assignment3_sol

chandan u

2/25/2017

Question 1

given:
Let ith category of x be x_i
Let jth category of y be y_j
The column width be written as width(x_i) or width(y_j)
The probabilities is wirrten as p(x_i) or p(y_i)
Each cell can be represented as (x_i, y_j)
Area of a cell is area((x_i, y_j))

```
1.) The column widths are proportional to the marginal counts on total outcomes of first variable:
     width($x_i$) $\propto p(x_i)$

2.) The cell areas are proportional to the number of counts for each cell:

     area($(x_i, y_j)$) $\propto p(x_i, y_j)$
```

a

P(Y = y|X = x) = $\frac{p(Y=y \Lambda X=x)}{p(X=x)}$
$=\frac{p(Y=y \Lambda X=x)}{p(X=x)}$

from 2 we know that $p(Y = y \Lambda X = x$ is $\propto$ area((x, y))
from 1 we know that p(X=x) $\propto$ width(X=x)

So substituting these in the above equation we get:

P(Y = y|X = x) = $\frac{area(x,y)}{width(x)}$

so area(x,y) = width(x) * height(y),

P(Y = y|X = x) = $\frac{width(x)*height(y)}{width(x)}$

so from the given equation we can see that :

P(Y = y|X = x) $\propto$ height(x)

"hence proved"

b

For independence of X and Y: P(Y = y|X = x) = $\frac{p(Y=y \Lambda X=x)}{p(X=x)}$
P(Y = y|X = x) = $\frac{p(Y=y)*p(X=x)}{p(X=x)}$
P(Y = y|X = x) = $p(Y = y)$

So the value is proportional to the height of the plot.

So in an independed situation all hieghts (y) of each categoy across all the columns (x) is same.

So Lets devide the columns of mosaic plot based on count of x. The widhts of x can be different. We will have columns with widths based on the probabilites of categories in x. Similarly now we devide each column with probabilities. Since we have probabilities in the meanset So all the cells in the same row have same height because y is independent of x and it carries same values not matter what x is.

c

X,Y,Z are binary variables.

When Z is unknow:

$P(X^yZ) = P(X) * P(Y)$

When Z is known:

P(X ˆ Y ˆ Z) = P(X|Y ˆ Z) * P(Y|Z) * P(Z)

Question 2

load data

a

The code for rpart:

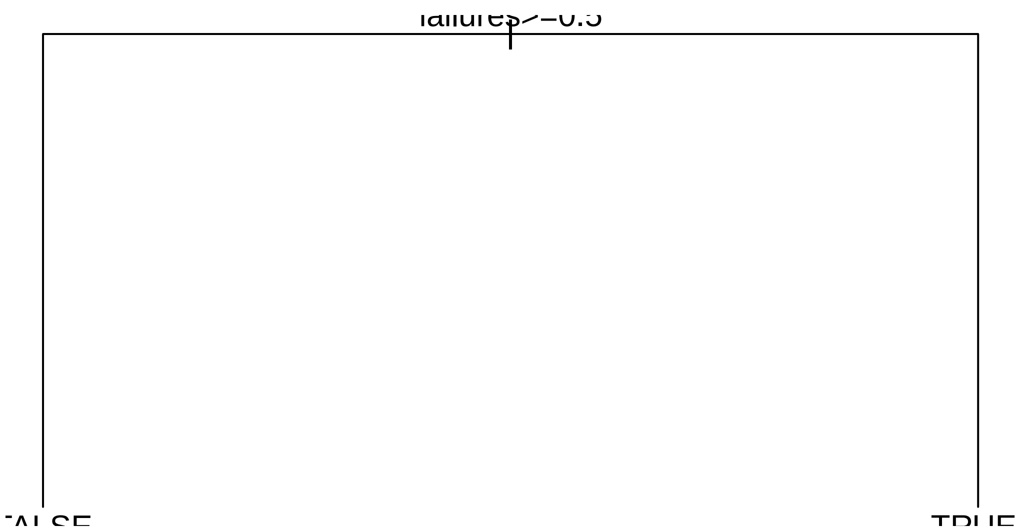
```
student_data = read.csv("./student/student-mat.csv", sep=";")
# rpart library: load
library(rpart)

# target class variable
c = student_data$G3 > 10


# grow the tree
fit = rpart(c ~ school+sex+age+address+famsize+Pstatus+Medu+Fedu+Mjob+Fjob+reason+guardian+traveltime+st

#printcp(fit)
# prune the tree: remove variables that do not show statistically significant improvement
fit <- prune(fit, cp=fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])


plot(fit)       # look at complex tree we built
text(fit)
```



