DecisionTree-HW

Franky Zhang

3/5/2022

8.1

Draw an example (of your own invention) of a partition of two- dimensional feature space that could result from recursive binary splitting. Your example should contain at least six regions. Draw a decision tree corresponding to this partition. Be sure to label all as- pects of your figures, including the regions R1, R2, . . ., the cutpoints t1,t2,. . . , and so forth.

Answer:

See Appendix

8.2

Answer:

Firstly, fit a decision stump to training set $(x_i, y_i), i = 1, 2, ..., N$, denote this stump as $f_1(X)$. Then calculate residual $r^1 = Y - \lambda f_1(X)$ and fit the second decision to set $(x_i, r_i^1), i = 1, 2, ..., N$, denoting the second stump as $f_2(X)$. At this stage, the latest model is $f(X) = f_1(X) + f_2(X)$. Repeat the stage for p times and get the final model $f(X) = f_1(X) + f_2(X) + ... + f_p(X)$.

8.3

Answer:

```
p <- seq(from = .01, to = .99, by = .01)
gini_index <- p* (1 - p)* 2
classification_error <- c()
for (i in 1: length(p)) {
  classification_error[i] <- 1 - max(p[i], 1 - p[i])
}
entropy <- -(p* log(p, base = 2) + (1 - p)* log(1-p, base = 2))
data <- data.frame(
  rbind(cbind(p = p, value = gini_index,
                        type = rep("gini_index", length(gini_index))),
        cbind(p = p, value = classification_error,
              type = rep("classification_error", length(classification_error))),
        cbind(p = p, value = entropy, type = rep("entropy", length(entropy)))))
data$type <- factor(data$type)
data$p <- as.numeric(data$p)
data$value <- as.numeric(data$value)
ggplot(data = data) +
```

```
  geom_line(aes(x = p, y = value, color = type, linetype = type), lwd = 1) +
  theme(legend.position = "bottom") +
  ggtitle("Camparsion of Gini, Class_err and entropy")
```

8.5

Answer:

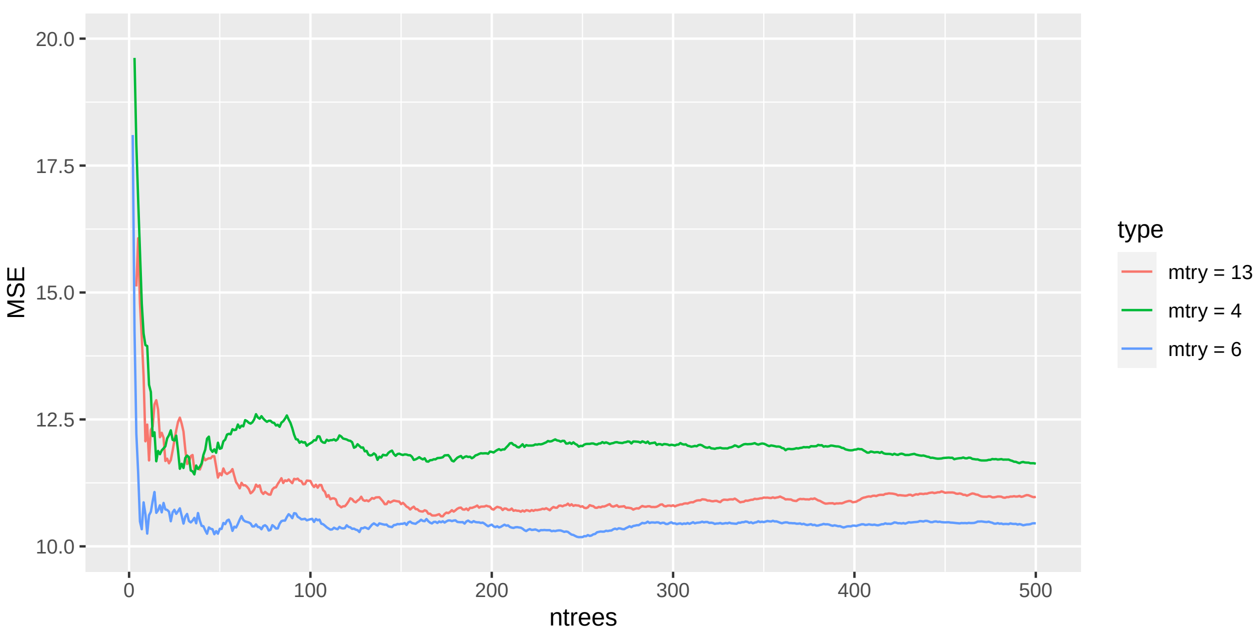
- *majority vote*: 6 vs 4
  - final classification: red
- *average probability*: .45
  - final classification: green

