STATS 202: Data Mining and Analysis
Instructor: Linh Tran

HOMEWORK # 4
Due date: August 20, 2021

Stanford University

Introduction

Homework problems are selected from the course textbook: *An Introduction to Statistical Learning*.

Problem 1 (10 points)

Chapter 8, Exercise 4 (p. 332).

(a) The tree corresponding to the given partitions is shown below.

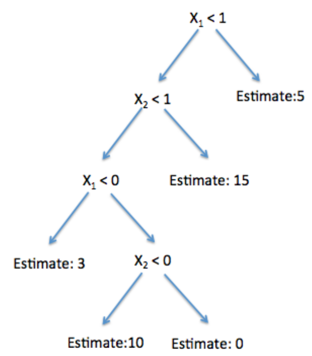


Figure 1: The binary decision tree matching Figure 8.12a

(b) The partitioning of the feature space corresponding to the given tree is shown below.

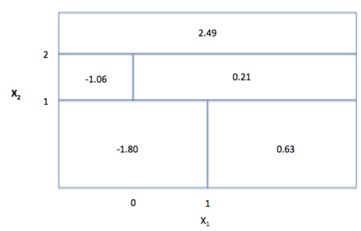


Figure 2: The partitioning of the feature space matching Figure 8.12b

Problem 2 (10 points)

Chapter 8, Exercise 8 (p. 333).

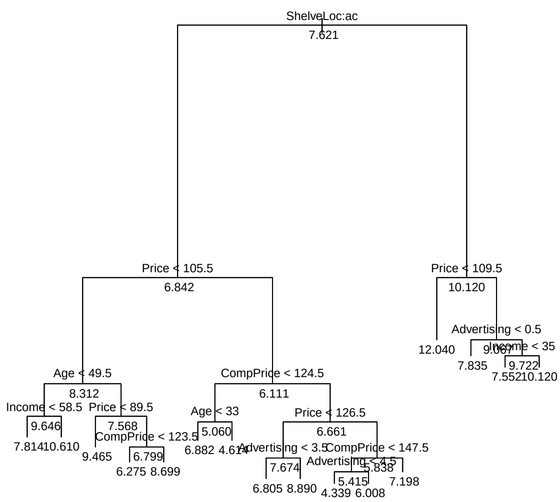
```
library(tree)
library(randomForest)
library(ISLR)
data(Carseats)
```

(a)
```
set.seed(1)
n <- nrow(Carseats)
training.idx <- sample(n, floor(0.8*n))
data.train <- Carseats[training.idx,]
data.test <- Carseats[-training.idx,]
```

(b)
```
tree.fit <- tree(Sales ~ ., data = data.train)
summary(tree.fit)
```

```
##
## Regression tree:
## tree(formula = Sales ~ ., data = data.train)
## Variables actually used in tree construction:
## [1] "ShelveLoc"    "Price"        "Age"          "Income"       "CompPrice"
## [6] "Advertising"
## Number of terminal nodes:  16
## Residual mean deviance:  2.572 = 781.9 / 304
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -4.45400 -1.07000 -0.05544  0.00000  1.14500  4.69600
```

```
plot(tree.fit)
text(tree.fit, all=TRUE, cex=.75)
```



```r
y.hat <- predict(tree.fit, newdata=data.test)
lm.mse <- mean((data.test$Sales - y.hat)^2)
cat(sprintf('Test error: %0.3f\n', lm.mse))
```

```
## Test error: 4.936
```

From the summary, we see that 6 variables are utilized in the tree. The training MSE is 2.572, while the test MSE is 4.936. Notice that the first split in the tree depends on the categorical variable `ShelveLoc`.

(c)

```r
tree.fit.cv <- cv.tree(tree.fit)
tree.fit.cv
```

```
## $size
##  [1] 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
##
## $dev
##  [1] 1478.442 1492.616 1498.950 1548.422 1576.654 1554.403 1554.494
##  [8] 1551.629 1563.750 1668.391 1668.149 1664.479 1691.338 1801.638
## [15] 1898.187 2522.138
##
## $k
##  [1]      -Inf  27.89334  28.71882  33.94500  36.84035  39.52580  45.45153
##  [8]  50.71762  51.19958  75.88308  80.34770  89.19748  94.32552 153.52516
## [15] 262.05898 623.54522
##
## $method
## [1] "deviance"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

The CV fits show us that the tree of size 16, as it results in the lowest error. If we apply the 1 standard error rule, we could instead rely upon a tree of size 14.