Generalization error and important feature and splits:

The generalization error is 0.80645 * 0.47089 = 0.3797492 Failuers is the important feature. It uses six splits.

```r
printcp(fit)
```

```
##
## Classification tree:
## rpart(formula = c ~ school + sex + age + address + famsize +
##     Pstatus + Medu + Fedu + Mjob + Fjob + reason + guardian +
##     traveltime + studytime + failures + schoolsup + famsup +
##     paid + activities + nursery + higher + internet + romantic +
##     famrel + freetime + goout + Dalc + Walc + health + absences,
##     data = student_data, method = "class")
##
## Variables actually used in tree construction:
## [1] failures
##
## Root node error: 186/395 = 0.47089
##
## n= 395
##
##         CP nsplit rel error xerror     xstd
## 1 0.231183      0   1.00000 1.0000 0.053336
## 2 0.032258      1   0.76882 0.7957 0.051721
```

```r
fit$variable.importance
```

```
##  failures       age  guardian    higher
## 17.449703  3.574036  2.102374  1.261424
```

b

```r
# target class variable


# grow the tree
```

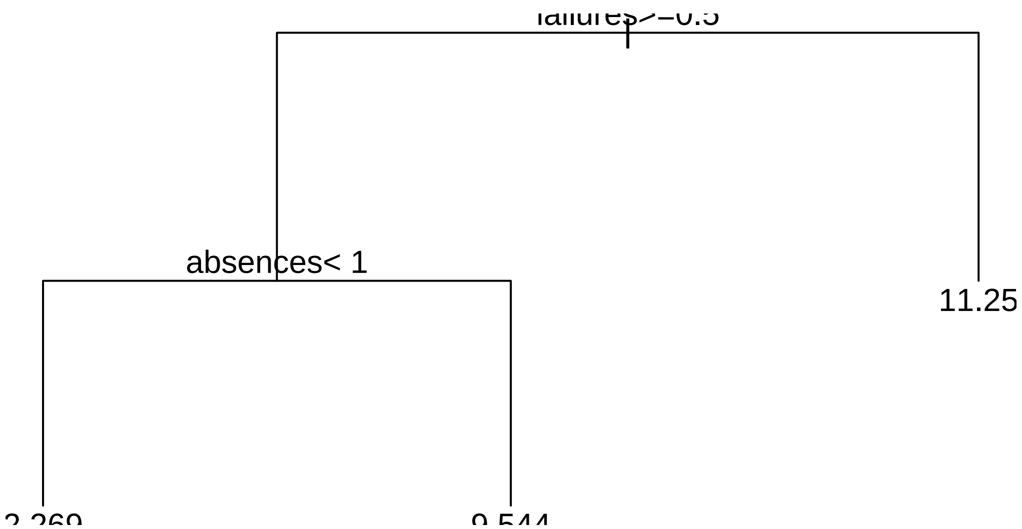
```r
fit = rpart(G3 ~ school+sex+age+address+famsize+Pstatus+Medu+Fedu+Mjob+Fjob+reason+guardian+traveltime+s

printcp(fit)
```

```
##
## Regression tree:
## rpart(formula = G3 ~ school + sex + age + address + famsize +
##     Pstatus + Medu + Fedu + Mjob + Fjob + reason + guardian +
##     traveltime + studytime + failures + schoolsup + famsup +
##     paid + activities + nursery + higher + internet + romantic +
##     famrel + freetime + goout + Dalc + Walc + health + absences,
##     data = student_data, method = "anova")
##
## Variables actually used in tree construction:
## [1] absences failures Fjob     freetime health   Mjob     reason   sex
##
## Root node error: 8269.9/395 = 20.936
##
## n= 395
##
##         CP nsplit rel error  xerror     xstd
## 1 0.126089      0   1.00000 1.00690 0.078250
## 2 0.114259      1   0.87391 0.93661 0.075767
## 3 0.023363      2   0.75965 0.76818 0.068630
## 4 0.014540      3   0.73629 0.86327 0.077812
## 5 0.012885     10   0.63404 0.91160 0.081034
## 6 0.010000     13   0.59538 0.92202 0.081625
```

```r
# prune the tree: remove variables that do not show statistically significant improvement
fit <- prune(fit, cp=fit$cptable[which.min(fit$cptable[,"xerror"]),"CP"])


plot(fit)                              # look at complex tree we built
text(fit)
```




```r
post(fit, file="2_b.ps")
```

