`gather`, factors and logistic regression

Ken Butler

February 2, 2016

1 `gather`

`gather` is a handy tool from the `tidyr` package that is used for gathering up columns of a data frame that are not variables in themselves but levels of a factor. This is an example:

```
d=data.frame(x=1:5,early=c(10,11,11,13,15),late=c(12,12,11,14,16))
d
```

```
##   x early late
## 1 1    10   12
## 2 2    11   12
## 3 3    11   11
## 4 4    13   14
## 5 5    15   16
```

Suppose `early` and `late` are a response variable `y` that is measured early and late in the course of the experiment. Then it would be better to have all the values in one column `y` with a label `timepoint` that says whether they were measured early or late. This is done with `gather` like this:

```
library(tidyr)
d2=gather(d,timepoint,y,early:late)
d2
```

```
##    x timepoint  y
## 1  1     early 10
## 2  2     early 11
## 3  3     early 11
## 4  4     early 13
## 5  5     early 15
## 6  1      late 12
## 7  2      late 12
## 8  3      late 11
## 9  4      late 14
## 10 5      late 16
```

This is all fine and well. We'd expect `timepoint` to be a factor since it is a categorical variable that distinguishes the timepoints in the experiment where the measurements were taken. Is it?

```
str(d2)

## 'data.frame': 10 obs. of  3 variables:
##  $ x        : int  1 2 3 4 5 1 2 3 4 5
##  $ timepoint: chr  "early" "early" "early" "early" ...
##  $ y        : num  10 11 11 13 15 12 12 11 14 16
```

No, it is `chr` or "text". I find this odd, but I know that Hadley Wickham, who wrote `tidyr`, has a thing about factors and likes things to be text.[1] *I* like factors to be factors. Sometimes it matters what kind of variable you have and sometimes it doesn't. For example, what if we want the mean value of `y` at each timepoint?

```
aggregate(y~timepoint,d2,mean)

##   timepoint  y
## 1     early 12
## 2      late 13
```

That works.

2 Logistic regression

What about if we do a (stupid[2]) logistic regression to predict timepoint from `x` and `y`?

```
timepoint.1=glm(timepoint~x+y,d2,family="binomial")

## Error in eval(expr, envir, enclos):  y values must be 0 <= y <=
1
```

That doesn't seem to work (and gives an arcane error). What R means by `y` is the response variable (the help for `glm` calls that `y`); it's trying to infer what we meant, and the best it can do is to guess that we were trying to have a numeric response variable where 1 is success and 0 is failure. That this wasn't what we were trying to do isn't R's fault, really.

Fortunately, we can make `gather` produce a factor instead. I just discovered this today. Here's how it's done:

[1] To be fairer to his point of view, I think he prefers that we should *deliberately create* factors when we need them, and not have factors pop out of functions unexpectedly.

[2] We might have been asking a different question: what is it about x and y that makes the timepoints different? This is what discriminant analysis does, which we see later.

```
d2=gather(d,timepoint,y,early:late,factor_key=T)
str(d2)

## 'data.frame': 10 obs. of  3 variables:
##  $ x        : int  1 2 3 4 5 1 2 3 4 5
##  $ timepoint: Factor w/ 2 levels "early","late": 1 1 1 1 1 2 2 2 2 2
##  $ y        : num  10 11 11 13 15 12 12 11 14 16
```

Indeed, `timepoint` is now a factor. Does a logistic regression now work?

```
timepoint.1=glm(timepoint~x+y,d2,family="binomial")
summary(timepoint.1)

##
## Call:
## glm(formula = timepoint ~ x + y, family = "binomial", data = d2)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.30584  -0.96714  -0.03815   0.81619   1.97498
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -10.9416     7.9504  -1.376    0.169
## x            -1.3547     1.1369  -1.192    0.233
## y             1.2008     0.8655   1.387    0.165
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 13.863  on 9  degrees of freedom
## Residual deviance: 11.184  on 7  degrees of freedom
## AIC: 17.184
##
## Number of Fisher Scoring iterations: 4
```

This now works, showing that `glm` (with `family="binomial"`) needs a genuine factor for a response variable.

3 Ordinal logistic regression

Let's do one like the coal-miners example from class. I'll make up some data in similar format, with explanatory variable `exposure` and a three-category response `low`, `medium`, `high` and frequencies in the cells:

```
ord.wide=data.frame(exposure=1:4,low=c(10,9,7,5),med=c(0,1,1,2),
  high=c(3,4,6,8))
ord.wide
```

```
##   exposure low med high
## 1        1  10   0    3
## 2        2   9   1    4
## 3        3   7   1    6
## 4        4   5   2    8
```

This is also untidy since we need all the frequencies in one column ("low", "med", "high" are not variable names but levels of a factor with a name like "severity"). The remedy is as above:

```
ord=gather(ord.wide,severity,frequency,low:high)
ord
```

```
##    exposure severity frequency
## 1         1      low        10
## 2         2      low         9
## 3         3      low         7
## 4         4      low         5
## 5         1      med         0
## 6         2      med         1
## 7         3      med         1
## 8         4      med         2
## 9         1     high         3
## 10        2     high         4
## 11        3     high         6
## 12        4     high         8
```

That variable `severity` is text, not a factor:

```
str(ord)
```

```
## 'data.frame': 12 obs. of  3 variables:
##  $ exposure : int  1 2 3 4 1 2 3 4 1 2 ...
##  $ severity : chr  "low" "low" "low" "low" ...
##  $ frequency: num  10 9 7 5 0 1 1 2 3 4 ...
```

Does that matter, when we come to predict severity from exposure?

```
library(MASS)
severity.1=polr(severity~exposure,data=ord,weights=frequency)
```

```
## Error in polr(severity ~ exposure, data = ord, weights = frequency):
response must be a factor
```

