Lab 2

Loyd Flores

11:59PM February 20

Basic Modeling

- In class we considered a variable `x_3` which measured "criminality". We imagined L = 4 levels "none", "infraction", "misdemeanor" and "felony". Create a variable `x_3` here with 100 random elements (equally probable). Create it as a nominal (i.e. unordered) factor.

```
#TO-DO
n = 100 # number of elements
x_3 = as.factor(
  sample(
    c( "none", "infraction", "misdemeanor", "felony"),
    size=100,
    replace=TRUE)
  )
x_3
```

```
##   [1] none        none        infraction  infraction  none        none
##   [7] infraction  felony      misdemeanor infraction  misdemeanor felony
##  [13] misdemeanor felony      none        infraction  infraction  none
##  [19] none        none        none        felony      felony      infraction
##  [25] misdemeanor none        felony      felony      misdemeanor infraction
##  [31] infraction  felony      felony      infraction  misdemeanor infraction
##  [37] misdemeanor felony      none        infraction  none        misdemeanor
##  [43] infraction  infraction  felony      none        none        misdemeanor
##  [49] infraction  misdemeanor felony      infraction  infraction  none
##  [55] misdemeanor infraction  misdemeanor infraction  misdemeanor infraction
##  [61] none        none        felony      felony      infraction  none
##  [67] infraction  felony      felony      none        none        felony
##  [73] none        none        felony      felony      misdemeanor felony
##  [79] felony      infraction  none        none        misdemeanor none
##  [85] infraction  misdemeanor none        none        none        none
##  [91] none        infraction  infraction  none        misdemeanor none
##  [97] infraction  infraction  infraction  felony
## Levels: felony infraction misdemeanor none
```

```
# measure criminality
# Replace = True allows duplicates to fulfill size
```

- Use `x_3` to create `x_3_bin`, a binary feature where 0 is no crime and 1 is any crime.

```r
#TO-DO
x_3_bin = ifelse(x_3 == "none", 0, 1)
x_3_bin
```

```
##   [1] 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 0 0 1 0 0
##  [75] 1 1 1 1 1 1 0 0 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 1
```

- Use `x_3` to create `x_3_ord`, an ordered factor variable. Ensure the proper ordinal ordering.

```r
#TO-DO
crime_level = c("none", "infraction", "misdemeanor", "felony")
x_3_ord = factor(x_3, ordered=TRUE, levels=crime_level)
x_3_ord
```

```
##   [1] none        none        infraction  infraction  none        none
##   [7] infraction  felony      misdemeanor infraction  misdemeanor felony
##  [13] misdemeanor felony      none        infraction  infraction  none
##  [19] none        none        none        felony      felony      infraction
##  [25] misdemeanor none        felony      felony      misdemeanor infraction
##  [31] infraction  felony      felony      infraction  misdemeanor infraction
##  [37] misdemeanor felony      none        infraction  none        misdemeanor
##  [43] infraction  infraction  felony      none        none        misdemeanor
##  [49] infraction  misdemeanor felony      infraction  infraction  none
##  [55] misdemeanor infraction  misdemeanor infraction  misdemeanor infraction
##  [61] none        none        felony      felony      infraction  none
##  [67] infraction  felony      felony      none        none        felony
##  [73] none        none        felony      felony      misdemeanor felony
##  [79] felony      infraction  none        none        misdemeanor none
##  [85] infraction  misdemeanor none        none        none        none
##  [91] none        infraction  infraction  none        misdemeanor none
##  [97] infraction  infraction  infraction  felony
## Levels: none < infraction < misdemeanor < felony
```

- Convert this variable into three binary variables without any information loss and put them into a data matrix.

```r
#TO-DO
bin_vector1 = ifelse(x_3_ord == "infraction", 1, 0)
bin_vector2 = ifelse(x_3_ord == "misdemeanor", 1, 0)
bin_vector3 = ifelse(x_3_ord == "felony", 1, 0)
crime_bin_vectors = cbind(bin_vector1, bin_vector2, bin_vector3)

colnames(crime_bin_vectors) = c(
  "[,1]",
  "[,2]",
  "[,3]"
)

#crime_bin_vectors
```

- What should the sum of each row be (in English)?

#TO-DO

Verify that.

```
#TO-DO
rowSums(crime_bin_vectors)
```

```
##   [1] 0 0 1 1 0 0 1 1 1 1 1 1 1 1 0 1 1 0 0 0 0 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 0 1 0 1 1 1 1 0 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 0 1 1 1 0 1 1 1 0 0 1 0 0
##  [75] 1 1 1 1 1 1 0 0 1 0 1 1 0 0 0 0 0 1 1 0 1 0 1 1 1 1
```

```
# A person could only be of 1 level : "none" , "infraction", "misdemeanor", "felony"
```

- How should the column sum look (in English)?