```r
tree.fit.prune <- prune.tree(tree.fit, best=14)
y.hat.prune <- predict(tree.fit.prune, newdata=data.test)
tree.mse <- mean((data.test$Sales - y.hat.prune)^2)
cat(sprintf('Test error: %0.3f\n', tree.mse))
```

```
## Test error: 4.923
```

Pruning appears to have a negligible effect on our MSE. However, we may still want to stick with the pruned model as it is more parsimonious.

(d)

```r
p <- ncol(Carseats) - 1
bag.fit <- randomForest(Sales ~., data=data.train, mtry=p, importance=T)
bag.fit
```

```
##
## Call:
##  randomForest(formula = Sales ~ ., data = data.train, mtry = p,      importance = T)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
```

```
##
##           Mean of squared residuals: 2.435155
##                     % Var explained: 68.52
```

We see that the training MSE is 2.435, which is lower than the training MSE from the single tree fit. Let's check the test MSE.

```
y.hat.bag <- predict(bag.fit, newdata=data.test)
bag.mse <- mean((data.test$Sales - y.hat.bag)^2)
cat(sprintf('Test error: %0.3f\n', bag.mse))

## Test error: 2.953
```

The test error of 2.953 is considerably lower than for the single tree fit (which is 4.923 for the pruned tree).

```
importance(bag.fit)

##               %IncMSE IncNodePurity
## CompPrice   35.238883     256.78439
## Income      10.299522     140.24737
## Advertising 23.002369     193.54415
## Population  -2.401561      69.76428
## Price       80.085452     741.31493
## ShelveLoc   80.270597     709.25579
## Age         25.943974     239.79209
## Education    2.149399      61.41957
## Urban       -1.614686      10.23359
## US           4.214299      10.17821
```

From a quick glance at the importance table, we see that Price and ShelveLoc are individually crucial for getting down MSE, followed by CompPrice, Age and Advertising. Other variables have low or no effects on the OOB MSE. Using the reduction in node impurity (given in the second column) to quantify importance also supports the same conclusion.

(e)

```
mtrys <- c(2, 3, 5)
rf.fits <- list()
for (m in mtrys) {
  rf.fits[[m]] <- randomForest(Sales ~ ., data=data.train, mtry=m,
                               importance=T)
  cat(sprintf('Train error (m=%d): %0.3f\n', m, mean(rf.fits[[m]]$mse)))
}

## Train error (m=2): 3.381
## Train error (m=3): 2.995
## Train error (m=5): 2.732
```

The training errors appear to be higher than the training error from bagging (which is 2.435). Lets check the test MSE.

```
for (m in mtrys) {
  y.hat.rf <- predict(rf.fits[[m]], newdata=data.test)
  rf.mse <- mean((data.test$Sales - y.hat.rf)^2)
  cat(sprintf('Test error (m=%d): %0.3f\n', m, rf.mse))
}
```

```
## Test error (m=2): 3.964
## Test error (m=3): 3.486
## Test error (m=5): 3.050
```

The test error appears to be the best for $m = 5$, slightly higher than for the bagging estimate (which is 2.953). Thus, it doesn't appear that random sampling of the predictors helps in this case. Let's check the importance for $m = 5$.

```
importance(rf.fits[[5]])
```

```
##                %IncMSE IncNodePurity
## CompPrice   26.9416170     238.39091
## Income       7.2124864     160.32005
## Advertising 19.2067911     200.95266
## Population  -0.6486601     110.39048
## Price       66.8165202     670.28420
## ShelveLoc   63.5767240     629.88252
## Age         19.7478280     262.91238
## Education    1.2232333      77.48801
## Urban       -2.9280689      14.72315
## US           4.4659804      26.34761
```

The importance table for random forest tells a very similar story to bagging.

Problem 3 (10 points)

Chapter 8, Exercise 10 (p. 334).