8.7

```
# Boston
# mtry: # of predictors to use; ntree: # of tree to grow
set.seed(1623)
train <- sample(1:nrow(Boston), 0.7*nrow(Boston))
train.boston <- Boston[train, ]
test.boston  <- Boston[-train, ]
train.X <- train.boston[, -14]
train.Y <- train.boston[, 14]
test.X <- test.boston[, -14]
test.Y <- test.boston[, 14]
```

```r
p <- ncol(train.X)
ntrees <- 500
# train.mse <- c()
# test.mse <- c()
# for (i in 1: ntrees) {
#   rf.boston <- randomForest(x = train.X, y = train.Y,
#                            ntree = i, mtry = 6)
#   train.mse[i] <- mean((train.Y - predict(rf.boston, newdata = train.X))^2)
#   test.mse[i]  <- mean((test.Y  - predict(rf.boston, newdata =  test.X))^2)
# }
# train.mse
# test.mse

test.mse <- c()
type <- c()
mtry <- c(round(p/2), round(sqrt(p)), p)
for (i in mtry) {
  rf.boston <- randomForest(x = train.X, y = train.Y,
                            xtest = test.X, ytest = test.Y,
                            ntree = ntrees, mtry = i)
  test.mse <- c(rf.boston$test$mse, test.mse)
}
for (i in mtry) {
  type <- c(type, paste("mtry = ", rep(i, ntrees), sep = ""))
}
plot.data <- data.frame(
  ntrees = rep(c(1:ntrees), length(mtry)),
  MSE = test.mse,
  type = factor(type))
ggplot(data = plot.data) +
  geom_line(aes(x = ntrees, y = MSE, group = type, color = type)) + ylim(10, 20)
```



8.8

(a)

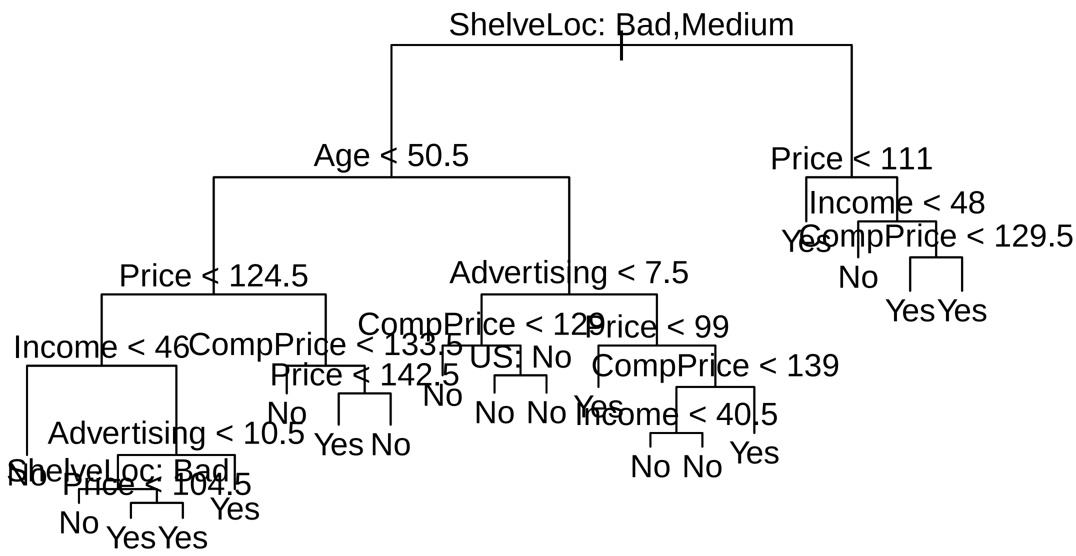
```r
# Carseats$Sales
rm(test.X, train.X, plot.data, train.boston, test.boston,
   mtry, ntrees, train.Y, test.Y, type)
set.seed(1141)
train <- sample(1:nrow(Carseats), 200)
High <- factor(ifelse(Carseats$Sales <= 8, "No", "Yes"))
train.X <- Carseats[train, -1]
test.X  <- Carseats[-train, -1]
train.Y <- High[train]
test.Y  <- High[-train]
training <- cbind(Sales = train.Y, train.X)
test <- cbind(Sales = test.Y, test.X)
```