#TO-DO Column sums should be counts of number of each crime level in the data

Verify that.

```
#TO-DO
colSums(crime_bin_vectors)
```

```
## [,1] [,2] [,3]
##   29   17   22
```

- Generate a matrix with 100 rows where the first column is realization from a normal with mean 17 and variance 38, the second column is uniform between -10 and 10, the third column is poisson with mean 6, the fourth column in exponential with lambda of 9, the fifth column is binomial with n = 20 and p = 0.12 and the sixth column is a binary variable with exactly 24% 1's dispersed randomly. Name the rows the entries of the `fake_first_names` vector.

```
# cbind == concatenate arrays to make a matrix
X = cbind(
  rnorm(n, mean=17, sd=sqrt(38)), #  realization from a normal with mean 17 and variance 38
  runif(n, -10, 10),                # uniform between -10 and 10
  rpois(n, 6),                      # poisson with mean 6
  rexp(n, 9),                       # fourth column in exponential with lambda of 9
  rbinom(n, 20, .12),               # binomial with n = 20 and p = 0.12
  sample(c(rep(1, round(n*.24)), rep(0, round(n*.76)))) # binary variable with exactly 24%
  # random 1's dispersed

)
fake_first_names = c(
  "Sophia", "Emma", "Olivia", "Ava", "Mia", "Isabella", "Riley",
  "Aria", "Zoe", "Charlotte", "Lily", "Layla", "Amelia", "Emily",
  "Madelyn", "Aubrey", "Adalyn", "Madison", "Chloe", "Harper",
  "Abigail", "Aaliyah", "Avery", "Evelyn", "Kaylee", "Ella", "Ellie",
  "Scarlett", "Arianna", "Hailey", "Nora", "Addison", "Brooklyn",
  "Hannah", "Mila", "Leah", "Elizabeth", "Sarah", "Eliana", "Mackenzie",
  "Peyton", "Maria", "Grace", "Adeline", "Elena", "Anna", "Victoria",
  "Camilla", "Lillian", "Natalie", "Jackson", "Aiden", "Lucas",
  "Liam", "Noah", "Ethan", "Mason", "Caden", "Oliver", "Elijah",
```

```r
  "Grayson", "Jacob", "Michael", "Benjamin", "Carter", "James",
  "Jayden", "Logan", "Alexander", "Caleb", "Ryan", "Luke", "Daniel",
  "Jack", "William", "Owen", "Gabriel", "Matthew", "Connor", "Jayce",
  "Isaac", "Sebastian", "Henry", "Muhammad", "Cameron", "Wyatt",
  "Dylan", "Nathan", "Nicholas", "Julian", "Eli", "Levi", "Isaiah",
  "Landon", "David", "Christian", "Andrew", "Brayden", "John",
  "Lincoln"
)

#TO-DO
#rownames(X) = fake_first_names
# X

# removed output for pdf conversion
```

- Create a data frame of the same data as above except make the binary variable a factor "DOMESTIC" vs "FOREIGN" for 0 and 1 respectively. Use RStudio's `View` function to ensure this worked as desired.

```r
#TO-DO
X_df = data.frame(
  normie = rnorm(n, mean=17, sd=sqrt(38)),
  eunice = runif(n, -10, 10),
  fish = rpois(n, 6),
  expy= rexp(n, 9),
  nomie = rbinom(n, 20, .12),
  origin = factor(sample(c(rep("FOREIGN", round(n*.24)), rep("DOMESTIC", round(n*.76)))))
  )
#rownames(X_df) = fake_first_names
#X_df
```

- Print out a table of the binary variable. Then print out the proportions of "DOMESTIC" vs "FOREIGN".

```r
#TO-DO
# X_df$origin -> Returns count for values within
manual_props_table = table(X_df$origin)/nrow(X_df)
manual_props_table
```

```
##
## DOMESTIC  FOREIGN
##     0.76     0.24
```

```r
prop.table(table(X_df$origin))
```

```
##
## DOMESTIC  FOREIGN
##     0.76     0.24
```