Generalization error and important feature and splits:

The generalization error is 20.936 * 0.78021 = 16.33448
failures is the important variable.
It uses just two splits.

```r
printcp(fit)
```

```
##
## Regression tree:
## rpart(formula = G3 ~ school + sex + age + address + famsize +
##     Pstatus + Medu + Fedu + Mjob + Fjob + reason + guardian +
##     traveltime + studytime + failures + schoolsup + famsup +
##     paid + activities + nursery + higher + internet + romantic +
##     famrel + freetime + goout + Dalc + Walc + health + absences,
##     data = student_data, method = "anova")
##
## Variables actually used in tree construction:
## [1] absences failures
##
## Root node error: 8269.9/395 = 20.936
##
## n= 395
##
##         CP nsplit rel error  xerror     xstd
## 1 0.126089      0   1.00000 1.00690 0.078250
## 2 0.114259      1   0.87391 0.93661 0.075767
## 3 0.023363      2   0.75965 0.76818 0.068630
```

```r
fit$variable.importance
```

```
##   failures   absences        age   guardian     higher traveltime
## 1042.74339  944.91294  213.57395  125.63173   75.37904   72.68561
##      goout
##   36.34281
```

Question 3

a

In the diagram we see that x=y line divides the boundary. Our new feature column values will be f(x,y)=y/x . if y/x >= 0 then they belong to circle else triangle.

b

In the diagram we can see the boundary region between the two classes is ellipse/ special case of ellipse circle. So the new feature will be f(x,y) = $\sqrt{x^2 + y^2}$. Equation of a circle or distance of point from center of circle (0,0). All the values less than the radius will be circle class else will be triangle class.

