Assignment 7 – kNN & Trees

Math 154, Computational Statistics
Fall 2015, Maria Martinez

Due: Tuesday, November 3, 2015, noon

Not related to HW Score:

7:13 Total hours spent on assignment: 5

Number of different "sittings" to finish assignment: 4

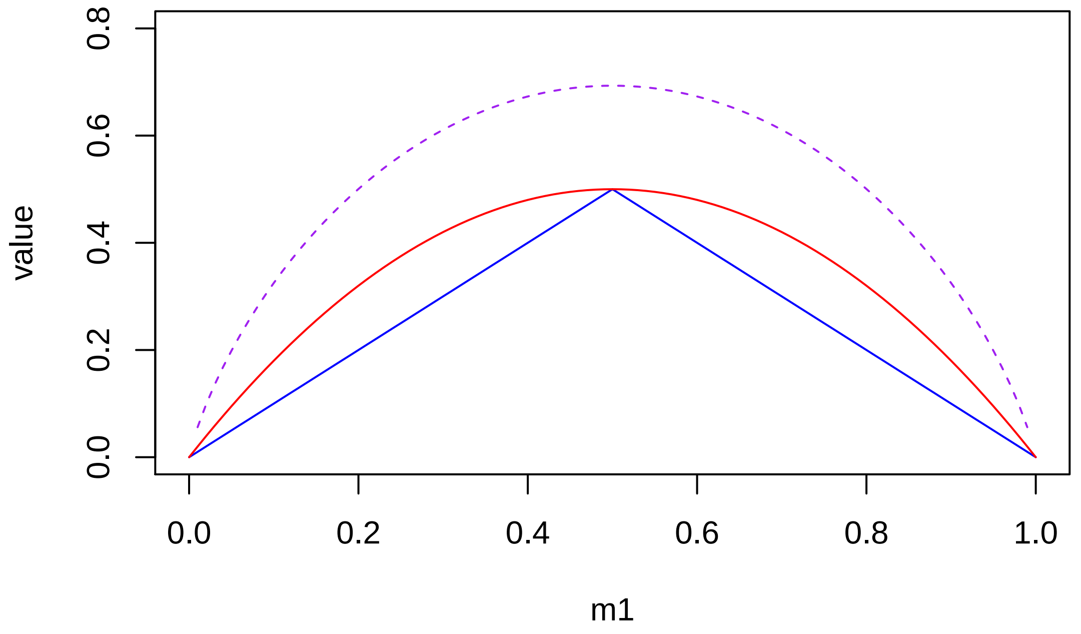
Assignment

1. Problem 7 in section 2.4 of *An Introduction to Statistical Learning.* Parts (a) and (b) only. The `dist` function in R gives Euclidean distances between rows of a matrix.
   (a) Compute the Euclidean distance between each observation and the test point, $X_1 = X_2 = X_3 = 0$. 0-0 0-3 0-0 = $sqrt9$ = 3, 2, 3.16, 2.24,1.414, 2.24

   (b) the point in class green, because it has the smallest distance.
2. 5.4.3) We now review k-fold cross-validation.
   (a) Explain how k-fold cross-validation is implemented.
   Take your dataset, and do a train/test split where your train data is split on $\frac{k-1}{k}$ and the test is what is left. You repeat this k times and then see how variable the test sets are.
   (b) What are the advantages and disadvantages of k-fold crossvalidation relative to:
      i. The validation set approach?
      The validation set approach does not let you use as much of your data for training as does k-fold. The variability is better seen through different data sets.
      ii. LOOCV?
      regular k-fold is faster than LOOCV.LOOCV also has higher variance.
3. Consider the Gini index, classification error, and cross-entropy in a simple classification setting with two classes. Create a single plot that displays each of these quantities as a function of $\hat{p}_(m1)$. The x-axis should display $\hat{p}_(m1)$, ranging from 0 to 1, and the y-axis should display the value of the Gini index, classification error, and entropy.

   The `lines` command in R will add a line to an existing plot.