(b)

```r
carseat.tree <- tree(Sales~., data = training)
summary(carseat.tree)
```

```
##
## Classification tree:
## tree(formula = Sales ~ ., data = training)
## Variables actually used in tree construction:
## [1] "ShelveLoc"   "Age"         "Price"        "Income"       "Advertising"
## [6] "CompPrice"   "US"
## Number of terminal nodes:  19
## Residual mean deviance:  0.4814 = 87.13 / 181
## Misclassification error rate: 0.11 = 22 / 200
```

```r
plot(carseat.tree)
text(carseat.tree, pretty = 0)
```

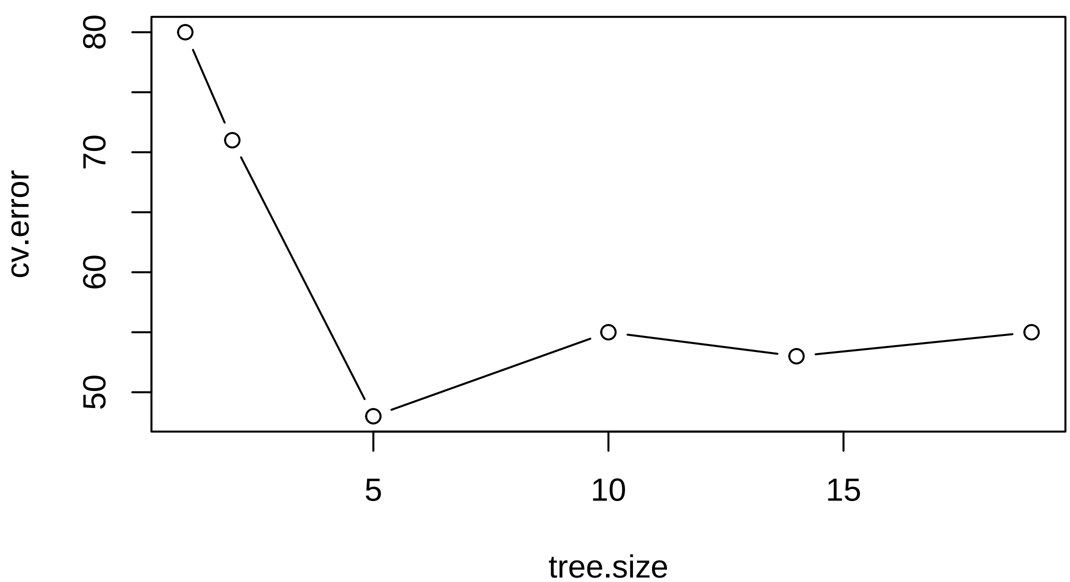



```
predict.Y <- predict(carseat.tree, newdata = test.X, type = "class")
confusion.mtrx <- table(test.Y, predict.Y)
test.MSE <- paste((confusion.mtrx[1, 1] + confusion.mtrx[2, 2])/nrow(test)*100,
                  "%", sep = "")
cat("test MSE = ", test.MSE)
```

```
## test MSE =  71.5%
```

(c)

```
carseat.cv <- cv.tree(carseat.tree, FUN = prune.misclass)
# k: alpha *|T|;
# dev: # of cv errors
plot(carseat.cv$size, carseat.cv$dev, type = "b", xlab = "tree.size", ylab = "cv.error")
```



```r
prune.carseats <- prune.misclass(carseat.tree, best = 8)
predict.Y <- predict(prune.carseats, newdata = test.X, type = "class")
confusion.mtrx <- table(test.Y, predict.Y)
test.MSE <- paste((confusion.mtrx[1, 1] + confusion.mtrx[2, 2])/nrow(test)*100,
                  "%", sep = "")
cat("new test MSE = ", test.MSE)
```

```
## new test MSE =  72%
```

Answer:

the performance of pruned tree on test data is nearly the same with unpruned tree.