Question 4

a

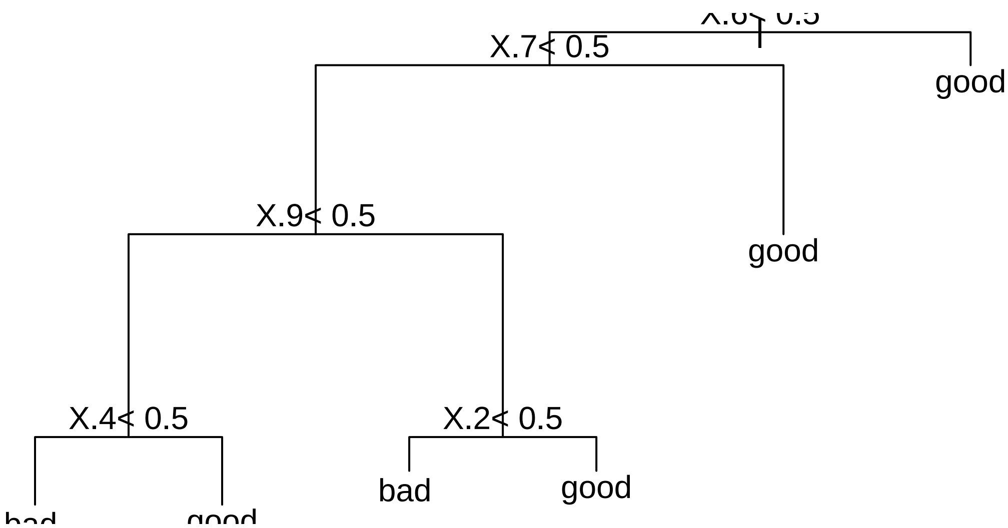
The training error rate:

```r
strange_binary = read.csv("strange_binary.csv")
fit <- rpart(strange_binary[,"c"] ~ X + X.1 + X.2 + X.3 + X.4 + X.5 + X.6 + X.7 + X.8 + X.9, data=strang

printcp(fit)
```

```
##
## Classification tree:
## rpart(formula = strange_binary[, "c"] ~ X + X.1 + X.2 + X.3 +
##     X.4 + X.5 + X.6 + X.7 + X.8 + X.9, data = strange_binary,
##     method = "class")
##
## Variables actually used in tree construction:
## [1] X.2 X.4 X.6 X.7 X.9
##
## Root node error: 64/200 = 0.32
##
## n= 200
##
##         CP nsplit rel error xerror    xstd
## 1 0.057292      0   1.00000 1.0000 0.10308
## 2 0.031250      3   0.82812 1.1094 0.10574
## 3 0.015625      4   0.79688 1.1719 0.10698
## 4 0.010000      5   0.78125 1.1250 0.10607
```

```r
plot(fit)
text(fit)
```




```r
post(fit,file="4_a.ps")
```

As you can see the xerror of 1.1250 has number of splits as three. For not more than three splits we must choose the model to have a cp of 0.031250. Hence the training error of this is:

training_error = relative_error * root_node_error
(root node error is scaled to one in the relative error. To get actual training error we must multiply with the root node error)

training_error_rate = 0.82812 * 0.32 = 0.2649984

We are getting 0.2649984 percent error on training data. But it is not reasonable to assume that the results on the testing data would be similar. As testing data is unseen data and it may have samples that are not present in training data. Also if the decission tree is too complex, it will absorb the training data (overfitting) but in such cases it usually does not generalize well on the testing data. It all depends on the nature of testing data. If you certain that your testing data will be always similar to the training data then the model might more or less perform the same on the testing data. But if it is not then a less complex model may perform well, then the one that overfits, but there is no guarentee that it performs the same with testing data as it perfroms on the training data.

b

From the summary we can see that all the values in the dataset are either 0 or 1.

Hence Lets try a new feature that takes the summation of all the features: Also number of zeros and number of ones in a row: A xor of the above features

So the accuracy is :1 - (0.70312 * 0.32) = 0.7750016

```r
summary(strange_binary)
```

```
##        X              X.1            X.2             X.3       
##  Min.   :0.000   Min.   :0.00   Min.   :0.00   Min.   :0.000  
##  1st Qu.:0.000   1st Qu.:0.00   1st Qu.:0.00   1st Qu.:0.000  
##  Median :1.000   Median :1.00   Median :1.00   Median :1.000  
##  Mean   :0.525   Mean   :0.52   Mean   :0.51   Mean   :0.505  
##  3rd Qu.:1.000   3rd Qu.:1.00   3rd Qu.:1.00   3rd Qu.:1.000  
##  Max.   :1.000   Max.   :1.00   Max.   :1.00   Max.   :1.000  
##       X.4            X.5             X.6              X.7       
##  Min.   :0.00   Min.   :0.000   Min.   :0.000   Min.   :0.000  
##  1st Qu.:0.00   1st Qu.:0.000   1st Qu.:0.000   1st Qu.:0.000  
##  Median :0.00   Median :1.000   Median :0.000   Median :0.000  
##  Mean   :0.44   Mean   :0.525   Mean   :0.485   Mean   :0.435  
##  3rd Qu.:1.00   3rd Qu.:1.000   3rd Qu.:1.000   3rd Qu.:1.000  
##  Max.   :1.00   Max.   :1.000   Max.   :1.000   Max.   :1.000  
##       X.8             X.9           c      
##  Min.   :0.000   Min.   :0.000   bad : 64  
##  1st Qu.:0.000   1st Qu.:0.000   good:136  
##  Median :1.000   Median :0.000             
##  Mean   :0.515   Mean   :0.485             
##  3rd Qu.:1.000   3rd Qu.:1.000             
##  Max.   :1.000   Max.   :1.000             
```

```r
newFeature1 = rowSums(strange_binary[,c(1,2,3,4,6,9)])
newFeature2 = rowSums(strange_binary[,c(5,7,8,10)])
newFeature = xor(newFeature1, newFeature2)
newFeature3 = rowSums(strange_binary[,1:10]==0)
newFeature4 = rowSums(strange_binary[,1:10]==1)
newFeature_2 = xor(newFeature3, newFeature4)


fit <- rpart(strange_binary[,"c"] ~ newFeature1+newFeature2+newFeature3+newFeature4+X +newFeature_2+ X.

printcp(fit)
```