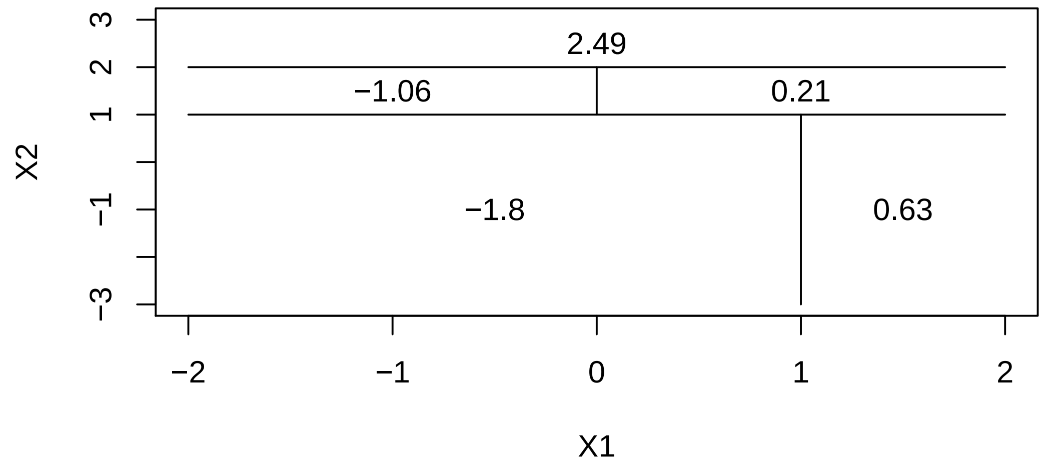
```r
gini = function(pm1){
  return (2*(pm1*(1-pm1)))
}

ent = function(pm1){
  pm2 = 1 - pm1
  return (-((pm1*log(pm1))+(pm2*log(pm2))))
}

classerr = function(pm1){
  return (1-max(pm1,1-pm1))
}

x = seq(0,1,by=0.01)
c.err = sapply(x,classerr)
g = sapply(x,gini)
e = sapply(x,ent)
d = data.frame(Gini.Index=g,Cross.Entropy=e)
plot(x,c.err,type='l',col="blue",xlab="m1",ylim=c(0,0.8),ylab="value")
matlines(x,d,col=c("red","purple"))
```



4. (a) Sketch the tree corresponding to the partition of the predictor space illustrated in the

left-hand panel of Figure 8.12. The numbers inside the boxes indicate the mean of Y within each region.
If $X_1 \geq 1$ then 5, else if $X_2 \geq 1$ then 15, else if $X_1 < 0$ then 3, else if $X_2 < 0$ then 10, else 0.

(b) Create a diagram similar to the left-hand panel of Figure 8.12, using the tree illustrated in the right-hand panel of the same figure. You should divide up the predictor space into the correct regions, and indicate the mean for each region.



5. Problem 9 in section 8.4 of *An Introduction to Statistical Learning*

(a) Create a training set containing a random sample of 800 observations, and a test set containing the remaining observations.

```
set.seed(4)
require(ISLR)
data(OJ)

train    = sample(1:nrow(OJ), 800, replace = F)
OJTrain = OJ[train, ]
OJTest  = OJ[-train, ]
```

(b) Fit a tree to the training data, with Purchase as the response and the other variables except for Buy as predictors. Use the summary() function to produce summary statistics about the tree, and describe the results obtained. What is the training error rate? How many terminal nodes does the tree have?

```
require(class)
require(tree)
```

```
  OJTree = tree(Purchase ~ ., data = OJTrain)
  summary(OJTree)
```

```
##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJTrain)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "SalePriceMM" "PriceDiff"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7677 = 608.8 / 793
## Misclassification error rate: 0.1788 = 143 / 800
```

The fitted tree has 7 terminal nodes and a training error rate of 0.1788.

(c) Type in the name of the tree object in order to get a detailed text output. Pick one of the terminal nodes, and interpret the information displayed.

