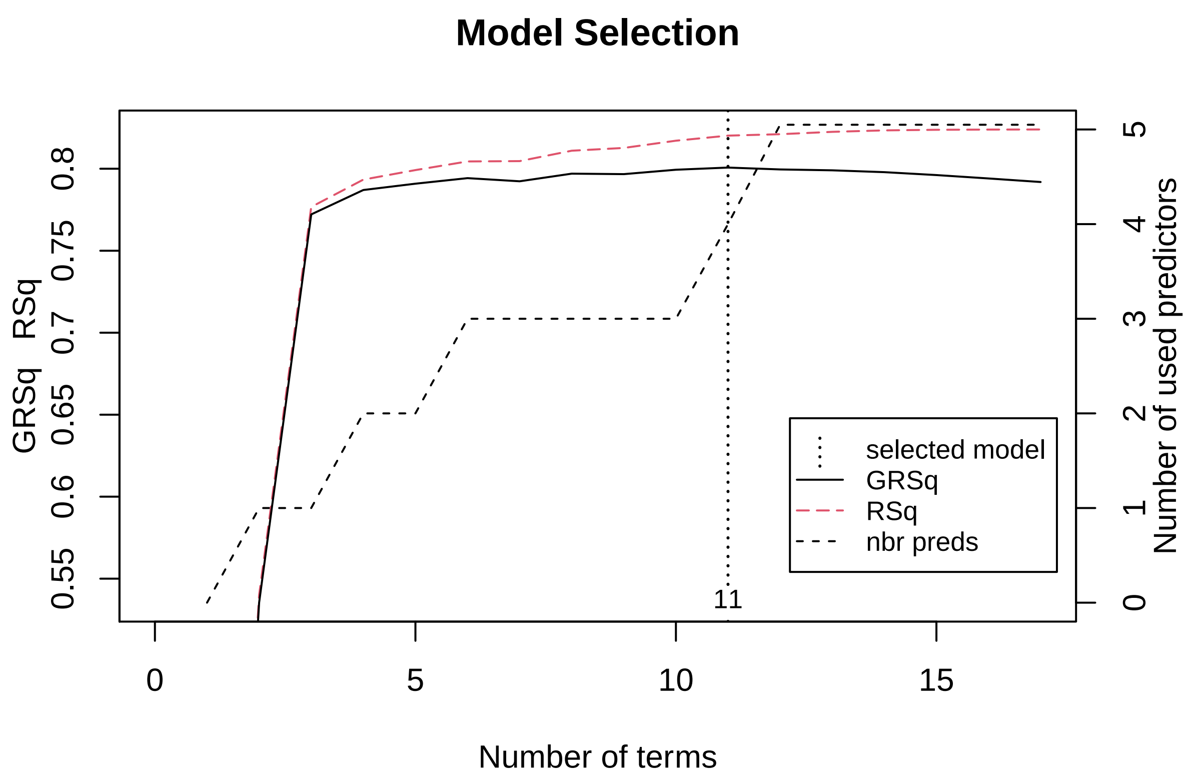
```
## Call: earth(formula=LSH7class~., data=fish, subset=training.set)
##
##                              coefficients
## (Intercept)                     0.4139332
## h(area.hectares-157.423)       -0.0030485
## h(241.071-area.hectares)       -0.0029211
## h(area.hectares-241.071)        0.0030446
## h(9.144-max_depth_m)           -0.0212701
## h(Secchi.lake.mean-1.51375)    -1.6394428
## h(Secchi.lake.mean-1.64448)     3.3602929
## h(Secchi.lake.mean-2.25591)    -4.4693784
## h(Secchi.lake.mean-2.3375)      3.0774933
## h(Secchi.lake.mean-2.98909)    -0.3186691
## h(23.35-alkalinity)            -0.0156677
##
```

