Non-tidy data

Jeff Leek 2016/02/17

During the discussion that followed the ggplot2 posts from David and I last week we started talking about tidy data and the man himself noted that matrices are often useful instead of "tidy data" and I mentioned there might be other data that are usefully "non tidy". Here I will be using tidy/non-tidy according to Hadley's definition. So tidy data have:

- One variable per column
- One observation per row
- Each type of observational unit forms a table

I push this approach in my guide to data sharing and in a lot of my personal work. But note that non-tidy data can definitely be already processed, cleaned, organized and ready to use.

> @hadleywickham @drob @mark_scheuerell I'm saying that not all data are usefully tidy (and not just matrices) so I care more abt flexibility
>
> — Jeff Leek (@jtleek) February 12, 2016

This led to a very specific blog request:

> @jtleek @drob I want a blog post on non-tidy data!
>
> — Hadley Wickham (@hadleywickham) February 12, 2016

So I thought I'd talk about a couple of reasons why data are usefully non-tidy. The basic reason is that I usually take a problem first, not solution backward approach to my scientific research. In other words, the goal is to solve a particular problem and the format I chose is the one that makes it most direct/easy to solve that problem, rather than one that is theoretically optimal.   To illustrate these points I'll use an example from my area.

Example data

Often you want data in a matrix format. One good example is gene expression data or data from another high-dimensional experiment. David talks about one such example in his post here. He makes the (valid) point that for students who aren't going to do genomics professionally, it may be more useful to learn an abstract tool such as tidy data/dplyr. But for those working in genomics, this can make you do unnecessary work in the name of theory/abstraction.

He analyzes the data in that post by first tidying the data.

```
library(dplyr)
library(tidyr)
library(stringr)
library(readr)
library(broom)

original_data %
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"), sep = "\\|\\|") %>%
  mutate_each(funs(trimws), name:systematic_name) %>%
  select(-number, -GID, -YORF, -GWEIGHT) %>%
  gather(sample, expression, G0.05:U0.3) %>%
  separate(sample, c("nutrient", "rate"), sep = 1, convert = TRUE)
```

It isn’t 100% tidy as data of different types are in the same data frame (gene expression and metadata/phenotype data belong in different tables). But its close enough for our purposes. Now suppose that you wanted to fit a model and test for association between the “rate” variable and gene expression for each gene. You can do this with David’s tidy data set, dplyr, and the broom package like so:

```
rate_coeffs = cleaned_data %>% group_by(name) %>%
     do(fit = lm(expression ~ rate + nutrient, data = .)) %>%
     tidy(fit) %>%
     dplyr::filter(term=="rate")
```

On my computer we get something like:

```
system.time( cleaned_data %>% group_by(name) %>%
+               do(fit = lm(expression ~ rate + nutrient, data = .)) %>%
+                tidy(fit) %>%
+                dplyr::filter(term=="rate"))
|==========================================================|100% ~0 s remaining
user  system elapsed
 12.431   0.258  12.364
```

Let’s now try that analysis a little bit differently. As a first step, lets store the data in two separate tables. A table of “phenotype information” and a matrix of “expression levels”. This is the more common format used for these type of data. Here is the code to do that:

```
expr = original_data %>%
  select(grep("[0:9]",names(original_data)))

rownames(expr) = original_data %>%
  separate(NAME, c("name", "BP", "MF", "systematic_name", "number"), sep = "\\|\\|") %>%
  select(systematic_name) %>% mutate_each(funs(trimws),systematic_name) %>% as.matrix()

vals = data.frame(vals=names(expr))
pdata = separate(vals,vals,c("nutrient", "rate"), sep = 1, convert = TRUE)

expr = as.matrix(expr)
```

If we leave the data in this format we can get the model fits and the p-values using some simple linear algebra

```
expr = as.matrix(expr)

mod = model.matrix(~ rate +  as.factor(nutrient),data=pdata)
rate_betas = expr %*% mod %*% solve(t(mod) %*% mod)
```

This gives the same answer after re-ordering

```
all(abs(rate_betas[,2]- rate_coeffs$estimate[ind]) < 1e-5,na.rm=T)
[1] TRUE
```

But this approach is much faster.

```
 system.time(expr %*% mod %*% solve(t(mod) %*% mod))
   user  system elapsed
  0.015   0.000   0.015
```

This requires some knowledge of linear algebra and isn't pretty. But it brings us to the first general point: **you might not use tidy data because some computations are more efficient if the data is in a different format.**

Many examples from graphical models, to genomics, to neuroimaging, to social sciences rely on some kind of linear algebra based computations (matrix multiplication, singular value decompositions, eigen decompositions, etc.) which are all optimized to work on matrices, not tidy data frames. There are ways to improve performance with tidy data for sure, but they would require an equal amount of custom code to take advantage of say C, or vectorization properties in R.

Ok now the linear regressions here are all treated independently, but it is very well known that you get much better performance in terms of the false positive/true positive tradeoff if you use an empirical Bayes approach for this calculation where you pool variances.

If the data are in this matrix format you can do it with R like so:

```
library(limma)
fit_limma = lmFit(expr,mod)
ebayes_limma = eBayes(fit_limma)
topTable(ebayes_limma)
```

This approach is again very fast, optimized for the calculations being performed and performs much better than the one-by-one regression approach. But it requires the data in matrix or expression set format. Which brings us to the second general point: ****you might not use tidy data because many functions require a different, also very clean and useful data format, and you don't want to have to constantly be switching back and forth.*** *Again, this requires you to be more specific to your application, but the potential payoffs can be really big as in the case of limma.

I'm showing an example here with expression sets and matrices, but in NLP the data are often input in the form of lists, in graphical analyses as matrices, in genomic analyses as GRanges lists, etc. etc. etc. One option would be to rewrite all infrastructure in your area of interest to accept tidy data formats but that would be going against conventions of a community and would ultimately cost you a lot of work when most of that work has already been done for you.

The final point, which I won't discuss here is that data are often usefully represented in a non-tidy way. Examples include the aforementioned GRanges list which consists of (potentially) ragged arrays of intervals and quantitative measurements about them. You could **force** these data to be tidy by the definition above, but again most of the infrastructure is built around a different format that is much more intuitive for that type of data. Similarly data from other applications may be more suited to application specific formats.

In summary, tidy data is a useful conceptual idea and is often the right way to go for general, small data sets, but may not be appropriate for all problems. Here are some examples of data formats (biased toward my area, but there are others) that have been widely adopted, have a ton of useful software, but don't meet the tidy data definition above. I will define these as "processed data" as opposed to "tidy data".

- Expression sets for expression data
- Summarized experiments for a variety of genomic experiments
- Granges Lists for genomic intervals
- Corpus objects for corpora of texts.
- igraph objects for graphs

I'm sure there are a ton more I'm missing and would be happy to get some suggestions on Twitter too.

When it comes to science - its the economy stupid.

Spreadsheets: The Original Analytics Dashboard