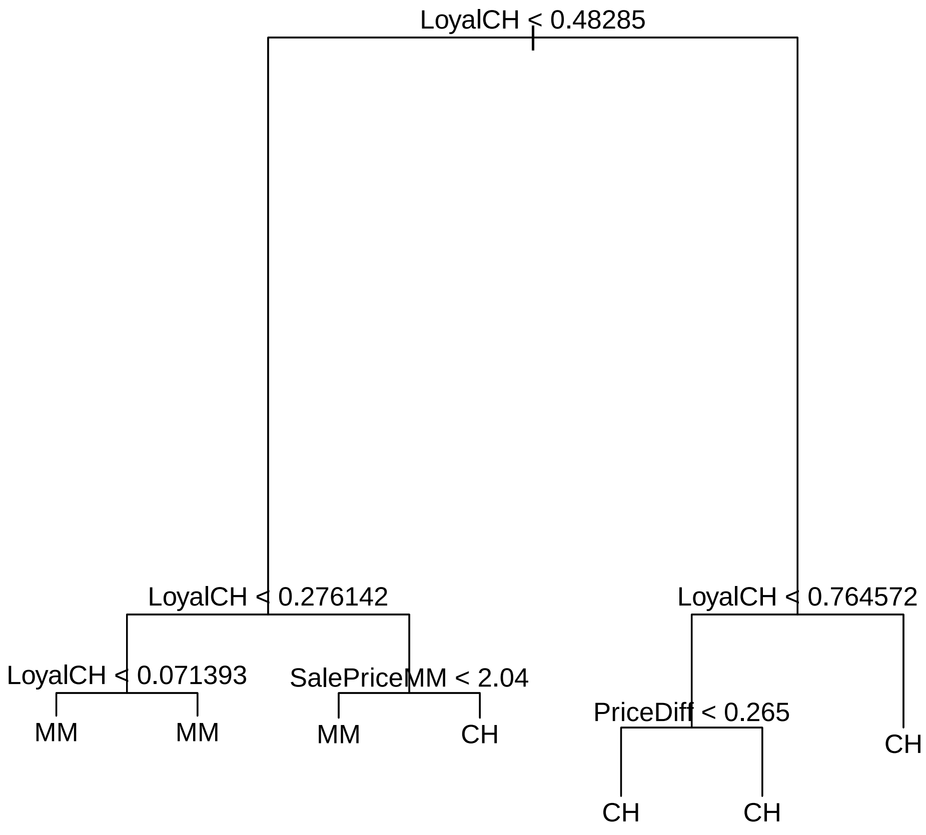
```
  OJTree
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
##  1) root 800 1073.00 CH ( 0.60625 0.39375 )
##    2) LoyalCH < 0.48285 302  324.60 MM ( 0.22848 0.77152 )
##      4) LoyalCH < 0.276142 166  100.70 MM ( 0.09036 0.90964 )
##        8) LoyalCH < 0.071393 75   10.62 MM ( 0.01333 0.98667 ) *
##        9) LoyalCH > 0.071393 91   78.14 MM ( 0.15385 0.84615 ) *
##      5) LoyalCH > 0.276142 136  182.70 MM ( 0.39706 0.60294 )
##       10) SalePriceMM < 2.04 76   87.60 MM ( 0.26316 0.73684 ) *
##       11) SalePriceMM > 2.04 60   82.11 CH ( 0.56667 0.43333 ) *
##    3) LoyalCH > 0.48285 498  445.50 CH ( 0.83534 0.16466 )
##      6) LoyalCH < 0.764572 232  282.40 CH ( 0.70259 0.29741 )
##       12) PriceDiff < 0.265 132  181.50 CH ( 0.55303 0.44697 ) *
##       13) PriceDiff > 0.265 100   65.02 CH ( 0.90000 0.10000 ) *
##      7) LoyalCH > 0.764572 266  103.80 CH ( 0.95113 0.04887 ) *
```

I picked the terminal node 7. The split criterion is LoyalCH < 0.764572, the number of observations is 266, deviance is 103.8 and an overall prediction for the branch of CH. A little more than 4 percent of the observations in node 7 take the value of MM, while around 95 percent take CH.

(d) Create a plot of the tree, and interpret the results.

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



The most important factor is LoyalCH, since the top three nodes are LoyalCH.