```
library(gbm)
library(glmnet)
library(ggplot2)
library(ISLR)
data(Hitters)
```

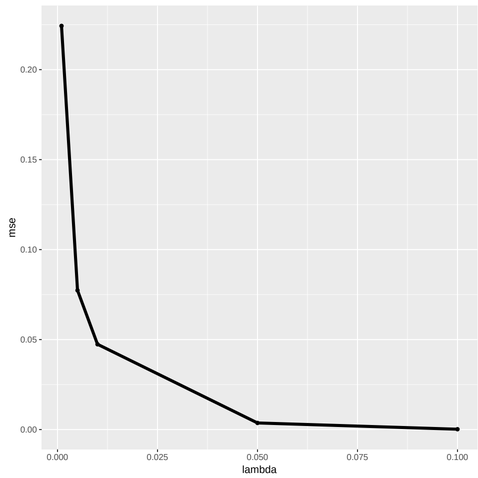
(a)

```
Hitters.updated <- subset(Hitters, !is.na(Salary))
Hitters.updated$log.salary <- log(Hitters.updated$Salary)
```

(b)

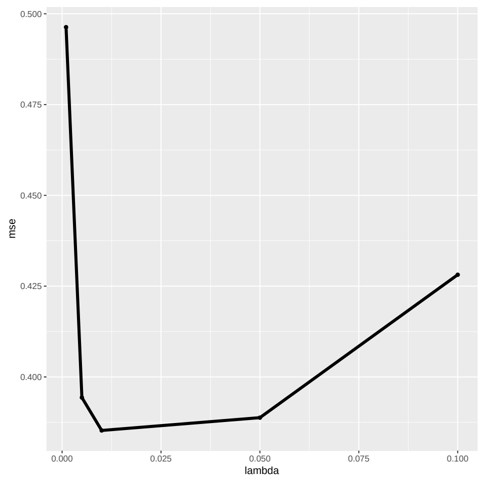
```
n <- nrow(Hitters.updated)
training.idx <- sample(n, 200)
data.train <- Hitters.updated[training.idx,]
data.test <- Hitters.updated[-training.idx,]
X <- model.matrix(
    ~ -1 + ., data.train[
        ,-which(names(Hitters.updated) %in% c('Salary', 'log.salary'))])
newX <- model.matrix(
    ~ -1 + ., data.test[
        ,-which(names(Hitters.updated) %in% c('Salary', 'log.salary'))])
```

(c)
```r
lambdas <- c(0.001, 0.005, 0.01, 0.05, 0.1)
train.mses <- rep(NA, length(lambdas))
test.mses <- rep(NA, length(lambdas))
names(train.mses) <- names(test.mses) <- lambdas

for(i in c(1:length(lambdas))) {
  boost.fit <- gbm(log.salary ~ . - Salary, data=data.train,
                   distribution='gaussian', n.trees=1000,
                   interaction.depth=4, shrinkage=lambdas[i])
  y.hat <- predict(boost.fit, newdata=data.test, n.trees=1000)
  test.mses[i] <- mean((data.test$log.salary - y.hat)^2)
  train.mses[i] <- tail(boost.fit$train.error, 1)
}

