ISLR_CH8

Sky Liu

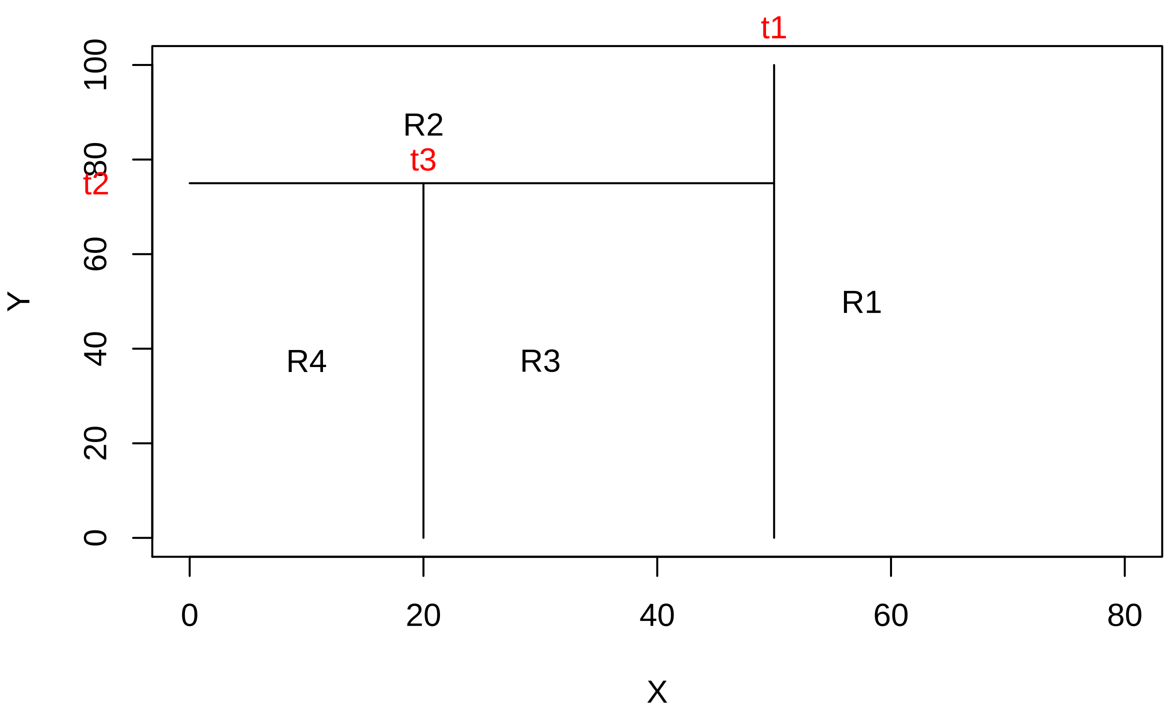
3/15/2019

8.4.1

```r
par(xpd = NA)
plot(NA, NA, type = "n", xlim = c(0, 80), ylim = c(0, 100), xlab = "X", ylab = "Y")
# t1: x = 40; (40, 0) (40, 100)
lines(x = c(50, 50), y = c(0, 100))
text(x = 50, y = 108, labels = c("t1"), col = "red")
# t2: y = 75; (0, 75) (50, 75)
lines(x = c(0, 50), y = c(75, 75))
text(x = -8, y = 75, labels = c("t2"), col = "red")

# t3: x = 20; (20,0) (20, 75)
lines(x = c(20, 20), y = c(0, 75))
text(x = 20, y = 80, labels = c("t3"), col = "red")


text(x = (40 + 75)/2, y = 50, labels = c("R1"))
text(x = 20, y = (100 + 75)/2, labels = c("R2"))
text(x = 30, y = 75/2, labels = c("R3"))
text(x = 10, y = 75/2, labels = c("R4"))
```



8.4.2

Boosting using depth-one trees is essetianlly equivalent to the additive model: for every tree generated, $\hat{f}^b(x)$ is fit with $d = 1$ split to the training data X, which amounts to single-variable linear model $\hat{f}_j(x)$. In total there are b_max trees, so there are b_max linear models generated.