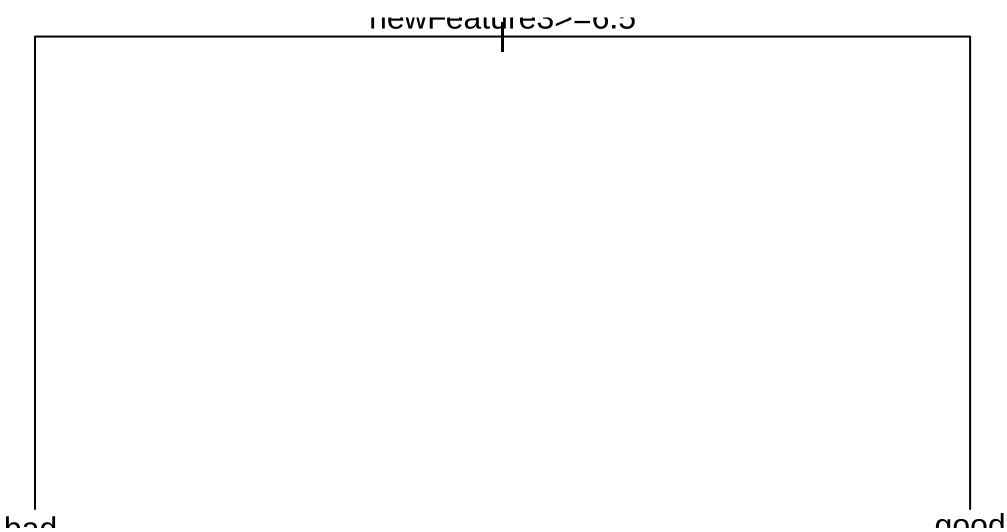
```
## 
## Classification tree:
## rpart(formula = strange_binary[, "c"] ~ newFeature1 + newFeature2 + 
##     newFeature3 + newFeature4 + X + newFeature_2 + X.1 + X.2 + 
```

```
##     X.3 + X.4 + X.5 + X.6 + X.7 + X.8 + X.9, data = strange_binary,
##     method = "class")
##
## Variables actually used in tree construction:
## [1] newFeature3
##
## Root node error: 64/200 = 0.32
##
## n= 200
##
##        CP nsplit rel error  xerror     xstd
## 1 0.29688      0   1.00000 1.00000 0.103078
## 2 0.01000      1   0.70312 0.70312 0.092274
```

```
fit$variable.importance
```

```
## newFeature3 newFeature4 newFeature1 newFeature2
##   17.385596   17.385596    6.686768    4.012061
```

```
plot(fit)
text(fit)
```



```
post(fit,file="4_b_plot.ps")
```