mse.df <- data.frame(lambda=lambdas, mse=train.mses, mse.type='train')
ggplot(data=mse.df, aes(x=lambda, y=mse)) +
  geom_line(size=1.5) + geom_point()
```



As expected, we see that holding the maximum number of trees fixed at 1000, as λ increases the training error falls.

(d)
```r
mse.df <- data.frame(lambda=lambdas, mse=test.mses, mse.type='test')
ggplot(data=mse.df, aes(x=lambda, y=mse)) +
  geom_line(size=1.5) + geom_point()
```



As expected, we see that the test error has the typical U-shape (indicative of overfitting for large λ). The best achieved test error is about 0.385.

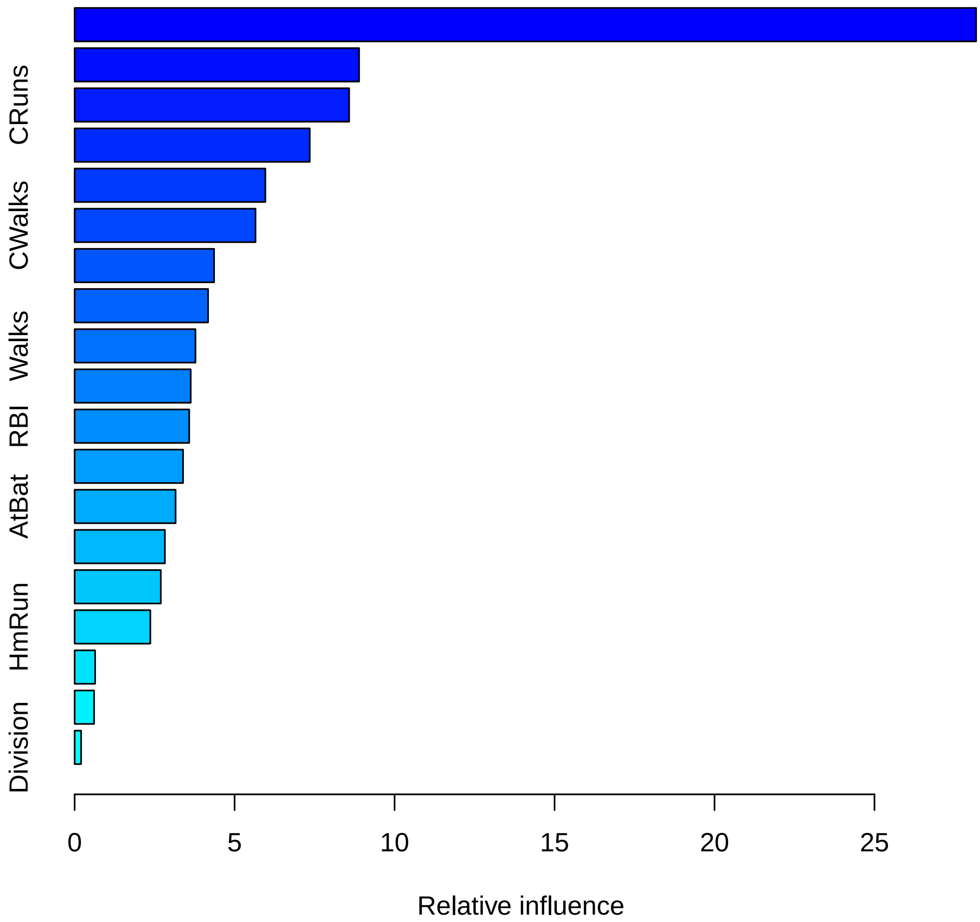
(e) We compare boosting to a simple linear model and Lasso.

```
  lm.fit <- lm(log.salary ~ . - Salary, data=data.train)
  lm.pred <- predict(lm.fit, newdata=data.test)
  lm.mse <- mean((lm.pred - data.test$log.salary)^2)
  cat(sprintf('Test error (Linear): %0.3f\n', lm.mse))

## Test error (Linear): 0.545

  cv.lasso.fit <- cv.glmnet(X, data.train$log.salary, alpha=1)
  lasso.fit <- glmnet(X, data.train$log.salary, alpha=1)
  best.lambda <- cv.lasso.fit$lambda.min
  lasso.pred <- predict(lasso.fit, s=best.lambda, newx=newX)
  lasso.mse <- mean((lasso.pred - data.test$log.salary)^2)
  cat(sprintf('Test error (Lasso): %0.3f\n', lasso.mse))

## Test error (Lasso): 0.522

  cat(sprintf('\tBest lambda: %0.3f', best.lambda))

##  Best lambda: 0.021
```

We see that both models have relatively close MSE, as the LASSO penalty of $\lambda = 0.02$ isn't very large. These are both about 42% worst than boosting.

(f)

```
  boost.fit <- gbm(log.salary ~ . - Salary, data=data.train,
                    distribution="gaussian", n.trees=1000,
                    interaction.depth=4, shrinkage=0.05)
  summary(boost.fit)
```




