Shiny/EDA Tutorial

Ted Laderas

July 18, 2016

1.1 Our Goal For Today

The goal of today's workshop is to examine a set of genes across a number of cohorts for Dream Challenge Dataset. We will examine the subsetted data using the visualization framework built in R called Shiny and ask the question: Do we think there is enough signal in the exposure data to warrant further study of this dataset?

In order to do so, we'll first need to explore the data format and explore the data at two levels:

1. Single gene level (with multiple patients)
2. Aggregated pathway level ()

1.2 Study Design

In this dataset for the DREAM Respiratory Virus Challenge there are seven studies in total with the following characteristics. The most important is the Duke Rhinovirus dataset, which contained patients who were exposed to Rhinovirus (notated as `rhino`) and patients who were exposed to a control (annotated as `SHAM`). We will only examine two of the studies: the Duke Rhinovirus study and the DEE3 H1N1 study (annotated as `DEE3 H1N1`).

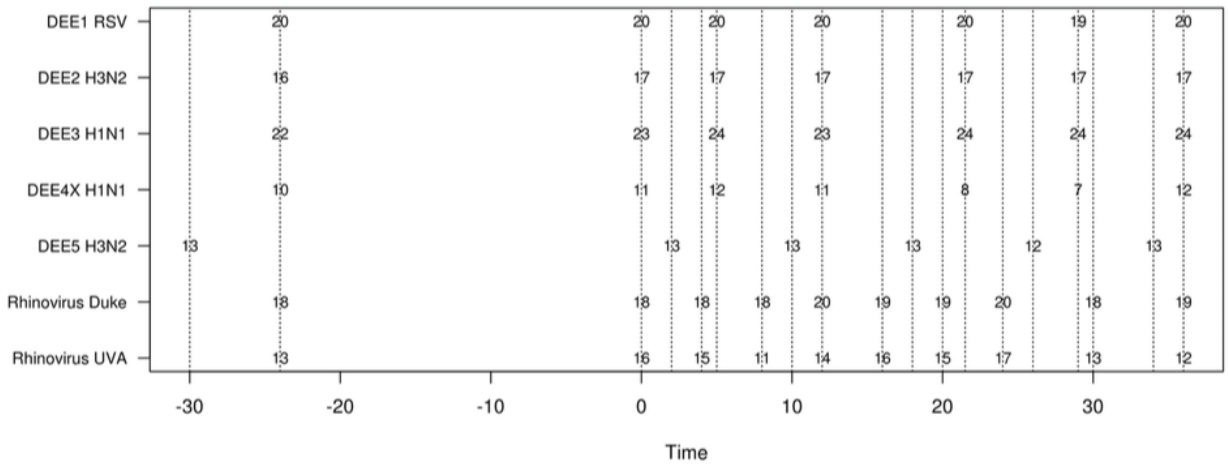

Figure 1: Study Design of Dataset

Basically, for most of the timepoints and for most of the patients within a study, we have a microarray measurement. Note that some patients are missing certain timepoints, which complicates our analysis.

The set of 20,000 probesets from the Affymetrix U133 2.0 microarray were mapped to the gene identifiers (in this case Gene Symbols) using the `hgu133plus2.db` annotation package and filtered using the following approach: Coefficients of Variation for the Sham (non exposed) patients at all timepoints were compared to the CVs for those patients exposed to Rhinovirus. The thought behind this is that genes of interest should show more variability (due to the time-series design) in the Rhinovirus patients than the sham patients.

Based on visualizing the distributions of CVs, a cutoff of `cvRhinovirus/cvSham` > 1.4 was used, leading to a set of 836 genes that had higher observed variability in the Rhinovirus compared to Shams. We'll be visualizing this much smaller set across multiple patients, diseases, and time points.

1.3 Before You Get Started

Please make sure that you have the following packages installed before you proceed.

```
library(data.table)
library(ggplot2)
library(dplyr)
library(shiny)
```

Clone the repo into a directory. If you are using the GitHub desktop client, you will need to go to the webpage for the repo and select ">>Clone or Download >> Open in Desktop":

```
git clone https://github.com/laderast/DreamEDAShiny
```

When you have cloned the repo, open the `DreamEDAShiny.Rproj` in RStudio (Use **File >> Open Project** to open it.)