(e) Predict the response on the test data, and produce a confusion matrix comparing the test labels to the predicted test labels. What is the test error rate?

```
  tree.pred <- predict(OJTree, OJTest, type = "class")
  table(tree.pred, OJTest$Purchase)

##
## tree.pred  CH  MM
##        CH 153  38
##        MM  15  64

  1 - (153 + 64) / (153 + 15 + 38 + 64)

## [1] 0.1962963
```

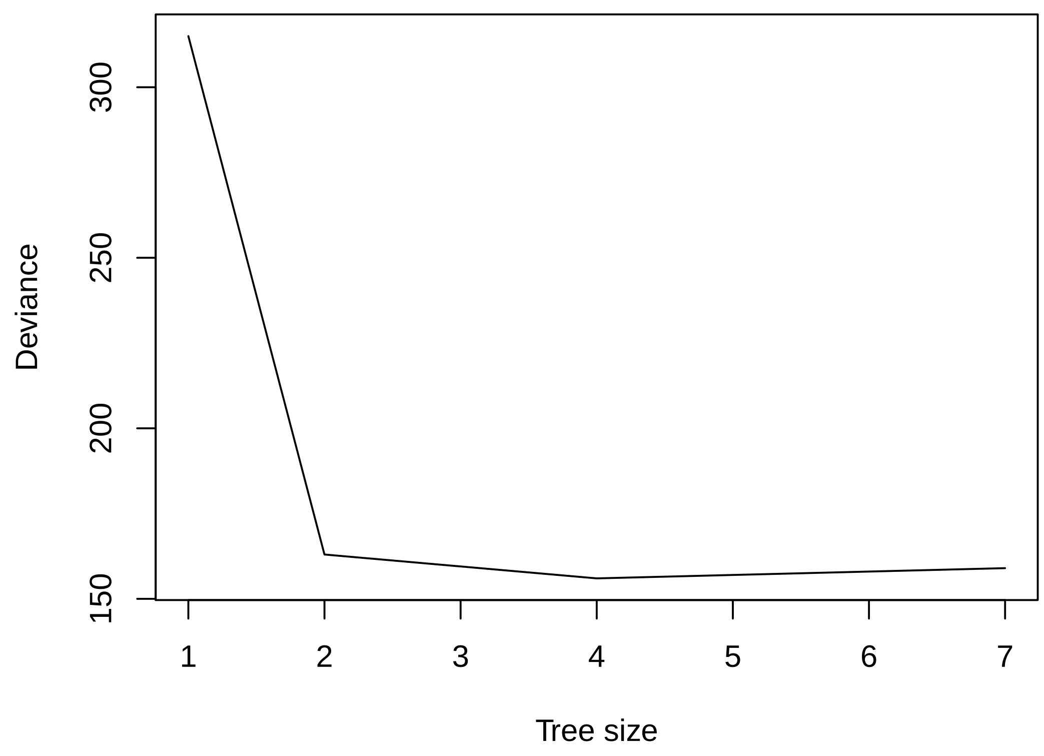
The test error rate is thus around 1.96 percent.

(f) Apply the cv.tree() function to the training set in order to determine the optimal tree size.

```
  cv.oj <- cv.tree(OJTree, FUN = prune.misclass); cv.oj

## $size
## [1] 7 4 2 1
##
## $dev
## [1] 159 156 163 315
##
## $k
## [1] -Inf    0    4  164
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

(g) Produce a plot with tree size on the x-axis and cross-validated classification error rate on the y-axis.

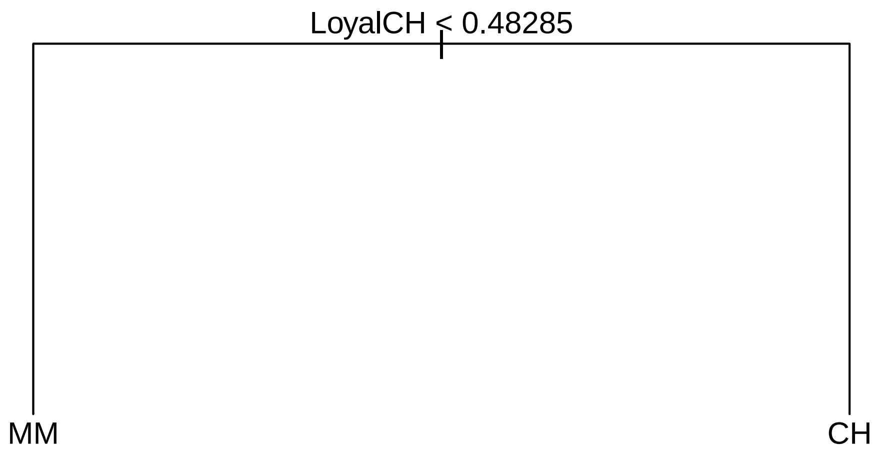
```
  plot(cv.oj$size, cv.oj$dev, type = "l",
       xlab = "Tree size", ylab = "Deviance")
```



(h) Which tree size corresponds to the lowest cross-validated classification error rate?
4 - node is the smallest tree with the smallest classification error rate.

(i) Produce a pruned tree corresponding to the optimal tree size obtained using cross-validation. If cross-validation does not lead to selection of a pruned tree, then create a pruned tree with five terminal nodes.