(d)

```r
ntrees <- 500
mtry <- ncol(train.X)
bagging.carseats <- randomForest(x = train.X, y = train.Y,
                                 ntree = ntrees, mtry = mtry, importance = TRUE)
predict.Y <- predict(bagging.carseats, newdata = test.X, type = "class")
confusion.mtrx <- table(test.Y, predict.Y)
test.MSE <- paste((confusion.mtrx[1, 1] + confusion.mtrx[2, 2])/nrow(test)*100,
                  "%", sep = "")
cat("new test MSE = ", test.MSE)
```

```
## new test MSE =  79%
```

```r
importance(bagging.carseats)
```

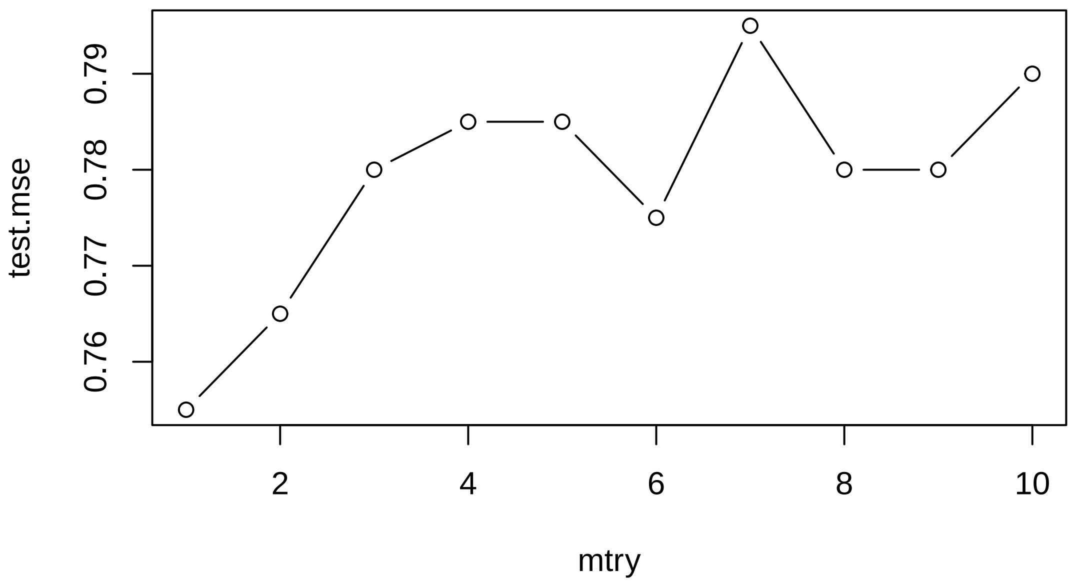
```
##                     No         Yes MeanDecreaseAccuracy MeanDecreaseGini
## CompPrice   10.5681973  7.87947174           12.6917798       12.1611634
## Income       6.0182883  7.63470778            9.2664333       10.1928409
## Advertising  7.2974265 11.53570424           12.5928115       12.3736809
## Population  -2.7465088 -0.86455164           -2.7518563        6.4837177
## Price       19.1386334 14.54935633           22.6591471       20.7786858
## ShelveLoc   19.4874521 20.45157633           25.7677759       15.2287296
## Age         10.4532541 16.21798039           16.5276994       13.1118793
## Education   -0.4267147 -0.08100658           -0.2158516        3.9834588
## Urban       -1.8527586 -0.67374000           -1.6924852        0.2964760
## US           1.8126646  4.27844875            3.8874342        0.6357877
```

Answer:

Yeah, bagging improve test MSE from 71% to 79%! According to the importance result, CompPrice, Income, Advertising, Price, ShelveLoc and Age are most important.