```
##                   var    rel.inf
## CAtBat         CAtBat 28.1769915
## CRBI             CRBI  8.8914643
## CRuns           CRuns  8.5794398
## Years           Years  7.3493828
## PutOuts       PutOuts  5.9604796
## CWalks         CWalks  5.6546387
## CHits           CHits  4.3579874
## CHmRun         CHmRun  4.1700413
## Walks           Walks  3.7768167
## Hits             Hits  3.6284113
## RBI               RBI  3.5804868
## Runs             Runs  3.3884204
## AtBat           AtBat  3.1557812
## Errors         Errors  2.8213142
## Assists       Assists  2.6933807
## HmRun           HmRun  2.3659863
## League         League  0.6402469
## NewLeague NewLeague  0.6069830
## Division     Division  0.2017469
```

The most signifcant variable is clearly `CAtBat`, followed by a long list of somewhat helpful predictors, e.g. CRBI, CRuns, Years, etc.

(g)
```
p <- ncol(Hitters) - 1
bag.fit <- randomForest(log.salary ~ . - Salary, data=data.train, mtry=p,
                        importance=T)
y.hat.bag <- predict(bag.fit, newdata=data.test)
bag.mse <- mean((data.test$log.salary - y.hat.bag)^2)
cat(sprintf('Test error: %0.3f\n', bag.mse))

## Test error: 0.337
```

Bagging achieves a test MSE of about 0.337, which is actually slightly better than boosting.

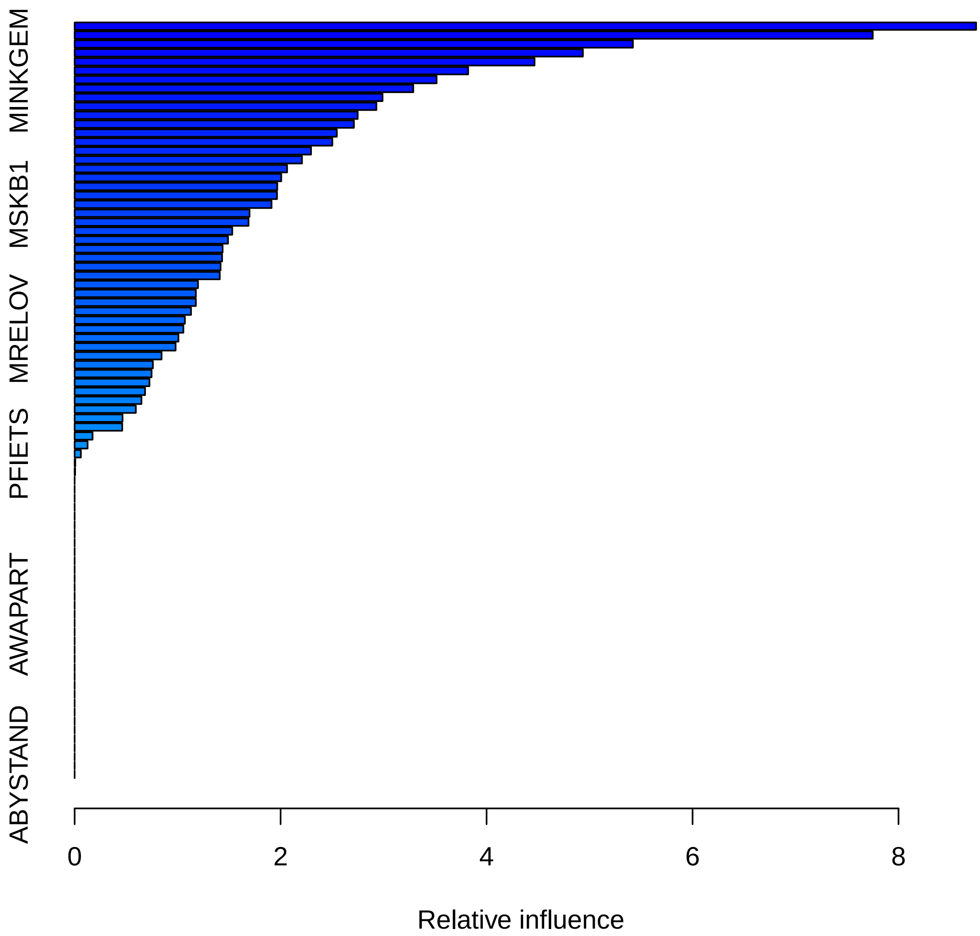
Problem 4 (10 points)

Chapter 8, Exercise 11 (p. 335).

