Data and the plug-in principle

{r setup, include=FALSE} knitr::opts_chunk$set(cache=TRUE)

1

```
return_probs <- function(row){
row <- as.numeric(row)

product_1 <- p_v1_v11[,1][row[1]+1]* p_v2_v11[,1][row[2]+1]* p_v3_v11[,1][row[3]+1]* p_v4_v1
product_2 <- p_v1_v11[,2][row[1]+1]* p_v2_v11[,2][row[2]+1]* p_v3_v11[,2][row[3]+1]* p_v4_v1
product_3 <- p_v1_v11[,3][row[1]+1]* p_v2_v11[,3][row[2]+1]* p_v3_v11[,3][row[3]+1]* p_v4_v1

c(P_1*product_1, P_2*product_2, P_3*product_3)

}

binary_data <- read.csv('naive_bayes_binary.csv')
binary_data_train <- binary_data[1:(length(binary_data$V1)/2),]
binary_data_test <- binary_data[2501:5000,]
true_values <- binary_data_test$V11
binary_data_test <- binary_data_test[1:10]


#create tables of all variables and derive their conditional probs

table_v1_v11 <- table(binary_data_train$V1, binary_data_train$V11)
table_v2_v11 <- table(binary_data_train$V2, binary_data_train$V11)
table_v3_v11 <- table(binary_data_train$V3, binary_data_train$V11)
table_v4_v11 <- table(binary_data_train$V4, binary_data_train$V11)
table_v5_v11 <- table(binary_data_train$V5, binary_data_train$V11)
table_v6_v11 <- table(binary_data_train$V6, binary_data_train$V11)
table_v7_v11 <- table(binary_data_train$V7, binary_data_train$V11)
table_v8_v11 <- table(binary_data_train$V8, binary_data_train$V11)
table_v9_v11 <- table(binary_data_train$V9, binary_data_train$V11)
table_v10_v11 <- table(binary_data_train$V10, binary_data_train$V11)
```

```
# table_vecs <-

p_v1_v11 <- prop.table(table_v1_v11,2)
p_v2_v11 <- prop.table(table_v2_v11,2)
p_v3_v11 <- prop.table(table_v3_v11,2)
p_v4_v11 <- prop.table(table_v4_v11,2)
p_v5_v11 <- prop.table(table_v5_v11,2)
p_v6_v11 <- prop.table(table_v6_v11,2)
p_v7_v11 <- prop.table(table_v7_v11,2)
p_v8_v11 <- prop.table(table_v8_v11,2)
p_v9_v11 <- prop.table(table_v9_v11,2)
p_v10_v11 <- prop.table(table_v10_v11,2)

number_1 <- sum(binary_data_train$V11 ==1)
number_2 <- sum(binary_data_train$V11 ==2)
number_3 <- sum(binary_data_train$V11 ==3)

P_1 <- number_1/length(binary_data_train$V11)
P_2 <- number_2/length(binary_data_train$V11)
P_3 <- number_3/length(binary_data_train$V11)

probs_v11_numbers <- c(P_1, P_2, P_3)

results <- c()
for (i in 1:length(binary_data_test$V1)) {
    temp <- return_probs(binary_data_test[i,])
    results <- c(results, match(max(temp), temp))
}
```

2

a)

```
library('rpart')
library('rpart.plot')
student_data <- read.csv('student/student-mat.csv', sep=';')

student_data$class <- ifelse(student_data$G3 > 10,0,1)
new_dat <- student_data[1:30]
new_dat <- cbind(new_dat, student_data[34])

sample_ind <- sample(nrow(new_dat),nrow(new_dat)*0.70)
train <- new_dat[sample_ind,]
```

```
test <- new_dat[-sample_ind,]

set.seed(100)
fit <- rpart(formula = class~., data = train, method = "class", control = rpart.control(cp =

printcp(fit)

test$pred <- predict(fit, test, type = "class")

#pruning
fit_pruned <- prune(fit, cp = 0.038)
test$pred <- predict(fit_pruned, test, type = "class")

rpart.plot(fit_pruned)
```

b)

training error = root node error X relative error

training error = 0.47 * 0.62 = 0.2914

generalisation error = 0.47 * 0.6641 = 0.3121

c)

d)

```
student_data_reg <- read.csv('student/student-mat.csv', sep=';')
new_dat <- student_data[1:30]
new_dat <- cbind(new_dat, student_data[33])

sample_ind <- sample(nrow(new_dat), nrow(new_dat)*0.70)
train_reg <- new_dat[sample_ind,]
test_reg <- new_dat[-sample_ind,]

set.seed(100)
fit_reg <- rpart(formula = G3~., data = train_reg, method = "anova", control = rpart.control

printcp(fit_reg)

fit_pruned_reg <- prune(fit_reg, cp = 0.027)

rpart.plot(fit_pruned_reg)
```

4

a)

```
strange_data <- read.csv('strange_binary.csv')

set.seed(100)

fit_strange <- rpart(c~., data=strange_data, method='class', control = rpart.control(cp = 0)

plot(fit_strange)
fit_strange
printcp(fit_strange)

#          CP nsplit rel error xerror    xstd
# 1 0.0572917      0   1.00000 1.0000 0.10308
# 2 0.0312500      3   0.82812 1.0938 0.10540
# 3 0.0156250      4   0.79688 1.1094 0.10574
# 4 0.0078125      5   0.78125 1.1094 0.10574
# 5 0.0000000      9   0.75000 1.0781 0.10504

fit_pruned_strange <- prune(fit_strange, cp = 0.0312)
plot(fit_pruned_strange)

strange_predict <- predict(fit_pruned_strange, strange_data, type='class')
score <- mean(strange_predict == strange_data$c)
```

```
printcp(fit_pruned_strange)
```

The error on the test set will be different from that of training error. As you can see for the given tree, cp=0.00781 looks the most optimal parametre with the least validation error. Here the validation is much more than training error. So it cannot be said that training and test will be the same for this tree.

6

```
classification_data <- read.csv('classification_accuracy.csv')

wins_decision_tree <- sum(classification_data$decision_tree > classification_data$svm) + sun
wins_svm <- sum(classification_data$svm > classification_data$decision_tree) + sum(classific
wins_naive_bayes <- sum(classification_data$naive_bayes > classification_data$svm) + sum(cla

loss_decision_tree <- sum(classification_data$decision_tree < classification_data$svm) + sun
loss_svm <- sum(classification_data$svm < classification_data$decision_tree) + sum(classific
loss_naive_bayes <- sum(classification_data$naive_bayes < classification_data$svm) + sum(cla
```

```
draw_decision_tree <- sum(classification_data$decision_tree == classification_data$svm) + su
draw_svm <- sum(classification_data$svm == classification_data$decision_tree) + sum(classifi
draw_naive_bayes <- sum(classification_data$naive_bayes == classification_data$svm) + sum(cl
```