Finally, the output of boosting is represented by

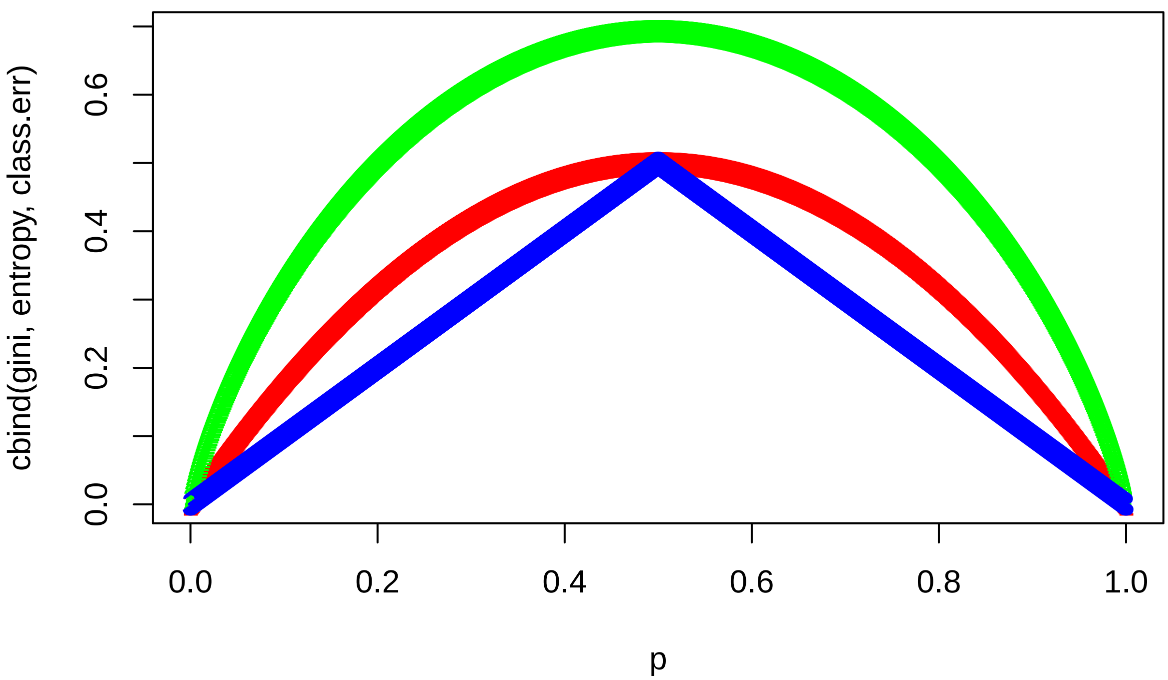
$$\hat{f}(x) = \sum_{b=1}^{b_{max}} \lambda \hat{f}^b(x)$$

, taking λ out of the sum, it can be writen as a form of additive model.

$$\hat{f}(x) = \lambda \sum_{j=1}^{b_{max}} \hat{f}_j(x)$$

8.4.3

```
p <- seq(0, 1, 0.001)
gini <- 2*p*(1 - p)
entropy <- -(p * log(p) + (1 - p) * log(1 - p))
class.err <- 1 - pmax(p, 1 - p)
matplot(p, cbind(gini, entropy, class.err), col = c("red", "green", "blue"))
```



8.4.5

```
p <- c(0.1, 0.15, 0.2, 0.2, 0.55, 0.6, 0.6, 0.65, 0.7, 0.75)
```

majority vote approach

```r
sum(p >= 0.5) < sum(p < 0.5)
```

```
## [1] FALSE
```

There are more red prediction, thus, it's RED

average approach

```r
mean(p)
```

```
## [1] 0.45
```

The average predicted probability is $0.45 < 0.5$, thus, it's GREEN.