```
data(Caravan)
```

(a)
```
Caravan$Purchase <- ifelse(Caravan$Purchase=='Yes', 1, 0)
n <- nrow(Caravan)
training.idx <- sample(n, 1000)
data.train <- Caravan[training.idx,]
data.test <- Caravan[-training.idx,]
X <- model.matrix(
    ~ -1 + ., data.train[
        ,-which(names(Caravan) == 'Purchase')])
newX <- model.matrix(
    ~ -1 + ., data.test[
        ,-which(names(Caravan) == 'Purchase')])
```

(b)
```
boost.fit <- gbm(Purchase ~ ., data=data.train, distribution='bernoulli',
                 n.trees=1000, interaction.depth=4, shrinkage=0.01)

## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution,
: variable 60: PZEILPL has no variation.

## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution,
: variable 81: AZEILPL has no variation.

summary(boost.fit)
```




```
##                  var    rel.inf
## PPERSAUT PPERSAUT 8.75353043
## MOSTYPE   MOSTYPE 7.74983116
## MBERMIDD MBERMIDD 5.42083568
## MOPLLAAG MOPLLAAG 4.93487103
## MOPLMIDD MOPLMIDD 4.46590823
## MINKGEM   MINKGEM 3.82125258
## PWAPART   PWAPART 3.51503932
## MKOOPKLA MKOOPKLA 3.28805969
## MAUT1       MAUT1 2.99005316
## MGODGE     MGODGE 2.93014623
## PBRAND     PBRAND 2.74873940
## MFWEKIND MFWEKIND 2.71180127
## MINK7512 MINK7512 2.54599457
## MINK3045 MINK3045 2.50222587
## MOPLHOOG MOPLHOOG 2.29547304
## MBERHOOG MBERHOOG 2.20747930
## MFALLEEN MFALLEEN 2.06302864
## APERSAUT APERSAUT 2.00632956
## MHHUUR     MHHUUR 1.96762557
## MBERARBG MBERARBG 1.96491638
```

```
## MSKB1       MSKB1 1.91213316
## MGODPR     MGODPR 1.69841560
## MFGEKIND MFGEKIND 1.68884397
## MAUT2       MAUT2 1.53070196
## MSKB2       MSKB2 1.48931781
## MZPART     MZPART 1.43697019
## MGEMOMV   MGEMOMV 1.43195340
## MGEMLEEF MGEMLEEF 1.41714700
## MSKC         MSKC 1.40931355
## MBERARBO MBERARBO 1.19789676
## MSKA         MSKA 1.17731566
## MINKM30   MINKM30 1.17730978
## MRELGE     MRELGE 1.13010441
## MHKOOP     MHKOOP 1.07075812
## MRELOV     MRELOV 1.05594602
## MINK4575 MINK4575 1.00802191
## MAUTO       MAUTO 0.98057760
## MGODOV     MGODOV 0.84449551
## MGODRK     MGODRK 0.76004954
## PLEVEN     PLEVEN 0.74717733
## MBERZELF MBERZELF 0.72645403
## MSKD         MSKD 0.68390315
## MOSHOOFD MOSHOOFD 0.64878138
## MZFONDS   MZFONDS 0.59426244
## MINK123M MINK123M 0.46527526
## MRELSA     MRELSA 0.46209203
## ALEVEN     ALEVEN 0.17392169
## MBERBOER MBERBOER 0.12576795
## PFIETS     PFIETS 0.06074438
## MAANTHUI MAANTHUI 0.00638292
##  [ reached 'max' / getOption("max.print") -- omitted 35 rows ]
```

PPERSAUT appears to be the most important variable, followed by MOSTYPE and MBERMIDD.

(c)

```
boost.pred <- predict(boost.fit, newdata=data.test, n.trees=1000,
                      type='response')
y.hat <- ifelse(boost.pred > 0.20, 1, 0)
table(y.hat, data.test$Purchase)
```

```
##
## y.hat    0    1
##     0 4331  249
##     1  200   42
```

We see that $42/(200+42) \approx 17.4\%$ of the the people predicted to make a purchase actually do.