Question 5

a

From the below output we can see that the minimal value for m is 4 and depth of the tree is 9 (nsplit) for the near perfect accuracy.

```
n = 1000;
x = rep(0,n);
s = c(1,5,4,2,3);
k = length(s);
for (i in (k+1):n) {
  j = s[(i %% k) + 1]; # i %% k is i mod k
  x[i] = 1 - x[i-j];
```

```
}

m = 4
x_i_1 = c(tail(x, -1), head(x, 1))
x_i_2 = c(tail(x, -2), head(x, 2))
x_i_3 = c(tail(x, -3), head(x, 3))
x_i_4 = c(tail(x, -4), head(x, 4))

df = data.frame(x_i_1, x_i_2, x_i_3, x_i_4)

fit <- rpart(x ~ x_i_1+x_i_2+x_i_3+x_i_4, data=df,method = "class")

printcp(fit)
```

```
##
## Classification tree:
## rpart(formula = x ~ x_i_1 + x_i_2 + x_i_3 + x_i_4, data = df,
##     method = "class")
##
## Variables actually used in tree construction:
## [1] x_i_1 x_i_2 x_i_3 x_i_4
##
## Root node error: 498/1000 = 0.498
##
## n= 1000
##
##          CP nsplit rel error   xerror      xstd
## 1 0.196787      0 1.0000000 1.088353 0.0316376
## 2 0.100402      1 0.8032129 0.887550 0.0315354
## 3 0.099398      3 0.6024096 0.753012 0.0307416
## 4 0.098896      5 0.4036145 0.618474 0.0293156
## 5 0.010000      9 0.0080321 0.018072 0.0059969
```

Question 6

a

given:
E(log(x)) $\leq log(E(x))$

To prove:
$q_l H(p_l) + q_r H(p_r) \leq H(p)$

proof:
Entropy function is: $H(p) = -\Sigma_{i=1}^{n} p_i log(p_i)$

So taking the LHS of the statement to be proved and substituting the entropy equations:

$q_l H(p_l) + q_r H(p_r) = q_l * (-\Sigma_{i=1}^{n} p_{li} log(p_{li})) + q_r * (-\Sigma_{i=1}^{n} p_{li} log(p_{li}))$

$= -(q_l * \Sigma_{i=1}^{n} p_{li} log(p_{li})) - (q_r * \Sigma_{i=1}^{n} p_{li} log(p_{li}))$

The expected value for a distribution x is $\Sigma_{i=1}^{n} xp(x)$, using that:

$= - (q_l * E(log(p_l)) - (q_r * E(log(p_r))) $

Now from the given we know that E(log(x)) $\leq long(E(x))$:

$-(q_l * E(log(p_l)) - (q_r * E(log(p_r))) <= -(q_l * log(E(p_l)) - (q_r * log(E(p_r)))$

Question 7

a

```r
accuracies = read.csv("4_9.csv")

classifiers = c( "decision_tree", "naive_bayes", "svm")
print(c("classifiers", classifiers))
```

```
## [1] "classifiers"   "decision_tree" "naive_bayes"   "svm"
```

```r
for (i in c(1:2)){
  for (j in c(i+1:3)){
    classifier_1 =  classifiers[i]
    classifier_2 =  classifiers[j]
    if (is.na(classifier_2)){
      break
    }

    wins = 0
    loss = 0
    draws = 0
    for (dataset in c(1:nrow(accuracies))){
      if ( accuracies[dataset, classifier_1] > accuracies[dataset, classifier_2]){
        wins = wins + 1;
      }
      if ( accuracies[dataset, classifier_1] == accuracies[dataset, classifier_2]){
        draws = draws + 1;
      }
      if (  accuracies[dataset, classifier_1] < accuracies[dataset, classifier_2]){
        loss = loss + 1;
      }
    }# for
    print(c(classifier_1, classifier_2, wins, loss, draws))

  }# for

}# for
```

```
## [1] "decision_tree" "naive_bayes"   "11"            "11"
## [5] "1"
## [1] "decision_tree" "svm"           "8"             "15"
## [5] "0"
## [1] "naive_bayes" "svm"         "3"           "20"          "0"
```

So the produced table is :

1] “classifiers” “decision_tree” “naive_bayes” “svm”
[1] “decision_tree” 0-0-23 11-11-1 8-15-0
[1] “naive_bayes” 11-11-1 0-0-23 3-30-0
[1] “svm” 8-15-0 3-30-0 0-0-23