Project 2: Naïve Bayes Classifier

Advanced Big Data Methods

July 21, 2016

1 NB Function

Write your own Naïve Bayes classifier function in `R`. This function should be called `NB` and it should take three inputs: 1. *formula* describing the model that learning should be performed on 2. *data* denoting the data-frame in which the data reside on 3. *query* which is the instantiation of the predictor terms in the model. This function should output a list containing two elements: 1. *prediction* category of the dependent variable 2. *probability* of the prediction. Your program can *only* rely on the following external functions:
`is.formula, is.data.frame, is.vector, is.na, length, match, levels, nlevels, array, table, sum`

Please note that `is.formula` is in the `plyr` package. You can use the following code inside your function to parse the input formula:

```
m <- match.call(expand.dots = FALSE)
m[[1]] <- as.name("model.frame")
m <- m[-4]
m <- eval(m, parent.frame())
Terms <- attr(m, "terms")
if (any(attr(Terms, "order") > 1))
  stop("NB cannot handle interaction terms")
Y <- model.extract(m, "response")
X <- m[, -attr(Terms, "response"), drop = FALSE]
```

There are other functions available for parsing `R` formulas. Instead of the code above, you can use the `ParseFormula` function in the `DescTools` package.

Hint: start by implementing how the *PlayTennis* example discussed during class can be implemented in `R`. First, load this data in to a data-frame:

```
data <- read.csv("Tennis.csv")
data
```

```
##    Day  Outlook Temperature Humidity   Wind Play
## 1    1    Sunny         Hot     High   Weak   No
## 2    2    Sunny         Hot     High Strong   No
## 3    3 Overcast         Hot     High   Weak  Yes
## 4    4     Rain        Mild     High   Weak  Yes
## 5    5     Rain        Cool   Normal   Weak  Yes
## 6    6     Rain        Cool   Normal Strong   No
## 7    7 Overcast        Cool   Normal Strong  Yes
## 8    8    Sunny        Mild     High   Weak   No
## 9    9    Sunny        Cool   Normal   Weak  Yes
## 10  10     Rain        Mild   Normal   Weak  Yes
## 11  11    Sunny        Mild   Normal Strong  Yes
## 12  12 Overcast        Mild     High Strong  Yes
## 13  13 Overcast         Hot   Normal   Weak  Yes
## 14  14     Rain        Mild     High Strong   No
```

Then, calculate the probability of *Play* given the following situation: *Outlook* = `Sunny`, *Temperature* = `Cool`, *Humidity* = `High`, *Wind* = `Strong`. Next, write a function that accepts the following formula: $Play \sim Outlook + Temperature + Humidity + Wind$, and this query: $c(\text{“}Sunny\text{”}, \text{“}Cool\text{”}, \text{“}High\text{”}, \text{“}Strong\text{”})$. Finally, generalize this function so that it can handle any data-frame.

1.1 Examples:

```
formula <- Play ~ Outlook + Temperature + Humidity + Wind
query <- c("Sunny", "Cool", "High", "Strong")
NB(formula, data, query)

## [1] "No"                 "0.0205714285714286"

query <- c("Sunny", "Cool", "Normal", "Weak")
NB(formula, data, query)

## [1] "Yes"                "0.0211640211640212"

query <- c("Hot", "Cool", "Normal", "Weak")
NB(formula, data, query)

## Error in NB(formula, data, query):
## Query does not match the model

query <- c("Hot", "Cool", "Normal")
NB(formula, data, query)

## Error in NB(formula, data, query):
## Query does not match the model
```

```
query <- c("Sunny", "Cool", "Normal", "Weak")
NB(formula, c("1", "2"), query)
```

```
## Error in NB(formula, c("1", "2"), query):
## NB can only handle dataframes
```

```
query <- data[data$Outlook == "Sunny", ]
NB(formula, data, query)
```

```
## Error in NB(formula, data, query):
## Query needs to be a vector
```

```
houseData <- read.csv("house-votes-84.data",na.strings="?")
colnames(houseData) <- c("party",       "handicapped-infants",
                    "water-project-cost-sharing",
                    "adoption-of-the-budget-resolution",
                    "physician-fee-freeze",
                    "el-salvador-aid",
                    "religious-groups-in-schools",
                    "anti-satellite-test-ban",
                    "aid-to-nicaraguan-contras",
                    "mx-missile",
                    "immigration",
                    "synfuels-corporation-cutback",
                    "education-spending",
                    "superfund-right-to-sue",
                    "crime",    "duty-free-exports",
                    "export-administration-act-south-africa")


mydata <- houseData[complete.cases(houseData),]
formula <- party ~ .
query <- c(as.vector(t(mydata[2,-1])))
NB(formula,mydata[-2,],query)
```

```
## [1] "republican"          "0.00914842096885727"
```

2 Leave-One-Out Cross-Validation

Write a function that performs $Leave-One-OutCross-Validation$ using the `NB` function. This function should input the following two variables: 1. $formula$ 2. $data$. It should output the classification accuracy of the model. The only external functions, in addition to the ones listed in Section 1, that can be used in the `LOOCV` function is `as.vector`.

```
data <- read.csv("Tennis.csv")
formula <- Play ~ Outlook + Temperature + Humidity + Wind
LOOCV(formula,data)

## [1] 0.5714286

formula <- Outlook ~ Temperature + Humidity + Wind + Play
LOOCV(formula,data)

## [1] 0.1428571

formula <- party ~ .
LOOCV(formula,mydata)

## [1] 0.9181034
```

3 Sentiment Analysis

Use the NB and LOOCV functions to perform sentiment analysis on labeled text. You should use the spam/no-spam example in the slides as the guiding example. Use this function to predict the `sentiment` category of the `First GOP Debate Twitter Sentiment` corpus. You can use the `tm` and `SnowballC` packages. Hint: if using the `tm` package, use the `weightBin` weight function. Also, you can simplify the input by assuming that the input data-frame will only have two columns: 1. sentiment 2. text. Once the *sentiment* function is implemented, explore the Twitter data by running sentiment analysus per candidate, subject matter, confidence in sentiment, etc.

Optional: 1. If following the procedure above, your code probably runs very slowly and it will take it hours to finish analyzing the whole data-set. Can you modify your to optimize this process? 2. Can the other information in the data-set (such as "candidate" and "subject-matter") be used to improve the accuracy of the classification?

```
## Loading required package:  NLP
```

```
load("sentiment")
data <- subset(r,select=c("sentiment","text"))
sentiment(data)

## [1] 0.5836638

DT <- as.data.frame(
    lapply(subset(r, candidate=="Donald Trump"),
           function(x) if(is.factor(x)) factor(x) else x
```

```
    )
)
data <- subset(DT,select=c("sentiment","text"))
sentiment(data)

## [1] 0.6469961

JK <- as.data.frame(
    lapply(subset(r, candidate=="John Kasich"),
           function(x) if(is.factor(x)) factor(x) else x
    )
)
data <- subset(JK,select=c("sentiment","text"))
sentiment(data)

## [1] 0.4958678
```