Data Cleaning with R

Due Friday, April 26 by 11:59pm.

Introduction

We will be using `R` to do all of our data-driven decision-making in this class. For this assignment and for the next assignment, I assume that you already know how to use `R`. If you do not know how to use `R`, or are still a beginner, I would highly suggest looking at some resources online such as the book, "R for Data Science" by Garrett Grolemund and Hadley Wickham (https://r4ds.had.co.nz). That said, I will try to explain a lot of the `R` code to make it as easy as possible to pick up.

In this worksheet, we will go over some basic data manipulation techniques to help get your data ready to run machine learning models such as decision tree classifiers.

Motivation: Identifying Spam Email

Suppose you want to decide which emails to treat seriously and which emails to delete as spam (ignoring any spam filters you may already have in place). You might be able to make some heuristic rules about which emails are spam, such as ones that say you're a winner, or aren't responses, but how well can these heuristics actually work? And, is there a better way to identify spam emails?

When you have data to help guide your decision-making, you have the opportunity to make your decisions as objective as possible. With so many pitfalls with heuristics, and the possibility for biases to cloud your judgement at every turn, the ability to make decisions based on hard data can be extremely useful. In this example, we are trying to figure out how to best detect spam email, so we can decide which ones to read. While it might be possible to do this ourselves without the benefit of statistical models, it would probably not be as good, and take much longer to do so.

Here, we'll start by simply bringing in a dataset, exploring the dataset, and working with it to set it up for the models that we'll run. This might seem like "boring" work, but it's actually the step that will likely take most of your time in real life, so it's very important to learn how to do this part well.

Getting the Data

Make sure you download the `email.csv` file from ELMS. This is the dataset that we'll be using for all of the examples here. Let's bring that data in from the CSV file. Remember to save the file into the same folder as your `R` working directory, or to specify the full path name.

```r
email <- read.csv('~/Downloads/email.csv', header = TRUE)
```

Now that we've done this, we have a Data Frame object called `email`. You can look at the top few rows of the dataset using the `head()` function.

```r
head(email)
```

If we want to look at the structure of the `email` data frame, we can use `str()`.

```r
str(email)
```

This can help you determine types of variables, but be careful! Just because a variable has numbers doesn't mean it's a numerical variable!

We can check the number of rows and columns with the following commands:

```
nrow(email) # this is for the number of rows
ncol(email) # this is for the number of columns
dim(email) # this is for the dimensions (rows, columns)
```

Initial Exploration

Let's get some initial summary statistics about the data.

```
summary(email)
```

```
##        X              spam          to_multiple          from
##  Min.   :   1   Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.: 981   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:1.0000
##  Median :1961   Median :0.00000   Median :0.0000   Median :1.0000
##  Mean   :1961   Mean   :0.09361   Mean   :0.1581   Mean   :0.9992
##  3rd Qu.:2941   3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:1.0000
##  Max.   :3921   Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
##                 NA's   :22
##        cc            sent_email                        time
##  Min.   : 0.0000   Min.   :0.000   2012-03-03 17:40:27:   5
##  1st Qu.: 0.0000   1st Qu.:0.000   2012-03-14 17:10:42:   4
##  Median : 0.0000   Median :0.000   2012-03-23 07:07:28:   4
##  Mean   : 0.4045   Mean   :0.278   2012-01-26 07:10:40:   3
##  3rd Qu.: 0.0000   3rd Qu.:1.000   2012-03-16 00:58:22:   3
##  Max.   :68.0000   Max.   :1.000   2012-03-16 21:08:53:   3
##                                    (Other)            :3899
##      image              attach            dollar         winner
##  Min.   : 0.00000   Min.   : 0.0000   Min.   : 0.000   no  :3823
##  1st Qu.: 0.00000   1st Qu.: 0.0000   1st Qu.: 0.000   yes :  64
##  Median : 0.00000   Median : 0.0000   Median : 0.000   NA's:  34
##  Mean   : 0.04846   Mean   : 0.1329   Mean   : 1.467
##  3rd Qu.: 0.00000   3rd Qu.: 0.0000   3rd Qu.: 0.000
##  Max.   :20.00000   Max.   :21.0000   Max.   :64.000
##
##     inherit          viagra            password          num_char
##  Min.   :0.000   Min.   :0.00000   Min.   : 0.0000   Min.   :  0.001
##  1st Qu.:0.000   1st Qu.:0.00000   1st Qu.: 0.0000   1st Qu.:  1.459
##  Median :0.000   Median :0.00000   Median : 0.0000   Median :  5.856
##  Mean   :0.038   Mean   :0.00204   Mean   : 0.1081   Mean   : 10.707
##  3rd Qu.:0.000   3rd Qu.:0.00000   3rd Qu.: 0.0000   3rd Qu.: 14.084
##  Max.   :9.000   Max.   :8.00000   Max.   :28.0000   Max.   :190.087
##
##   line_breaks         format          re_subj         exclaim_subj
##  Min.   :   1.0   Min.   :0.0000   Min.   :0.0000   Min.   :0.00000
##  1st Qu.:  34.0   1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000
##  Median : 119.5   Median :1.0000   Median :0.0000   Median :0.00000
##  Mean   : 231.2   Mean   :0.6952   Mean   :0.2614   Mean   :0.08034
##  3rd Qu.: 299.0   3rd Qu.:1.0000   3rd Qu.:1.0000   3rd Qu.:0.00000
##  Max.   :4022.0   Max.   :1.0000   Max.   :1.0000   Max.   :1.00000
##  NA's   :17
##   urgent_subj        exclaim_mess        number
##  Min.   :0.000000   Min.   :   0.000   big  : 545
```

```
##  1st Qu.:0.000000   1st Qu.:   0.000   none : 549
##  Median :0.000000   Median :   1.000   small:2827
##  Mean   :0.001785   Mean   :   6.584
##  3rd Qu.:0.000000   3rd Qu.:   4.000
##  Max.   :1.000000   Max.   :1236.000
##
```