Print out a summary of the whole dataframe.

```
#TO-DO
summary(X_df)
```

```
##      normie           eunice            fish           expy
##  Min.   :-0.9224   Min.   :-9.6660   Min.   : 1.00   Min.   :0.002796
##  1st Qu.:11.4780   1st Qu.:-4.8685   1st Qu.: 4.00   1st Qu.:0.024227
##  Median :16.7780   Median :-0.8305   Median : 6.00   Median :0.077206
##  Mean   :16.0953   Mean   :-0.1110   Mean   : 6.18   Mean   :0.104276
##  3rd Qu.:20.7960   3rd Qu.: 5.5705   3rd Qu.: 8.00   3rd Qu.:0.141978
##  Max.   :32.0573   Max.   : 9.9584   Max.   :13.00   Max.   :0.487444
##      nomie          origin
##  Min.   :0.00   DOMESTIC:76
##  1st Qu.:1.00   FOREIGN :24
##  Median :2.00
##  Mean   :2.35
##  3rd Qu.:3.00
##  Max.   :7.00
```

Dataframe creation

Imagine you are running an experiment with many manipulations. You have 14 levels in the variable "treatment" with levels a, b, c, etc. For each of those manipulations you have 3 submanipulations in a variable named "variation" with levels A, B, C. Then you have "gender" with levels M / F. Then you have "generation" with levels Boomer, GenX, Millenial. Then you will have 6 runs per each of these groups. In each set of 6 you will need to select a name without duplication from the appropriate set of names (from the last question). Create a data frame with columns treatment, variation, gender, generation, name and y that will store all the unique unit information in this experiment. Leave y empty because it will be measured as the experiment is executed. Hint, we've been using the `rep` function using the `times` argument. Look at the `each` argument using `?rep`.

```
names = list(
    Boomer = list(
        M = strsplit("Theodore, Bernard, Gene, Herbert, Ray, Tom, Lee, Alfred, Leroy, Eddie", ", ")[[1]
        F = strsplit("Gloria, Joan, Dorothy, Shirley, Betty, Dianne, Kay, Marjorie, Lorraine, Mildred",
    ),
    GenX = list(
        M = strsplit("Marc, Jamie, Greg, Darryl, Tim, Dean, Jon, Chris, Troy, Jeff", ", ")[[1]],
        F = strsplit("Tracy, Dawn, Tina, Tammy, Melinda, Tamara, Tracey, Colleen, Sherri, Heidi", ", ")
    ),
    Millenial = list(
        M = strsplit("Zachary, Dylan, Christian, Wesley, Seth, Austin, Gabriel, Evan, Casey, Luis", ",
        F = strsplit("Samantha, Alexis, Brittany, Lauren, Taylor, Bethany, Latoya, Candice, Brittney, C
    )
)

#names

n = 14 * 3 * 2 * 3 * 6

X = data.frame(
  treatment=rep(letters[1:14], each=3 * 2 * 3 * 6),
  variation=rep(LETTERS[1:3], each=2 * 3 * 6, times=14),
```

```r
  gender=rep(c('M', 'F'), each=3 * 6, times=14 * 3),
  generation=rep(c("Boomer", "GenX", "Millenial"), each=6, times=14 * 3 * 2),
  name=0
)

X2 = data.frame(expand.grid(
  name=rep(NA,6),
  generation=c("Boomer", "GenX", "Millenial"),
  gender=rep(c('M', 'F')),
      variation=rep(LETTERS[1:3]),
          treatment=rep(letters[1:14])

  ))


#X2

for (i in seq(from=1, to=n, by=6)) {
  X$names[i:(i+5)] = sample(names[[X$generation[i]]][[X$gender[i]]], 6)
}

for (i in seq(from=1, to=n, by=6)) {
  X2$names[i:(i+5)] = sample(names[[X2$generation[i]]][[X2$gender[i]]], 6)
}


#X
#X2
```

Basic Binary Classification Modeling

- Load the famous **iris** data frame into the namespace. Provide a summary of the columns using the **skim** function in package **skimr** and write a few descriptive sentences about the distributions using the code below in English.

```r
#TO-DO
data("iris")

pacman::p_load(skimr)

skim(iris)
```

Table 1: Data summary

Name	iris
Number of rows	150
Number of columns	5
Column type frequency:	
factor	1
numeric	4

Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
Species	0	1	FALSE	3	set: 50, ver: 50, vir: 50

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
Sepal.Length	0	1	5.84	0.83	4.3	5.1	5.80	6.4	7.9	
Sepal.Width	0	1	3.06	0.44	2.0	2.8	3.00	3.3	4.4	
Petal.Length	0	1	3.76	1.77	1.0	1.6	4.35	5.1	6.9	
Petal.Width	0	1	1.20	0.76	0.1	0.3	1.30	1.8	2.5	

TO-DO: We have dataset that has plant information. Based on the skim there are no missing values in each category/feature and has a complete rate of 100%. It also shows the mean values in each feature as well as the standard deviation and percentile values.