(e)

```
mtry <- 1:10
test.mse <- c()
for (i in mtry) {
  set.seed(i*1000)
  rf.carseats <- randomForest(x = train.X, y = train.Y,
                              ntree = ntrees, mtry = i)
  predict.Y <- predict(rf.carseats, newdata = test.X, type = "class")
  confusion.mtrx <- table(test.Y, predict.Y)
  test.mse <- c(test.mse,
                (confusion.mtrx[1, 1] + confusion.mtrx[2, 2])/nrow(test))}
plot(mtry, test.mse, type = "b")
```




```
# when m = 8, random forest model achieves the best performance
rf.carseats <- randomForest(x = train.X, y = train.Y,
                            ntree = ntrees, mtry = 7,
                            importance = TRUE)
importance(rf.carseats)
```

```
##                    No        Yes MeanDecreaseAccuracy MeanDecreaseGini
## CompPrice   10.668446  8.8064630            13.537164       11.9840074
## Income       3.192644  7.4616927             7.216788       10.5105526
## Advertising  7.079331 11.7124553            13.471091       14.1250723
## Population  -2.901628  0.3624930            -2.016323        6.4372050
## Price       17.296612 13.9424049            21.342960       19.1125307
## ShelveLoc   20.135766 18.4859219            24.696138       14.4813014
## Age         11.173733 13.4191396            16.025762       13.0767679
## Education   -1.776510 -0.8612653            -1.825459        4.1412868
## Urban       -2.015214 -1.2428878            -2.315994        0.4184205
## US           1.455324  3.6411723             3.790695        0.8649155
```

Answer:

the best performance appears when m = 8. the MSE of this random forest model is 79.5% and the most important predictors are CompPrice, Income, Advertising, Price, ShelveLoc and Age, which are exactly the same with bagging forest result.

8.11

(a)

```
rm(bagging.carseats, carseat.cv, carseat.tree, data, prune.carseats, rf.boston,
   rf.carseats, test, training, test.X, train.X)
rm(classification_error, confusion.mtrx, mtry, ntrees, predict.Y, test.mse,
   test.MSE, test.Y, train.Y)
library(ISLR)
```

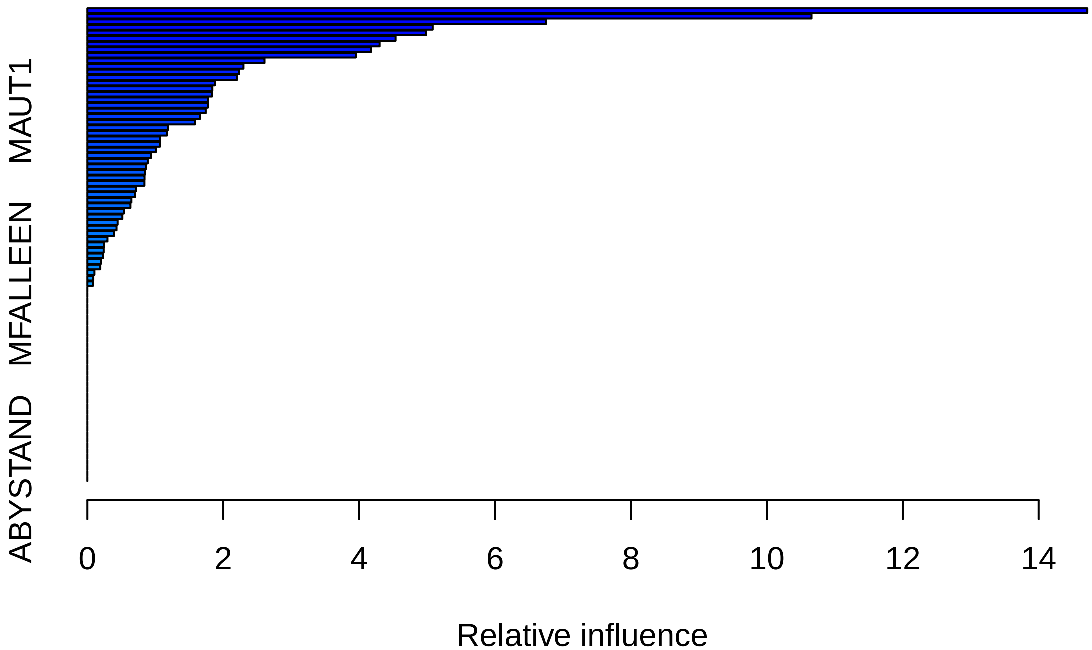
```
##
## Attaching package: 'ISLR'
```

```
## The following objects are masked from 'package:ISLR2':
##
##     Auto, Credit
```

```
train = 1:1000
data <- Caravan
data$Purchase = ifelse(Caravan$Purchase == "Yes", 1, 0)
train.set <- data[train, ]
test.set  <- data[-train, ]
```