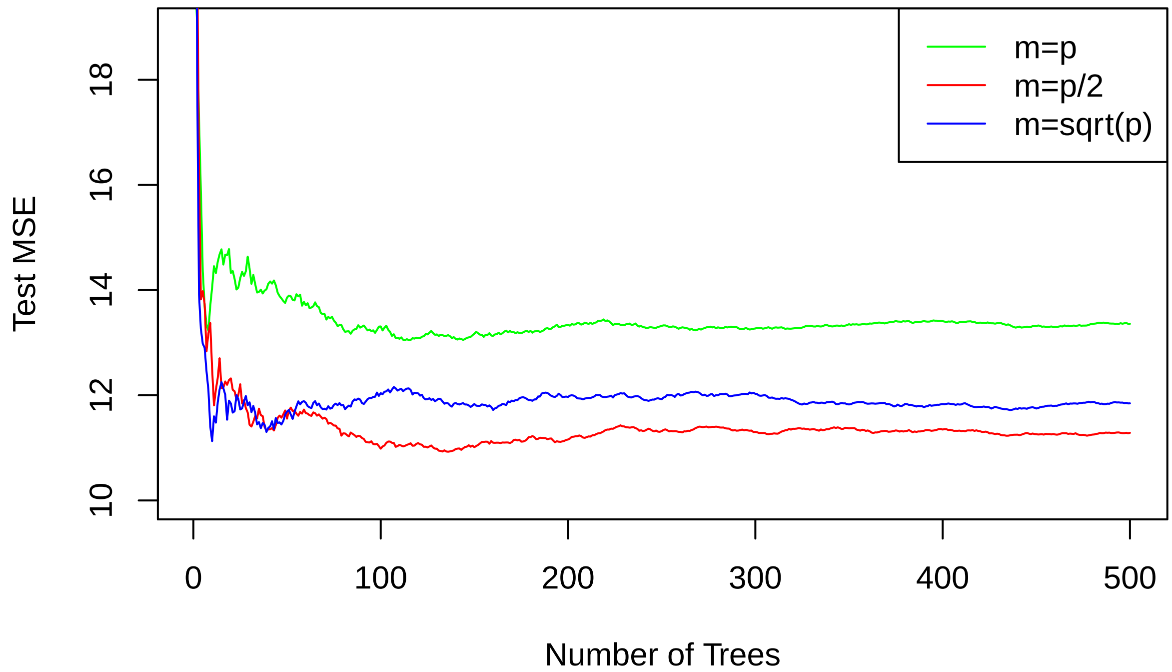
8.4.7

```r
attach(Boston)
set.seed(1)
train <- sample(1:nrow(Boston), nrow(Boston)/2)
X.train <- Boston[train, -14]
X.test <- Boston[-train, -14]
Y.train <- Boston[train, 14]
Y.test <- Boston[-train, 14]

p <- dim(Boston)[2] - 1
p_2 <- p/2
p_sq <- sqrt(p)

rf_p <- randomForest(X.train, Y.train, xtest = X.test, ytest = Y.test,
    mtry = p, ntree = 500)
rf_p_2 <- randomForest(X.train, Y.train, xtest = X.test, ytest = Y.test,
    mtry = p_2, ntree = 500)
rf_p_sq <- randomForest(X.train, Y.train, xtest = X.test, ytest = Y.test,
    mtry = p_sq, ntree = 500)

plot(1:500, rf_p$test$mse, col = "green", type = "l", xlab = "Number of Trees",
    ylab = "Test MSE", ylim = c(10, 19));lines(1:500, rf_p_2$test$mse, col = "red", type = "l");lines(1
```



m=p/2 seems to provide the smallest test MSE. When the number of trees gets larger, this result become stable.