```
glm.fit <- glm(Purchase ~ ., data=data.train, family='binomial')

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

glm.pred <- predict(glm.fit, newdata=data.test, type='response')

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

y.hat <- ifelse(glm.pred < 0.20, 1, 0)
table(y.hat, data.test$Purchase)
```

```
##
## y.hat     0    1
##     0   372   69
##     1  4159  222
```

Using logistic regression, we see that the precision is $275/(4431+275) \approx 6.8\%$. This is noticeably lower than the boosting model.

```
knn.fit <- knn(X, newX, cl=factor(data.train$Purchase), k=3, prob=T)
table(knn.fit, data.test$Purchase)
```

```
##
## knn.fit    0    1
##       0 4477  280
##       1   54   11
```

Using KNN with $k=3$, we see that the precision is $11/(54+11) \approx 16.9\%$.

Problem 5 (10 points)

Let $x_i : i = 1, ..., p$ be the input predictor values and $a_k^{(2s)} : k = 1, ..., K$ be the K-dimensional output from a 2-layer and M-hidden unit neural network with sigmoid activation $\sigma(a) = \{1 + e^{-a}\}^{-1}$ such that

$$a_j^{(1s)} = w_{j0}^{(1s)} + \sum_{i=1}^{p} w_{ji}^{(1s)} x_i : j = 1, ..., M$$
$$a_k^{(2s)} = w_{k0}^{(2s)} + \sum_{j=1}^{M} w_{kj}^{(2s)} \sigma\left(a_j^{(1s)}\right)$$

Show that there exists an equivalent network that computes exactly the same output values, but with hidden unit activation functions given by $tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$, i.e.

$$a_j^{(1t)} = w_{j0}^{(1t)} + \sum_{i=1}^{p} w_{ji}^{(1t)} x_i : j = 1, ..., M$$
$$a_k^{(2t)} = w_{k0}^{(2t)} + \sum_{j=1}^{M} w_{kj}^{(2t)} \tanh\left(a_j^{(1t)}\right)$$

Hint: first derive the relation between $\sigma(a)$ and $tanh(a)$. Then show that the parameters of the two networks differ by linear transformations.

We first show the relation between $\sigma(a)$ and $tanh(a)$:

$$\begin{aligned}
\tanh(a) &= \frac{e^a - e^{-a}}{e^a + e^{-a}} \\
&= -1 + \frac{2e^a}{e^a + e^{-a}} \\
&= -1 + 2\frac{1}{1 + e^{-a}} \\
&= 2\sigma(2a) - 1
\end{aligned}$$

Consequently, we have that

$$
\begin{aligned}
a_k^{(2t)} &= w_{k0}^{(2t)} + \sum_{j=1}^{M} w_{kj}^{(2t)} \tanh\left(a_j^{(1t)}\right) \\
&= w_{k0}^{(2t)} + \sum_{j=1}^{M} w_{kj}^{(2t)} \left[2\sigma\left(2a_j^{(1t)}\right) - 1\right] \\
&= \left[w_{k0}^{(2t)} - \sum_{j=1}^{M} w_{kj}^{(2t)}\right] + \sum_{j=1}^{M} 2w_{kj}^{(2t)}\sigma\left(2a_j^{(1t)}\right)
\end{aligned}
$$

Thus, to make the two networks equivalent we set

$$
\begin{aligned}
w_{k0}^{(2t)} &= w_{k0}^{(2t)} - \sum_{j=1}^{M} w_{kj}^{(2t)} \\
w_{kj}^{(2t)} &= 2w_{kj}^{(2t)} \\
a_j^{(1s)} &= 2a_j^{(1t)}
\end{aligned}
$$

We can satisfy the third condition by updating the weights for the first layer such that

$$
\begin{aligned}
w_{j0}^{(1s)} &= 2w_{j0}^{(1t)} \\
w_{ji}^{(1s)} &= 2w_{ji}^{(1t)}
\end{aligned}
$$