Win-Vector Blog

The Win-Vector LLC data science blog

R has some sharp corners

May 15, 2014 John Mount Opinion, Practical Data Science, Rants, Statistics
coding, corner cases, empty string, R

R is definitely our first choice go-to analysis system. In our opinion you really shouldn't use something else until you have an articulated reason (be it a need for larger data scale, different programming language, better data source integration, or something else). The advantages of R are numerous:

- Single integrated work environment.
- Powerful unified scripting/programming environment.
- Many many good tutorials and books available.
- Wide range of machine learning and statistical libraries.
- Very solid standard statistical libraries.
- Excellent graphing/plotting/visualization facilities (especially ggplot2).
- Schema oriented data frames allowing batch operations, plus simple row and column manipulation.
- Unified treatment of missing values (regardless of type).

For all that we always end up feeling just a little worried and a little guilty when introducing a new user to R. R is very powerful and often has more than one way to perform a common operation or represent a common data type. So you are never very far away from a strange and painful corner case. This why when you get R training you need to make sure you get an R expert (and not an R apologist). One of my favorite very smart experts is Norm Matloff (even his most recent talk title is smart: "What no one else will tell you about R"). Also, buy his book; we are very happy we purchased it.

But back to corner cases. For each method in R you really need to double check if it actually works over the common R base data types (numeric, integer, character, factor, and logical). Not all of them do and and sometimes you get a surprise.

Recent corner case problems we ran into include:

- randomForest regression fails on character arguments, but works on factors.
- mgcv `gam()` model doesn't convert strings to formulas.
- R maps can't use the empty string as a key (that is the string of length 0, not a `NULL` array or `NA` value).

These are all little things, but can be a pain to debug when you are in the middle of something else.

For our concrete example let's concentrate on the pain generated by the empty string.

In R strings represent free-form text and factors represent strings from a pre-defined finite set of possibility (actually called "levels"). The difference can be subtle and you may not always know which one you have (R may have converted for you) and which one will work (R may fail to convert for you).

Take for example the simple case of building a linear model mapping a string or factor valued x to numeric y-values:

```
d <- data.frame(y=c(1,2,3),x=c('a','b','c'))
lmModel <- lm(y~0+x,data=d)
d$lmPred <- predict(lmModel,newdata=d)

print(d)
  y x lmPred
1 1 a      1
2 2 b      2
3 3 c      3
```

We have used x to predict y. This works (under the covers) because R converts the string-values of x into factor levels and then uses. For the basic details of how this works try: `help('model.matrix')` or `help('contrasts')`. For a bit more on conversion to indicators (which is pretty much automatic in R, but can be a painful manual step in some other machine learning frameworks) see chapter 2 section 2.2.3 of Practical Data Science with R.

Of course it is silly of us to use the entire `lm()` framework to model an expected value conditioned on a single categorical variable. That is what `table()` and `aggregate()` are for. `lm()` gets the right answer but it has to do some unnecessary steps (such as forming and inverting a design matrix, which happens to be diagonal- but probably wastes

space in a non-sparse representation). To directly build a model that maps level to expected value we do the following:

```
mkMapModel <- function(yVarName,xVarName,data) {
  means <- aggregate(as.formula(paste(yVarName,xVarName,sep='~')),
     data=data,FUN=mean)
  model <- as.list(means[[yVarName]])
  names(model) <- means[[xVarName]]
  model
}
mapModel <- mkMapModel('y','x',d)
d$mapPred <- as.numeric(mapModel[d$x])

print(d)
  y x lmPred mapPred
1 1 a      1       1
2 2 b      2       2
3 3 c      3       3
```

This works great. It makes sense, and is *much* more efficient for variables that have a very large number of levels (see Modeling Trick: Impact Coding of Categorical Variables with Many Levels for more details). However it turns out this is only working because the x-variable is encoded as a factor. As the code below shows, when the x-variable takes on string values (called character in R) the `lm()` works (likely as it triggers a conversion to factor at some point), but our hand-rolled `mkMapModel()` fails:

```
d <- data.frame(y=c(1,2,3),x=c('a','b',''),stringsAsFactors=FALSE)
lmModel <- lm(y~0+x,data=d)
d$lmPred <- predict(lmModel,newdata=d)

print(d)
  y x lmPred
1 1 a      1
2 2 b      2
3 3        3
```

```
mapModel <- mkMapModel('y','x',d)
d$mapPred <- as.numeric(mapModel[d$x])

Error: (list) object cannot be coerced to type 'double'
```

This is because the empty string `""` (not null, just the string of length 0) isn't a legal map-key in R. Likely this is due R's linkages between evaluation environments and maps (and while the empty string may be a traditional string, it isn't a traditional

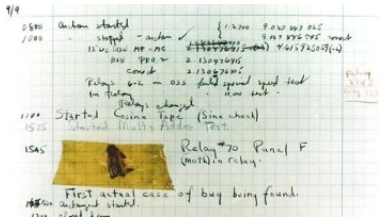
variable name; see the "environment->list coercion" example in `help('as.list')` for the connection).

One work around is to make sure the x-variable is a factor (not a character array). We demonstrate this in the working code below. Another fix would be to use `paste()` to prefix all strings (so none are empty).

```
d <- data.frame(y=c(1,2,3),x=c('a','b',''),stringsAsFactors=TRUE)
mapModel <- mkMapModel('y','x',d)
d$mapPred <- as.numeric(mapModel[d$x])

print(d)
  y x mapPred
1 1 a       1
2 2 b       2
3 3         3
```

The requirement to ensure strings are already converted to factors is not unique to our function `mkMapModel()` (for example the randomForest package has similar issues in some circumstances, but at least is does work with factors unlike some Python scikit-learn packages). Even with such issues I think you are net-ahead using R (notice we didn't have to write any for-loops due to the vectorized nature of the operator `[]`). With some defensive coding you certainly are ahead in using R's built in models (like `lm()`).



May 15, 2014 John Mount Opinion, Practical Data Science, Rants, Statistics coding, corner cases, empty string, R

2 thoughts on “R has some sharp corners”

Fr.

May 16, 2014 at 7:40 am

Stata probably suffers from similar strings/factors issues on many commands.

It’s true that avoiding these oddities can take up a lot of scripting time, but from a data perspective, they force you to know exactly what you feed your models with ~ which is great, especially when you are building the dataset yourself prior to modeling.

So strangely enough, having to care more about data structure is something that I appreciated when moving to R for data analysis, as opposed to more straightforward stats with Stata.

jmount

May 16, 2014 at 7:57 am

Thanks, @Fr. I agree. In R the good greatly outweighs the bad. Since you have a good scripting language, once you are aware of the problem you can script re-usable fixes. Getting into the habit of knowing your data and checking results is a good thing. And finally R overall does a good job with factors (certainly a much better job than a system that doesn’t have them or one that doesn’t encode indicators/contrasts for you automatically).

Comments are closed.