8.4.8

part a

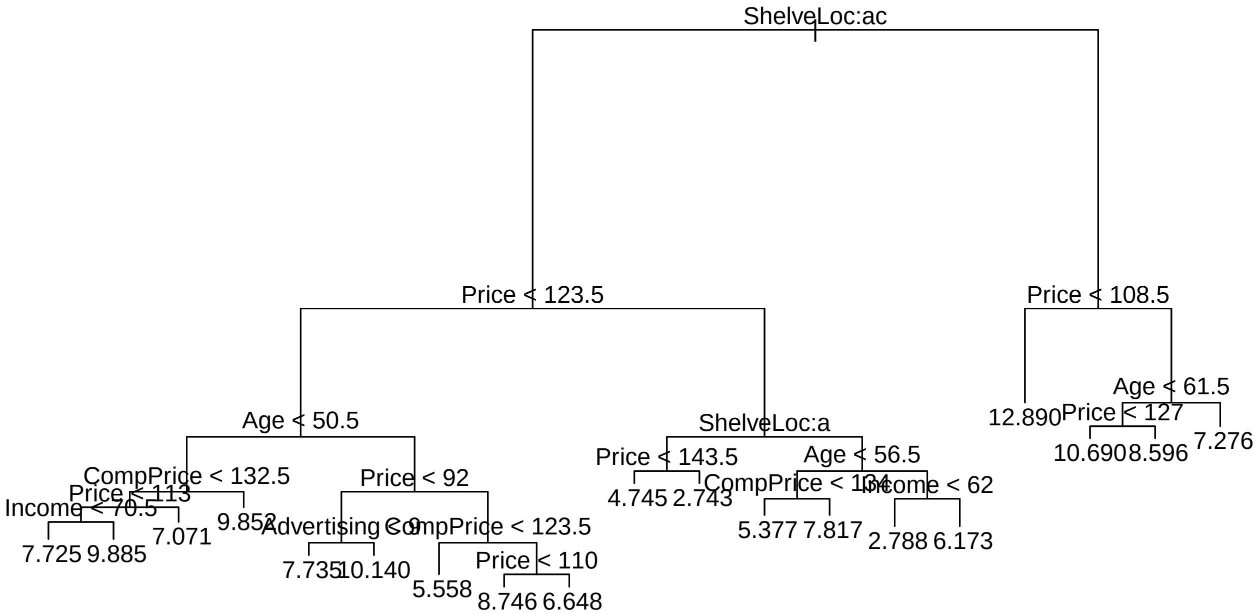
```
attach(Carseats)
train <- sample(1:nrow(Carseats), nrow(Carseats)/2)
Carseats_train <- Carseats[train, ]
Carseats_test <- Carseats[-train, ]
```

part b

```
tree_carseats <- tree(Sales ~ ., data = Carseats_train)
summary(tree_carseats)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = Carseats_train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Price"       "Age"         "CompPrice"   "Income"
## [6] "Advertising"
## Number of terminal nodes:  19
## Residual mean deviance:  2.306 = 417.4 / 181
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.26500 -0.90750 -0.04562  0.00000  1.01900  3.38000
```

```
plot(tree_carseats)
text(tree_carseats)
```



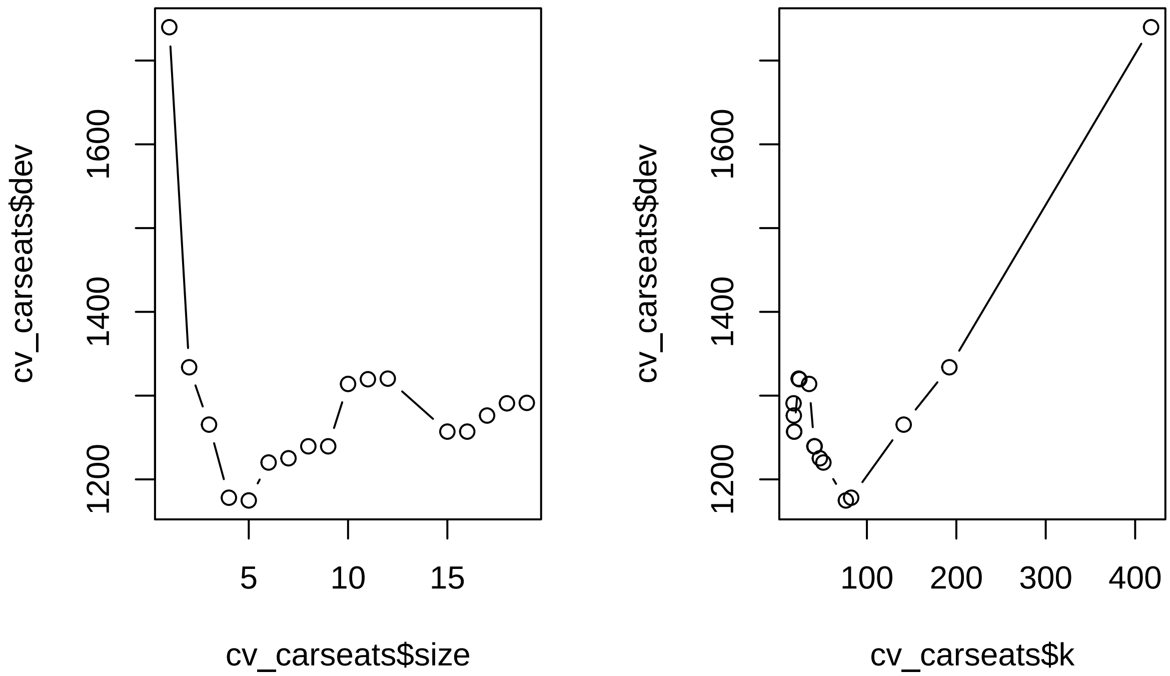

```
pred_carseats <- predict(tree_carseats, Carseats_test)
mse <- mean((Carseats_test$Sales - pred_carseats)^2)
mse
```

```
## [1] 4.704263
```