This can be a lot of information to take in at once, so individual variables work as well.

```
summary(email$spam)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
## 0.00000 0.00000 0.00000 0.09361 0.00000 1.00000      22
```

But wait, the `spam` variable is actually a categorical variable (0 means it is not spam and 1 means it is). Since it's been coded as numbers, we need to change it to a categorical variable to make sure the `R` recognizes it as such.

```
email$spam <- factor(email$spam)
summary(email$spam)
```

```
##    0    1 NA's
## 3534  365   22
```

The `factor` function changes the type of the variable into a categorical one, so it now creates the table instead of providing a five number summary like it did before.

In addition, notice how some of these variables, such as `spam` and `winner` have NA values in them. Let's find out which rows of `spam` contain NAs.

```
which(is.na(email$spam))
```

```
##  [1]   26  179  229  463  645  882  994 1008 1014 1093 1097 1241 1618 1814
## [15] 2180 2249 2515 3025 3185 3258 3371 3680
```

The `is.na` function gives a good way of finding out which values are NAs, which typically happens when there are some weird characters or there simply was no value to begin with. Usually, with messy real-world datasets, you'll have NA values in at least some of your variables, so it's helpful to know how to deal with them. The `which` function finds the indices of where those NA values are. We can use this information to subset our data to not include those NA values, which we'll go over in the next section.

Creating Subsets of the Data

Many times, especially with larger datasets and datasets where you're trying to remove rows with NA values, you'll want to only look at certain portions of the data. We can subset our dataset by indicating which rows we want. We do this using a vector of boolean values with `TRUE` whenever it's a row that we want and `FALSE` whenever it's a row we don't want. This vector should have a length equal to the number of rows in the data frame. To get this vector, we can just use a logical operator. For example, consider the following code.

```
email$spam == 0
```

This evaluates every element of the `email$spam` vector and returns a vector of the same length with `TRUE` and `FALSE` values. We can then use this vector to subset the dataset so that we only have the rows with `TRUE` values.

```
nospam <- email[email$spam == 0,]
head(nospam)
```

The new object we've created, `nospam`, is a data frame that contains only the rows from `email` with a value of 0 in `spam`. We can also use other logical operators to create different subsets.

```
# What do you think this subset is?
example_subset <- email[email$spam == 0 & email$line_breaks < 10,]
head(example_subset)
```

The following table shows a list of some logical operators you might find useful.

Operator	Meaning
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
==	equal to
!=	not equal to
&	and
|	or

Recall that we found the indices of the NA values in the `spam` variable. We can use this to create a data frame with no NA values in the `spam` variable.

```
email_clean <- email[!is.na(email$spam),]
dim(email_clean)
```

```
## [1] 3899   22
```

The exclamation point is the same thing as "not", so this subsets the data frame to only include the rows in which the value of spam is not NA.

Creating a Random Sample of Rows

When we start actually running our classification tree models, we'll want to split up our dataset into what we call **train** and **test** sets. We'll go over why we do this more in the next assignment, but for now, let's go over how to create a random sample of rows.

The most basic way to do this is to get a random sample of indices, then just select the appropriate rows. Here, we'll take a random 10% of the dataset and set that aside as our "test" dataset.

```
# Find what 10% is (we use floor to have a round number)
num_test <- floor(nrow(email_clean) * 0.1)

# Sample the indices
test_rows <- sample(1:nrow(email_clean),num_test)

# Create test set with indices
test_email <- email_clean[test_rows,]

# Create train set with the rest
train_email <- email_clean[-test_rows,]
```

We're doing this in steps, first finding how many rows to take a random sample of, then using the `sample` function to draw that many numbers. Then, we use those indices to create the test set, subsetting the data frame so that it only contains those rows, and the rest get relegated to the train set, which we subset by putting a "-" in front of the indices we don't want.