```r
prune.oj <- prune.misclass(OJTree, best = 2)
plot(prune.oj)
text(prune.oj, pretty = 0)
```



(j) Compare the training error rates between the pruned and unpruned trees. Which is higher?

```
  summary(OJTree)

##
## Classification tree:
## tree(formula = Purchase ~ ., data = OJTrain)
## Variables actually used in tree construction:
## [1] "LoyalCH"     "SalePriceMM" "PriceDiff"
## Number of terminal nodes:  7
## Residual mean deviance:  0.7677 = 608.8 / 793
## Misclassification error rate: 0.1788 = 143 / 800

  summary(prune.oj)

##
## Classification tree:
## snip.tree(tree = OJTree, nodes = c(3L, 2L))
## Variables actually used in tree construction:
## [1] "LoyalCH"
## Number of terminal nodes:  2
## Residual mean deviance:  0.9651 = 770.1 / 798
## Misclassification error rate: 0.1888 = 151 / 800
```

The misclassifcation error rate for the tree is 0.1788 and the prune tree is 0.1888. The pruned tree has a higher error rate.

(k) Compare the test error rates between the pruned and unpruned trees. Which is higher?

```
  prune.pred <- predict(prune.oj, OJTest, type = "class")
  table(prune.pred, OJTest$Purchase)

##
## prune.pred  CH  MM
##         CH 143  28
##         MM  25  74

  1 - (143 + 74) / (143 + 25 + 28 + 74)

## [1] 0.1962963
```

They were exactly the same.

6. Using the same test and training data from the previous problem, use *k*NN to classify the OJ data

   (a) Use k=1

```
  set.seed(5)
  train = OJTrain[,-c(1,14)]
  test  = OJTest[,-c(1,14)]
  k1.pred <- knn(train,test, cl = OJTrain$Purchase, k = 1)
  table(k1.pred, OJTest$Purchase)


##
## k1.pred  CH  MM
##      CH 124  41
##      MM  44  61


  accuracyK1 = (119 + 70) / (119 + 70 + 40 + 41)
```

   (b) Use k=5

```
  k5.pred <- knn(train,test, cl = OJTrain$Purchase, k = 5)
  table(k5.pred, OJTest$Purchase)


##
## k5.pred  CH  MM
##      CH 127  40
##      MM  41  62


  accuracyK5 = (127 + 64) / (127 + 64 + 47 + 32)
```

(c) Use k=11 ***

```
  k11.pred <- knn(train,test, cl = OJTrain$Purchase, k = 11)
  table(k11.pred, OJTest$Purchase)


##
## k11.pred  CH  MM
##       CH 133  44
##       MM  35  58

  accuracyK11 = (132 + 58) / (132 + 58 + 53 + 27)
```

(d) Which value of k gave the most accurate predictions for the Purchase variable?

```
  unlist(list(One = accuracyK1, Five = accuracyK5, Eleven = accuracyK11))


##       One      Five    Eleven
## 0.7000000 0.7074074 0.7037037
```

K = 5 gives us the most accurate prediction.

(e) In parts (a)-(c) you required to adjust the explanatory variables in a way that was not required for the CART classification. What was the adjustment and why was it necessary?

We had to get rid of the categorical variables because kNN does not take them into consideration as oppose to CART which knows how to deal with them.