2.0 Examining the Rhinovirus Data

In this section, we'll examine the subsetted data in order to understand its format and learn some more about the `data.tables` package, which we'll use to store the data in memory. In case you are interested, there are a number of sections

2.1 Looking at the data

All of the data is in the `/data/twoStudies.RData` object. Let's look at the format of the data. Let's start by looking at the data objects in the `RData` file:

```
#load the data up
library(data.table)
load('data/twoStudies.RData')

#list the objects
ls()
```

```
## [1] "averageProfiles" "pathways"        "viralData"
```

Look at the `viralData` table. How big is the data set (how many rows)? In what column is the expression value? Take a look at the `dataDescription.pdf` in the main directory of the repo. What do the other columns correspond to? What are our outcome variables that we want to examine?

```
viralData
```

```
##           FEATUREID     value AGE GENDER TIMEHOURS   STUDYID SUBJECTID
##      1: 200000_s_at 10.504459  19 Female         0      SHAM HRV10-029
##      2: 200000_s_at 10.685820  21   Male       -24      SHAM HRV10-025
##      3: 200000_s_at 10.654310  24   Male       -24      SHAM HRV10-013
##      4: 200000_s_at 10.494799  21 Female       -24      SHAM HRV10-027
##      5: 200000_s_at 10.493724  24 Female       -24      SHAM HRV10-030
##     ---
## 304313:    55081_at  8.757976  20   Male        36 DEE3 H1N1      3021
## 304314:    55081_at  9.011929  20   Male         0 DEE3 H1N1      3021
## 304315:    55081_at  8.840402  27   Male         0 DEE3 H1N1      3024
## 304316:    55081_at  9.032293  26   Male         0 DEE3 H1N1      3023
## 304317:    55081_at  8.942934  19   Male         0 DEE3 H1N1      3022
##         SHEDDING_SC1 SYMPTOMATIC_SC2 geneSymbol
##      1:            0               1      PRPF8
##      2:            0               1      PRPF8
##      3:            0               0      PRPF8
##      4:            0               1      PRPF8
##      5:            0               1      PRPF8
##     ---
## 304313:            0               1    MICALL1
## 304314:            0               1    MICALL1
## 304315:            0               0    MICALL1
## 304316:            1               0    MICALL1
## 304317:            0               0    MICALL1
```

The `viralData` is in what's called `long` format, where each single observation (in this case, a microarray value for a probeset) is on its own line, accompanied by its metadata. This long format is what multiple packages, such as `dplyr` and `ggplot2` (which we use in this workshop) expect. We'll examine a method to `cast` (transform) the data into a `wide` matrix format later.

```
#look at the averaged profiles
averageProfiles
```

```
##        TIMEHOURS   STUDYID   geneSymbol  meanExpr     sdExpr
##     1:       -24      SHAM        PRPF8 10.625561 0.16838837
##     2:       -24      SHAM       CAPNS1 11.993976 0.12774703
##     3:       -24      SHAM      SLC25A3 12.574918 0.04656287
##     4:       -24      SHAM        ABCF1  9.541392 0.09070969
##     5:       -24      SHAM         DAD1 10.491738 0.11523139
##    ---
## 22568:        36 DEE3 H1N1       TBC1D2  7.583416 0.33751715
## 22569:        36 DEE3 H1N1      SLC27A3  9.452224 0.39332789
## 22570:        36 DEE3 H1N1       LRRC59  9.694646 0.20521245
## 22571:        36 DEE3 H1N1       ISYNA1  7.104442 0.25699441
## 22572:        36 DEE3 H1N1 C1QTNF9B-AS1  4.770560 0.20922887
```

How does the `averageProfiles` table differ from the `viralData` profile? What aspect of the data did we average over?

The last object we have is the `pathways` object.