(b)

```
Carvan.boost <- gbm(Purchase~., data = train.set,
                    n.trees = 1000, shrinkage = 0.01, distribution = "bernoulli")
summary(Carvan.boost)
```




```
##                 var     rel.inf
## PPERSAUT PPERSAUT 14.71622639
## MKOOPKLA MKOOPKLA 10.65747615
## MOPLHOOG MOPLHOOG  6.74893327
## MBERMIDD MBERMIDD  5.08204053
## PBRAND     PBRAND  4.98254974
## MGODGE     MGODGE  4.53611282
## ABRAND     ABRAND  4.30115820
## MINK3045 MINK3045  4.17425237
## MOSTYPE   MOSTYPE  3.95019248
## MAUT2       MAUT2  2.60721960
## MSKC         MSKC  2.29703960
## PWAPART   PWAPART  2.23146600
## MSKA         MSKA  2.20391617
## MGODPR     MGODPR  1.87700299
## PBYSTAND PBYSTAND  1.83935678
## MSKB1       MSKB1  1.83505128
## MBERARBG MBERARBG  1.77495272
## MGODOV     MGODOV  1.77328461
## MAUT1       MAUT1  1.74182641
## MINKGEM   MINKGEM  1.65999330
## MBERHOOG MBERHOOG  1.58688967
## MRELGE     MRELGE  1.18608323
## MRELOV     MRELOV  1.17179068
## MINK4575 MINK4575  1.06895257
## MGODRK     MGODRK  1.06852196
## MAUT0       MAUT0  1.00718923
## MINKM30   MINKM30  0.93835926
## MFWEKIND MFWEKIND  0.88852047
## MINK7512 MINK7512  0.86271742
## MFGEKIND MFGEKIND  0.84954635
## MHKOOP     MHKOOP  0.84161551
```

```
## MGEMLEEF MGEMLEEF  0.83998776
## MSKD         MSKD  0.71623092
## MZFONDS   MZFONDS  0.70475142
## MBERARBO MBERARBO  0.64792265
## MGEMOMV   MGEMOMV  0.63396590
## MHHUUR     MHHUUR  0.53650579
## MOPLMIDD MOPLMIDD  0.51493393
## APERSAUT APERSAUT  0.44525255
## PLEVEN     PLEVEN  0.42952061
## PMOTSCO   PMOTSCO  0.39292795
## MOSHOOFD MOSHOOFD  0.29607058
## MBERBOER MBERBOER  0.24795547
## MSKB2       MSKB2  0.24034366
## MOPLLAAG MOPLLAAG  0.23017910
## MZPART     MZPART  0.20205815
## MINK123M MINK123M  0.19049025
## MBERZELF MBERZELF  0.10400276
## MRELSA     MRELSA  0.08717811
## MFALLEEN MFALLEEN  0.07948471
## MAANTHUI MAANTHUI  0.00000000
## PWABEDR   PWABEDR  0.00000000
## PWALAND   PWALAND  0.00000000
## PBESAUT   PBESAUT  0.00000000
## PVRAAUT   PVRAAUT  0.00000000
## PAANHANG PAANHANG  0.00000000
## PTRACTOR PTRACTOR  0.00000000
## PWERKT     PWERKT  0.00000000
## PBROM       PBROM  0.00000000
## PPERSONG PPERSONG  0.00000000
## PGEZONG   PGEZONG  0.00000000
## PWAOREG   PWAOREG  0.00000000
## PZEILPL   PZEILPL  0.00000000
## PPLEZIER PPLEZIER  0.00000000
## PFIETS     PFIETS  0.00000000
## PINBOED   PINBOED  0.00000000
## AWAPART   AWAPART  0.00000000
## AWABEDR   AWABEDR  0.00000000
## AWALAND   AWALAND  0.00000000
## ABESAUT   ABESAUT  0.00000000
## AMOTSCO   AMOTSCO  0.00000000
## AVRAAUT   AVRAAUT  0.00000000
## AAANHANG AAANHANG  0.00000000
## ATRACTOR ATRACTOR  0.00000000
## AWERKT     AWERKT  0.00000000
## ABROM       ABROM  0.00000000
## ALEVEN     ALEVEN  0.00000000
## APERSONG APERSONG  0.00000000
## AGEZONG   AGEZONG  0.00000000
## AWAOREG   AWAOREG  0.00000000
## AZEILPL   AZEILPL  0.00000000
## APLEZIER APLEZIER  0.00000000
## AFIETS     AFIETS  0.00000000
## AINBOED   AINBOED  0.00000000
## ABYSTAND ABYSTAND  0.00000000
```

Answer:

PPERSAUT is the most important predictor.