The outcome / label / response is `Species`. This is what we will be trying to predict. However, we only care about binary classification between "setosa" and "versicolor" for the purposes of this exercise. Thus the first order of business is to drop one class. Let's drop the data for the level "virginica" from the data frame.

```
#TO-DO
# We Want to make it a binary classification problem between
# setosa, versicolor hence we drop the extra column 'virginica'

iris_binary = subset(iris, subset=Species != "virginica")

#iris
#iris_binary
#temp = sum(iris_binary$Species == "virginica")
#temp
```

Now create a vector `y` that is length the number of remaining rows in the data frame whose entries are 0 if "setosa" and 1 if "versicolor".

```
#TO-DO
y = ifelse(iris_binary$Species == "setosa", 0, 1)
y
```

```
##   [1] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
##  [38] 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
```

```
setosa_count = sum(y == 1)
setosa_count
```

```
## [1] 50
```

```
versicolor_count = sum(y == 0)
versicolor_count
```

```
## [1] 50
```

- Write a function `mode` returning the sample mode of a vector of numeric values. Try not to look in the class notes.

```
#TO-DO
# we're trying the mode for g0
mode_function = function(x) {
    # Gets unique values
    uniq_x = unique(x)
    # match : Each element of the original vector is matched to its index in the unique elements.
    # Tabulate : Tabulates counts the occurences of the matched values
    # return the index of the max
    return(uniq_x[which.max(tabulate(match(x, uniq_x)))])
}

multiple_mode_function <- function(x) {
    uniq_x = unique(x)
    freq = tabulate(match(x, uniq_x))
    max_freq = max(freq)
    # Returns all the indices if multiple modes
    modes = uniq_x[freq == max_freq]
    return(modes)
}


# Example usage
single_mode = mode_function(y)
mult_mode = multiple_mode_function(y)

single_mode
```

```
## [1] 0
```

```
mult_mode
```

```
## [1] 0 1
```

- Fit a threshold model to `y` using the feature `Sepal.Length`. Write your own code to do this. What is the estimated value of the threshold parameter? Save the threshold value as `threshold`.

```
#TO-DO
best_threshold = NULL
best_accuracy = 0
for (threshold in iris_binary$Sepal.Length) {
```

```r
    # Classify based on the threshold
    # we compare threshold (or each sepal length)
    # predictions return a vector of size of sepal.length
    # creates a vector of predictions based on the current threshold

    predictions = ifelse(iris_binary$Sepal.Length <= threshold, 0, 1)

    # accuracy of predictions
    # sum(predictions == y) counts how many times we predicted right
    accuracy = sum(predictions == y) / length(y)

    # Update the best threshold if we get a better accuracy
    if (accuracy > best_accuracy) {
        best_accuracy = accuracy
        best_threshold = threshold
    }
}


print(best_threshold)
```

```
## [1] 5.4
```

```r
threshold = best_threshold
```

What is the total number of errors this model makes?

```r
#TO-DO
best_predictions = ifelse(iris_binary$Sepal.Length <= best_threshold, 0, 1)
count_correct_preds = sum(best_predictions == y)

error = 1 - (count_correct_preds / length(y))
cat("Error:",error,"%")
```

```
## Error: 0.11 %
```

Does the threshold model's performance make sense given the following summaries:

```r
threshold
```

```
## [1] 5.4
```

```r
summary(iris[iris$Species == "setosa", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.300   4.800   5.000   5.006   5.200   5.800
```

```r
summary(iris[iris$Species == "versicolor", "Sepal.Length"])
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   4.900   5.600   5.900   5.936   6.300   7.000
```

TO-DO: Write your answer here in English. Answer : It makes sense. The threshold is the cutoff point or the point where separates the two binary classes. The mean sepal length of setosa's are 5.006 which is below the thresh hold. The mean length of versicolors is 5.9 which is above the mean. Our thresold makes a split in the middle. There may be some interlaps or edge cases on both where a setosa could be above the threshold and a versicolor could be below the threshold.

Create the function `g` explicitly that can predict `y` from `x` being a new `Sepal.Length`.

```
g = function(x){
  return(ifelse(x <= threshold, "Setosa", "Versicolor"))
}

test_x = sample(iris_binary$Sepal.Length, size=1)
cat("Sepal Length:", test_x, "is a", g(test_x))
```

```
## Sepal Length: 5.5 is a Versicolor
```