2.2 An Introduction to the `data.table` package

We are using the `data.tables` package to store the data in memory. There are many advantages of a `data.table` versus using the regular `data.frame`:

1. Subsetting and doing aggregate calculations (such as you would use `tapply()` for) are very fast for large datasets. This is because `data.table` avoids many of the memory-copying problems of current base R functions.
2. The `fread()` function to load in data in delimited files is very fast, much faster than the base function `read.table()`.
3. Joining tables is very fast and efficient.

The main disadvantage of `data.table` is that the syntax is different than for `data.frames` or `dplyr`. It is just different enough to drive you crazy.

2.3 Keys For `data.table`

A `data.table` has a key, which you can set by using the `setkey()` function. This key is usually a column name, though you can also use multiple columns here. This key has two purposes: it provides an *index* to sort the `data.table`, and for joining with other tables, it provides the identifier to join on. Joins/merges with data.table are very quick.

```
#make a data.table version of iris data
data(iris)

irisDT <- data.table(iris)

setkey(irisDT, Sepal.Length)
irisDT
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
##   1:          4.3         3.0          1.1         0.1    setosa
##   2:          4.4         2.9          1.4         0.2    setosa
##   3:          4.4         3.0          1.3         0.2    setosa
##   4:          4.4         3.2          1.3         0.2    setosa
##   5:          4.5         2.3          1.3         0.3    setosa
##  ---
## 146:          7.7         3.8          6.7         2.2 virginica
## 147:          7.7         2.6          6.9         2.3 virginica
## 148:          7.7         2.8          6.7         2.0 virginica
## 149:          7.7         3.0          6.1         2.3 virginica
## 150:          7.9         3.8          6.4         2.0 virginica
```

2.4 Merges/Joins of Two `data.tables` (Optional)

Let's specify another table to join `irisDT` on.

```
#make a little table to merge
#initializing a data.table is identical to initializing a data.frame
testTable <- data.table(Species=c("setosa", "versicolor", "virginica"),
                        Likes=c(TRUE,FALSE,FALSE), Color = c("purple", "purple", "pink"))
#show the table
testTable
```

```
##       Species Likes  Color
## 1:     setosa  TRUE purple
## 2: versicolor FALSE purple
## 3:  virginica FALSE   pink
```

```
setkey(irisDT, Species)
irisDT
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
##   1:          4.3         3.0          1.1         0.1    setosa
##   2:          4.4         2.9          1.4         0.2    setosa
##   3:          4.4         3.0          1.3         0.2    setosa
##   4:          4.4         3.2          1.3         0.2    setosa
##   5:          4.5         2.3          1.3         0.3    setosa
##  ---
## 146:          7.7         3.8          6.7         2.2 virginica
## 147:          7.7         2.6          6.9         2.3 virginica
## 148:          7.7         2.8          6.7         2.0 virginica
## 149:          7.7         3.0          6.1         2.3 virginica
## 150:          7.9         3.8          6.4         2.0 virginica
```

```
setkey(testTable, Species)
```

Now that we have the keys set for both tables, we can join them together. In `data.table`, this is called a *rolling join*, and it is very fast, even for datasets with millions of rows.

```
#merge the two tables together using the keys we set (Species)
irisDT[testTable]
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width   Species Likes
##   1:          4.3         3.0          1.1         0.1    setosa  TRUE
##   2:          4.4         2.9          1.4         0.2    setosa  TRUE
##   3:          4.4         3.0          1.3         0.2    setosa  TRUE
##   4:          4.4         3.2          1.3         0.2    setosa  TRUE
##   5:          4.5         2.3          1.3         0.3    setosa  TRUE
##  ---
## 146:          7.7         3.8          6.7         2.2 virginica FALSE
## 147:          7.7         2.6          6.9         2.3 virginica FALSE
## 148:          7.7         2.8          6.7         2.0 virginica FALSE
## 149:          7.7         3.0          6.1         2.3 virginica FALSE
## 150:          7.9         3.8          6.4         2.0 virginica FALSE
##       Color
##   1: purple
##   2: purple
##   3: purple
##   4: purple
##   5: purple
##  ---
## 146:   pink
## 147:   pink
## 148:   pink
## 149:   pink
## 150:   pink
```

You may notice that the syntax here is completely different than `data.frame`. The way to read the above merge is that we want to merge the rows of `irisDT` and `testTable` given the keys that we have specified for each table.