(c)

```
prob <- predict(Carvan.boost, newdata = test.set, n.trees = 1000, type = "response")
predict.Y <- ifelse(prob > .2, 1, 0)
confusion.mtrx <- table(test.set$Purchase, predict.Y)
```

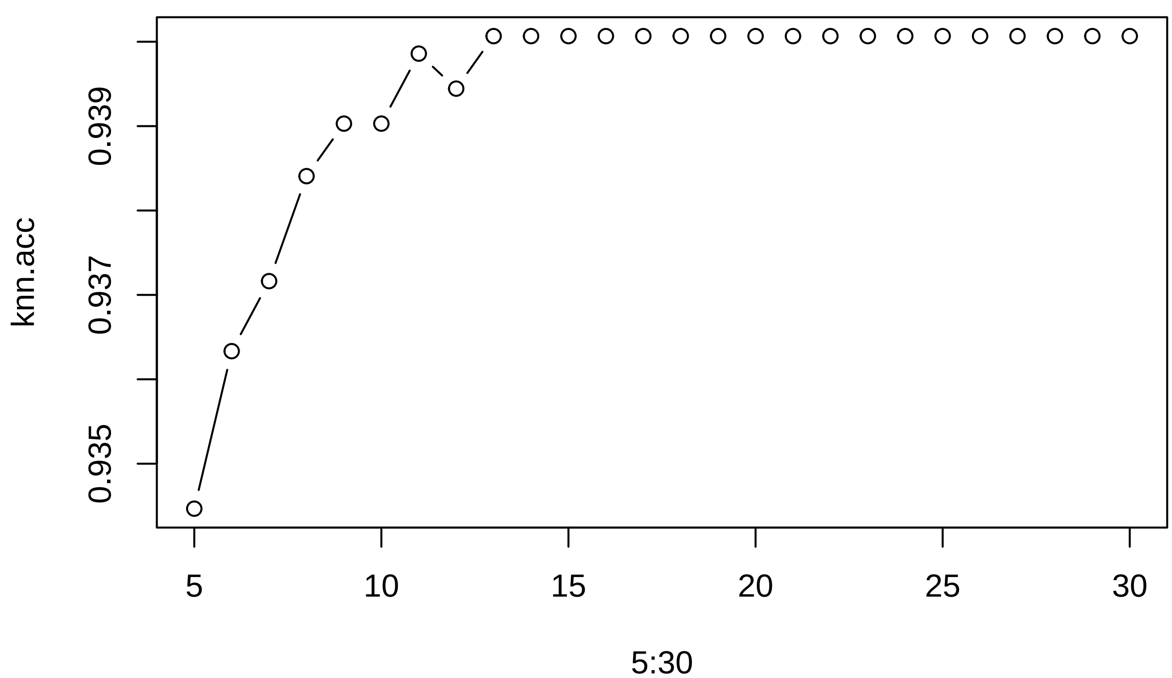
```
confusion.mtrx[2, 2]/(confusion.mtrx[1, 2] + confusion.mtrx[2, 2])
```

```
## [1] 0.2013423
```

Answer:

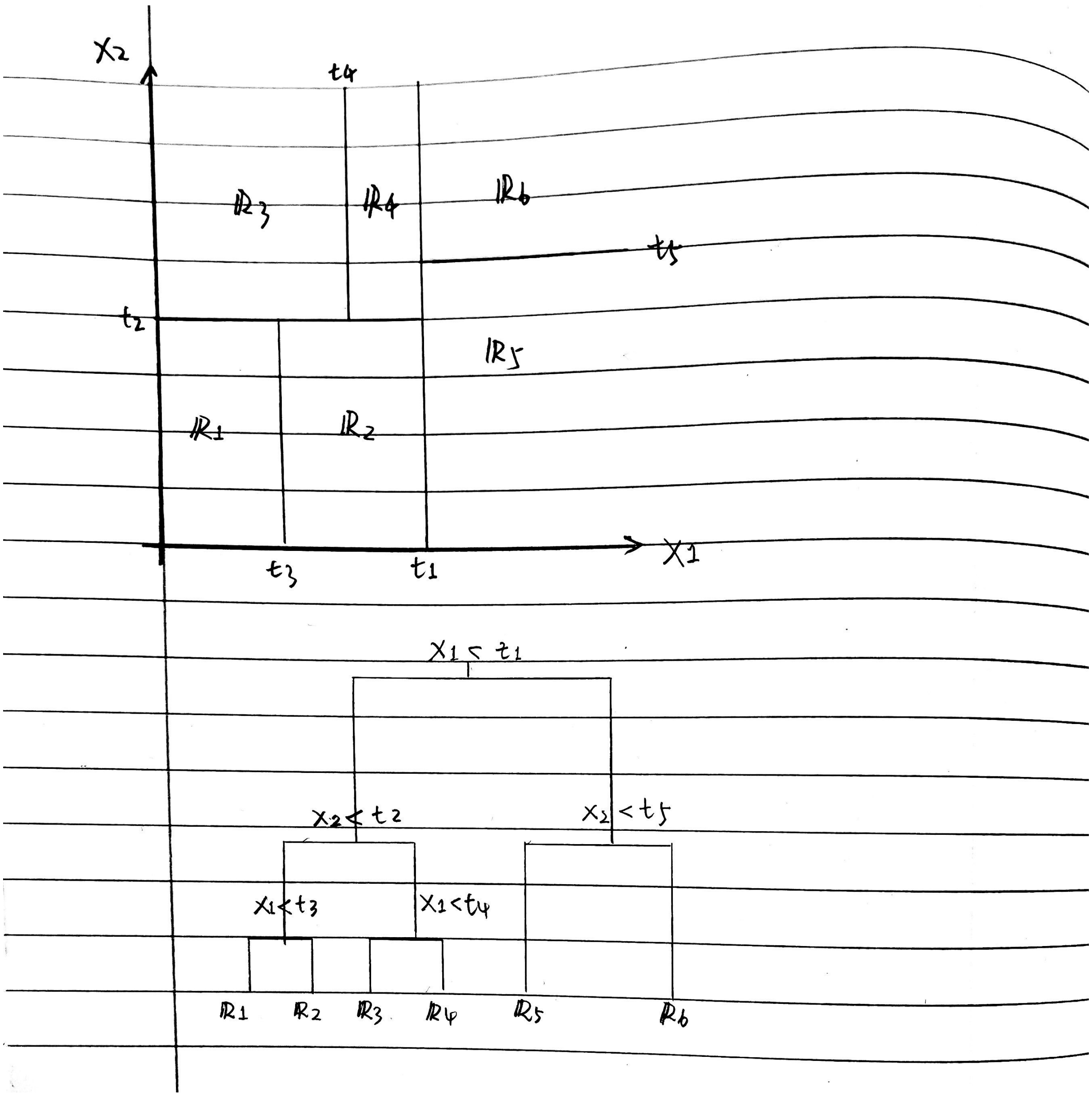
Around 20% of people predicted to make a purchase and actually made a purchase.

```
# which(colnames(train.set) == "Purchase")
train.X <- train.set[, -86]
train.Y <- train.set[,  86]
test.X <- test.set[, -86]
test.Y <- test.set[,  86]
knn.acc <- c()
for (i in 5:30) {
  Caravan.knn <- knn(train = train.X, test = test.X, cl = train.Y, k = i, prob = TRUE)
  confusion.mtrx <- table(test.Y,Caravan.knn)
  knn.acc <- c(knn.acc, (confusion.mtrx[1, 1] + confusion.mtrx[2, 2])/nrow(test.X))
}
plot(5:30, knn.acc, type = "b")
```



KNN predicts everybody to be no purchase!

Appendix



X2
t4
R3
R4
R6
t5
t2
R5
R1
R2
t3
t1
X1
X1 < t1
X2 < t2
X2 < t5
X1 < t3
X1 < t4
R1
R2
R3
R4
R5
R6