A Note on Data Manipulation Tools

So far, all of the data manipulation has been done in base `R`. There is a very popular suite of packages called the `tidyverse`, which has tools for data manipulation (`dplyr`), graphing (`ggplot2`) and more. They all are based on the same underlying design philosophy and grammar, making many of the tasks you'll have to do easier to read (in my opinion). In fact, these are so popular that they might nowadays almost be the default way of handling data.

You don't necessarily need to use the `tidyverse` suite of packages, but I highly recommend it if you plan on using `R` in some capacity in the future.

Using `dplyr`

Recall that I mentioned "grammar" in using these tools. The idea behind these packages is that we have our dataset (in our case, a data frame object), which is the noun, and actions we can perform with that dataset, which is the verb. Here, I will introduce a few useful verbs for manipulating datasets.

Subsetting rows using `filter`

`filter()` is the basic way of subsetting the data frame by selecting certain rows.

```
# First make sure you install and load dplyr!
# install.package('dplyr')
require(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.5.2
```

```
nospam2 <- filter(email, spam == 0)
head(nospam2)
```

Note that we have "`spam == 0`" and not "`email$spam == 0`".

To filter based on more than one criteria, we can just add more arguments. This does the same thing as subsetting by the "`&`" of the two conditions.

```
nospam2 <- filter(email, spam == 0, to_multiple == 0)
# This does the same thing as:
# nospam2 <- email[email$spam == 0 & email$to_multiple == 0,]
head(nospam2)
```

Subsetting by column with `select()`

You can subset the dataset by column using the `select()` function (some of you might note that this is the same terminology used in SQL).

```
head(select(email, spam, to_multiple))
```

Re-ordering rows with `arrange()`

Sometimes, we want to re-order our dataset by a certain variable. To do this, we can use `arrange()`.

```
arrange(email, line_breaks)
```

This changes the order of the email data frame so that it is in increasing order of the `line_breaks` variable. If we wanted descending order, we could have used `desc()`.

```r
arrange(email, desc(line_breaks))
```

Using Pipes

Sometimes, we want to do many of these actions at the same time. For example, if we want to take a subset of rows, select only a few columns, then re-order the dataset, we might do something like this:

```r
# What would this do?
newemail <- arrange(
  select(
    filter(email,spam == 0),
    spam, to_multiple, from, cc, line_breaks),
  line_breaks)
head(newemail)
```

This filters the email data frame to contain only rows with `spam == 0`, then selects only the `spam`, `to_multiple`, `from`, `cc`, and `line_breaks` columsn, then arranges it by `line_breaks` in increasing order. Unfortunately, this is very hard to read.

To make everything more readable, `dplyr` brings in the `%>%` operator from the `magrittr` package (you don't need to worry about the `magrittr` package for now). `x %>% f(y)` turns into `f(x,y)`, so we can pipe in functions one at a time to make the code more readable. Instead of having to read the code inside out, we can read it sequentially to see what it is doing.

Let's see it in action using the example we used above.

```r
# This does the same as above!
newemail <- email %>%
  filter(spam == 0) %>%
  select(spam, to_multiple, from, cc, line_breaks) %>%
  arrange(line_breaks)
head(newemail)
```

This takes the `email` data frame, does the filtering on `spam`, selects the variables we want, then arranges by `line_breaks`. In this way, we have our "noun" data frame, `email`, first, then all of the "verbs", or functions, are written in their order of execution.

Questions

First, make sure you download the `kickstarter.csv` file from ELMS.

```r
kickstarter <- read.csv('~/Downloads/kickstarter.csv', header = TRUE, row.names = 1)
```

Note that we're using an additional argument to specify row names, because one of the columns contains the indices of the rows.

This dataset represents a sample of kickstarter campaigns, including information about the name of the kickstarter, state (whether it failed or was successful), the category, how much money was pledged (in USD), what its initial goal was, and more.

Using the kickstarter dataset and the information above, answer the following questions.

1) Suppose you are trying decide whether to dedicate a significant amount of effort into building and launching a kickstarter project. However, you are unsure of what to do to make sure it is successful or even you should do it at all. Describe how you might use the kickstarter data to help make your decision. (3 points)

2) How many rows and columns does the kickstarter dataset have? (2 points)

3) What are the variables in the kickstarter dataset? Which ones might be relevant in trying to determine whether a kickstarter was successful or not? (2 points)

4) Which variables have NA values in them? (2 points)

5) What was the average number of people who pledged to a kickstarter project in this daaset? What was the highest number of backers that a kickstarter project in this dataset had? (2 points)

6) What is the code you might use to subset the dataset to only include non-NA values for the variable called "state"? You do not need to include the data frame in your answer – just include the code (but make sure you run it to confirm that it works!). (2 points)

7) Suppose you wanted to create a train and test split of the data with 20% of the data as your test. What would be the code to do this? As before, you do not need to include the actual data frames, only the code. (4 points)