2.5 Casting the long data into wide data (Optional)

If we wanted the data in a `wide` matrix format with each row belonging to a probeset/patient/study triplet, and each column corresponding to the value of the time series, we can transform the data using `dcast`. This function lets us *reshape* the data into another format.

The key to understanding `dcast` is understanding the formula interface for R, which works like this:

If we wanted

```
dcast(viralData, FEATUREID + SUBJECTID + STUDYID ~ TIMEHOURS, value.var= "value", fun.aggregate = mean)
```

```
##          FEATUREID SUBJECTID         STUDYID       -24         0         4
##     1: 200000_s_at HRV10-001            SHAM 10.591716 11.138535 10.042018
##     2: 200000_s_at HRV10-002            SHAM 10.962227 11.131125 10.893160
##     3: 200000_s_at HRV10-003 Rhinovirus Duke 10.550075 10.732898 10.772877
##     4: 200000_s_at HRV10-007 Rhinovirus Duke 10.488286       NaN 10.962661
##     5: 200000_s_at HRV10-008 Rhinovirus Duke 10.701332 10.968922  9.646992
##    ---
## 37277:    55081_at      3019       DEE3 H1N1  9.204657  8.746007       NaN
## 37278:    55081_at      3024       DEE3 H1N1  8.714051  8.840402       NaN
## 37279:    55081_at      3021       DEE3 H1N1  9.012449  9.011929       NaN
## 37280:    55081_at      3023       DEE3 H1N1  8.902532  9.032293       NaN
## 37281:    55081_at      3022       DEE3 H1N1  8.892839  8.942934       NaN
##               5        8        12       16       20     21.5        24
##     1:      NaN 10.30397 10.228630 10.80540 10.79489      NaN 10.597960
##     2:      NaN 10.94573 10.647014 11.02560 10.90027      NaN 11.013192
##     3:      NaN 10.36997 10.673494 10.64438 10.58817      NaN  9.808646
##     4:      NaN 10.88691 10.912892 10.44563 10.26850      NaN 10.291285
##     5:      NaN 10.37157 11.128713 10.59420 10.84475      NaN 10.780189
##    ---
## 37277: 9.063175      NaN  8.799034      NaN      NaN 9.172975       NaN
## 37278: 8.999460      NaN  8.828502      NaN      NaN 8.952315       NaN
## 37279: 8.795247      NaN  8.843854      NaN      NaN 8.799564       NaN
## 37280: 8.879908      NaN  9.009055      NaN      NaN 8.954986       NaN
## 37281: 8.841352      NaN  8.006422      NaN      NaN 8.808537       NaN
##              29        30        36
##     1:      NaN 11.028656 10.615470
##     2:      NaN 11.052908 10.957531
##     3:      NaN 10.496235 10.030312
##     4:      NaN  9.641070 10.931169
##     5:      NaN  9.532444 10.794055
##    ---
## 37277: 8.495089       NaN  8.900242
## 37278: 8.701077       NaN  8.576906
## 37279: 8.863944       NaN  8.757976
## 37280: 8.217440       NaN  8.950477
## 37281: 8.686102       NaN  8.769531
```

Subsetting Data

This operation is very fast, and is the main reason we're using `data.tables`. Let's look at the first entry in the `pathways` object.

```
#get the name of the first list item
pway <- names(pathways)[1]
pway
```

```
## [1] "Regulation of IFNA signaling"
```

```
path1 <- pathways[[pway]]
path1
```

```
## [1] "IFNA4"  "IL6"    "IFNA14" "IFNA16" "USP18"  "IFNA10"
```

Calculating New Values on Columns

3.0 Visualizing Gene Sets using Shiny

Now that we are familiar with the data format, let's start exploring the data set. Open the `global.R` file in the top folder and hit the "Run App" button in the top right corner of the script window to load the Shiny interface.

3.1 The

Discussion Time (Halfway Point)

What is your interpretation of the gene level versus pathway level?

What did you get and not get from examining the data?

What would you be interested in conditioning the plots on?

Clustering The Data