Test MSE is 4.71

part c

```
cv_carseats <- cv.tree(tree_carseats, FUN = prune.tree)
par(mfrow = c(1, 2))
plot(cv_carseats$size, cv_carseats$dev, type = "b")
plot(cv_carseats$k, cv_carseats$dev, type = "b")
```



The test MSE is the smallest when size is 9

```
pruned_carseats <- prune.tree(tree_carseats, best = 9)
pred_pruned <- predict(pruned_carseats, Carseats_test)
pruned_mse <- mean((Carseats_test$Sales - pred_pruned)^2)
pruned_mse
```

```
## [1] 4.52989
```

Pruning the tree does not improves the test MSE (from 4.71 to 5.45).

part d

```
bag_carseats <- randomForest(Sales ~ ., data = Carseats_train, mtry = 10, ntree = 500, importance = T)
bag_pred <- predict(bag_carseats, Carseats_test)
hag_mse <- mean((Carseats_test$Sales - bag_pred)^2)
hag_mse
```

```
## [1] 2.47614
```

```
importance(bag_carseats)
```

```
##               %IncMSE IncNodePurity
## CompPrice   20.255434    143.621120
## Income       8.030479     95.521905
## Advertising  9.113427     75.897281
## Population   1.011174     62.633198
## Price       55.679505    551.304273
## ShelveLoc   59.353315    507.781908
## Age         14.214033    156.726871
## Education    2.673099     55.170619
## Urban       -1.558460      8.917585
## US           0.271480      8.300157
```

Bagging improves the test MSE to 2.91. CompPrice,ShelveLoc and Price are the most important variables.

part e

```
rf_carseats <- randomForest(Sales ~ ., data = Carseats_train, mtry = 8, ntree = 500, importance = T)
rf_pred <- predict(rf_carseats, Carseats_test)
rf_mse<-mean((Carseats_test$Sales - rf_pred)^2)
rf_mse
```

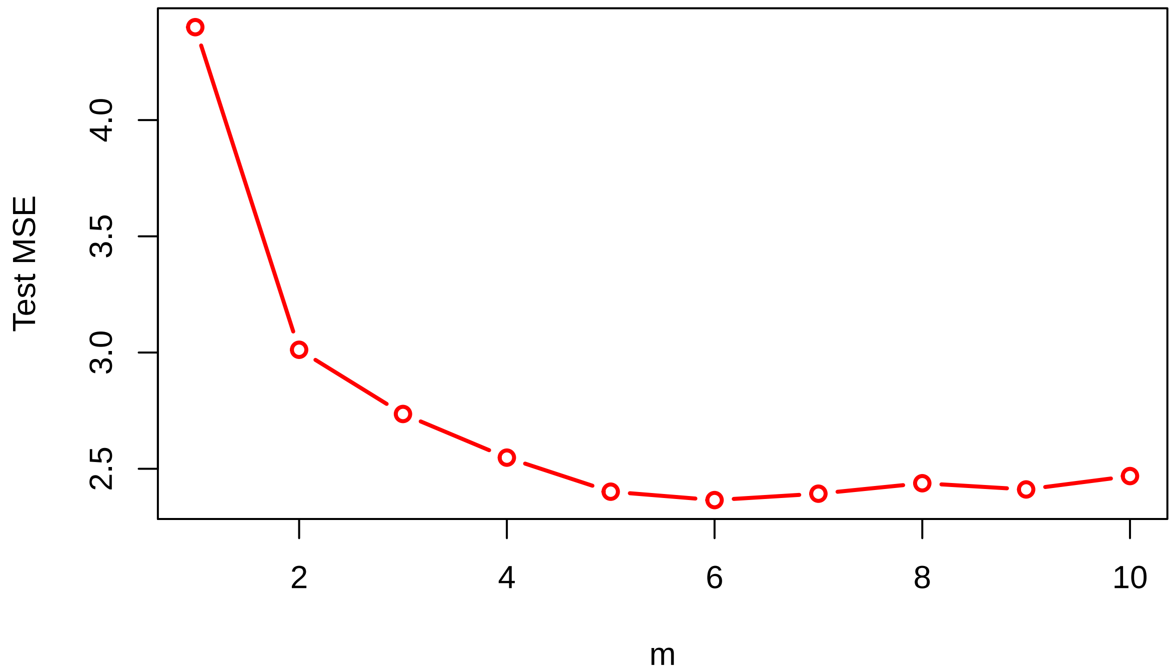
```
## [1] 2.380419
```

```
importance(rf_carseats)
```

```
##                 %IncMSE IncNodePurity
## CompPrice   18.08153539    146.805587
## Income       7.28442087     96.001096
## Advertising  8.43128558     75.156087
## Population  -0.76485895     67.419272
## Price       51.35108180    525.358133
## ShelveLoc   57.90822915    504.874241
## Age         13.94944627    175.527905
## Education    0.75967142     58.020287
## Urban       -1.40413862      9.504252
## US          -0.01818094     10.638819
```