```
## Selected 11 of 17 terms, and 4 of 11 predictors
## Termination condition: Reached nk 23
## Importance: Secchi.lake.mean, area.hectares, max_depth_m, alkalinity, ...
## Number of terms at each degree of interaction: 1 10 (additive model)
## GCV 0.04957475    RSS 17.80242    GRSq 0.8006983    RSq 0.8201776
```

```
print(fish_mars)
```

```
## Selected 11 of 17 terms, and 4 of 11 predictors
## Termination condition: Reached nk 23
## Importance: Secchi.lake.mean, area.hectares, max_depth_m, alkalinity, ...
## Number of terms at each degree of interaction: 1 10 (additive model)
## GCV 0.04957475    RSS 17.80242    GRSq 0.8006983    RSq 0.8201776
```

```
#model selection code for mars
plot(fish_mars, which=1)
```




```
#mars models scale invariant

#test err code
yhat=predict(fish_mars,newdata=fish[-training.set,])
mars.pred=rep("LargeEutrophic" ,207)
mars.pred[yhat >.5]="LargeMesotrophic"
table(mars.pred, LSH7class.test)
```

```
##                   LSH7class.test
## mars.pred           LargeEutrophic LargeMesotrophic
##   LargeEutrophic                85                1
```

```
##   LargeMesotrophic              7              114
```

```
1-mean(mars.pred==LSH7class.test)
```

```
## [1] 0.03864734
```

```
#3.86% error, not better than others
#var importance code
evimp(fish_mars)
```

```
##                  nsubsets   gcv    rss
## Secchi.lake.mean       10 100.0  100.0
## area.hectares           8  18.9   23.0
## max_depth_m             6  11.1   16.0
## alkalinity              1   4.0    6.1
```

CART Boosting

```
#best CART boosting

#5fold cv for boosting
fish.B=fish
fish.B[,12]=rep(0,nrow(fish))
fish.B[which(fish$LSH7class=="LargeEutrophic"),12]=1
fish_train<-fish.B[training.set,]
K=5
folds = sample(1:K,nrow(fish_train),replace=T)
bestmodel<-list(NA)
moderror<-list(NA)
test_mse<-list(NA)
yhat.boost<-list(NA)
fish_boost<-list(NA)
class_err<-list(NA)
testn<-seq(from=300, to=3000, length.out = 10)

for(k in 1:K){
  CV.train = fish_train[folds != k,]
  CV.test = fish_train[folds == k,]
  CV.ts_y = CV.test$LSH7class
for (i in 1:10){
  fish_boost[[i]]=gbm(LSH7class~.,data=CV.train, distribution="bernoulli",n.trees=testn[[i]], interacti
  yhat.boost[[i]]=round(predict(fish_boost[[i]] ,newdata =CV.test,n.trees=testn[[i]], type = "response"
  class_err[[i]]<-(1-mean(yhat.boost[[i]]==CV.ts_y))
  }
  moderror[[k]]<-class_err[[which.min(class_err)]]
  bestmodel[[k]]<-which.min(class_err)}
moderror
```

```
## [[1]]
## [1] 0.05
##
## [[2]]
## [1] 0.08333333
##
## [[3]]
## [1] 0.05063291
```

```
##
## [[4]]
## [1] 0.03921569
##
## [[5]]
## [1] 0.02531646
```

```
bestmodel
```

```
## [[1]]
## [1] 4
##
## [[2]]
## [1] 1
##
## [[3]]
## [1] 1
##
## [[4]]
## [1] 1
##
## [[5]]
## [1] 3
```

```
#best ntrees is 300 across all folds
#test on our test data
fish_boost=gbm(LSH7class~.,data=fish.B[training.set,], distribution="bernoulli",n.trees=300, interactio
yhat.boost=round(predict(fish_boost ,newdata =fish.B[-training.set,],n.trees=300, type = "response"),0)
1-mean(yhat.boost==fish.B[-training.set,]$LSH7class)
```

```
## [1] 0.03381643
```

```
#test error was 3.38%
```

Out of all of our methods, Random Forest with m set to 2 and number of trees was set to 300. I chose these values for our parameters using 5-fold CV on our training data. Our classification error rate was 2.41% on our test data. The variables we found to be important were secchi lake mean, area hectacres, mean gdd, and max depth.