Random forest improves the test MSE to 2.93. Advertising,ShelveLoc and Price are the most important variables.

```
p <- ncol(Carseats) - 1
errors <- c()
for ( d in seq(1, p) ) {
  rf_carseats <- randomForest(Sales~., data = Carseats, subset = train, mtry = d, ntree = 500)
  rf_pred <- predict(rf_carseats, Carseats_test)
  errors[d] <- mean((Carseats_test$Sales - rf_pred)^2)
}
plot(errors, col = "red", type = "b", lwd = 2, xlab = "m", ylab = "Test MSE")
```




```
which.min(errors)
```

```
## [1] 6
```

when $m = 8$, the test MSE is lowest

8.3.11

part a

```
attach(Caravan)
train = 1:1000
caravan <- Caravan
caravan$Purchase = ifelse(caravan$Purchase == "Yes", 1, 0)
caravan_train = caravan[train, ]
caravan_test = caravan[-train, ]
```

part b

```
set.seed(3)


boost_caravan <- gbm(Purchase~., data=caravan_train,
                    n.trees=1000, shrinkage=0.01, distribution="bernoulli", cv.folds = 10)
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 50: PVRAAUT has no variation.
```

```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution =
## distribution, : variable 71: AVRAAUT has no variation.
```

```r
head(summary.gbm(boost_caravan,plotit = FALSE),10)
```

```
##               var   rel.inf
## PPERSAUT PPERSAUT 14.822726
## MKOOPKLA MKOOPKLA  9.647725
## MOPLHOOG MOPLHOOG  6.516380
## MBERMIDD MBERMIDD  5.667377
## PBRAND     PBRAND  5.425667
## MGODGE     MGODGE  4.172987
## ABRAND     ABRAND  4.043386
## MINK3045 MINK3045  3.980106
## MOSTYPE   MOSTYPE  2.836807
## MSKA         MSKA  2.572698
```

part c

```r
boost_prob = predict(boost_caravan, caravan_test, n.trees = 1000, type = "response")
boost_pred = ifelse(boost_prob > 0.2, 1, 0)
table(caravan_test$Purchase, boost_pred)
```

```
##    boost_pred
##        0    1
##   0 4413  120
##   1  257   32
```

```r
32/(32+120)
```

```
## [1] 0.2105263
```

More than 20% of positive prediction are correct.

```r
lm_caravan = glm(Purchase ~ ., data = caravan_train, family = binomial)
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```r
lm_prob = predict(lm_caravan, caravan_test, type = "response")
```

```
## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading
```

```r
lm_pred = ifelse(lm_prob > 0.2, 1, 0)
table(caravan_test$Purchase, lm_pred)
```

```
##    lm_pred
##        0    1
##   0 4183  350
##   1  231   58
```

```r
58/(58+350)
```

```
## [1] 0.1421569
```

only 14% of positive prediction are correct. Boosting makes a better predictiom.