Yes, it does matter, and this time, we got a nice clear error message telling us so. So we'll use the variant of `gather` that we saw just now:

```
ord=gather(ord.wide,severity,frequency,low:high,factor_key=T)
str(ord)

## 'data.frame': 12 obs. of  3 variables:
##  $ exposure : int  1 2 3 4 1 2 3 4 1 2 ...
##  $ severity : Factor w/ 3 levels "low","med","high": 1 1 1 1 2 2 2 2 3 3 ...
##  $ frequency: num  10 9 7 5 0 1 1 2 3 4 ...
```

Look carefully at the levels listed for `severity`: they are low, medium, high *in that order*. That is the order we want, and not the alphabetical order that we might have gotten. What `gather` does, when run this way, is to create a factor *whose levels are in the order that they appear in the data frame*. So if that's what you want, as it is here, this is the way to make it happen.

So now the ordinal logistic regression should run with no problems:

```
severity.1=polr(severity~exposure,data=ord,weights=frequency)
```

and we can test for an effect of exposure thus, with the first line saying "take `exposure` out of the previous model":

```
severity.0=update(severity.1,.~.-exposure)
anova(severity.0,severity.1)

## Likelihood ratio tests of ordinal regression models
##
## Response: severity
##      Model Resid. df Resid. Dev   Test    Df LR stat.    Pr(Chi)
## 1        1        54   98.97189
## 2 exposure        53   93.65444 1 vs 2     1 5.317443 0.02111297
```

Exposure has a significant but not strongly significant effect, probably because there wasn't all that much (made-up) data.

4 Multinomial logistic regression

Our model for this one is the alligator question on the assignment. Let's think about favourite foods by age (presumably of humans), with made-up data like this, and the extra columns being frequencies again:

```
food=data.frame(age=c(12,14,16,21,22),meat=c(1,2,4,4,5),
  fish=c(0,0,1,1,1),ketchup=c(20,12,5,2,1),
  chicken.nuggets=c(10,8,2,1,1))
food
```

```
##   age meat fish ketchup chicken.nuggets
## 1  12    1    0      20              10
## 2  14    2    0      12               8
## 3  16    4    1       5               2
## 4  21    4    1       2               1
## 5  22    5    1       1               1
```

The last four columns are favourite foods and frequency of each, so we'll do the same `gather` thing that we did last time:

```
food2=gather(food,favourite,frequency,meat:chicken.nuggets)
food2
```

```
##    age       favourite frequency
## 1   12            meat         1
## 2   14            meat         2
## 3   16            meat         4
## 4   21            meat         4
## 5   22            meat         5
## 6   12            fish         0
## 7   14            fish         0
## 8   16            fish         1
## 9   21            fish         1
## 10  22            fish         1
## 11  12         ketchup        20
## 12  14         ketchup        12
## 13  16         ketchup         5
## 14  21         ketchup         2
## 15  22         ketchup         1
## 16  12 chicken.nuggets        10
## 17  14 chicken.nuggets         8
## 18  16 chicken.nuggets         2
## 19  21 chicken.nuggets         1
## 20  22 chicken.nuggets         1
```

Since we didn't say anything about `factor_key`, `favourite` is presumably text:

```
str(food2)
```

```
## 'data.frame': 20 obs. of  3 variables:
##  $ age      : num  12 14 16 21 22 12 14 16 21 22 ...
##  $ favourite: chr  "meat" "meat" "meat" "meat" ...
##  $ frequency: num  1 2 4 4 5 0 0 1 1 1 ...
```

Does that matter, as far as `multinom` is concerned? Let's see:

```
library(nnet)
favourite.1=multinom(favourite~age,data=food2,weight=frequency)

## # weights:  12 (6 variable)
## initial  value 112.289843
## iter  10 value 79.096874
## iter  20 value 78.929272
## iter  20 value 78.929272
## iter  20 value 78.929272
## final  value 78.929272
## converged
```

That seems to have worked, this time. Let's see if some predictions look reasonable:

```
ages=c(12,15,18,21)
ages

## [1] 12 15 18 21

new=expand.grid(age=ages)
new

##   age
## 1  12
## 2  15
## 3  18
## 4  21
```

I didn't really need to use `expand.grid` this time, since you can't really get "all combinations" of only one variable, but the advantage of doing it like this is that I can go my standard way without thinking too much. Then:

```
pp=predict(favourite.1,new,type="p")
cbind(new,pp)

##   age chicken.nuggets        fish   ketchup       meat
## 1  12       0.3247425 0.005267543 0.6183078 0.05168222
## 2  15       0.3003234 0.019592473 0.5311357 0.14894845
## 3  18       0.2246841 0.058952666 0.3690960 0.34726731
## 4  21       0.1190804 0.125661533 0.1817012 0.57355690
```

You see that the probability of favourite food being chicken nuggets or ketchup decreases sharply with age, while that of fish or meat goes up with age. (I made up the data to illustrate precisely this point, so it wasn't exactly a surprise to me.)

5 Conclusions

I think there are two major conclusions to come from this:

1. `gather`, when it gathers up columns, creates a "key" column, saying which original column each observation came from. By default, *that column is text, not a factor*. But you can make it be a factor by adding `factor_key=T` to your `gather`. In the examples we have seen here, it is *always safe* to do this, but sometimes unnecessary.[3]

2. Both the `glm` and `polr` versions of logistic regression *require* a factor response variable. For `multinom`, the response variable can be a `chr` (text) variable, but it could be a factor as well.

My advice in these situations is currently this (but subject to change as I learn more): when you do a `gather`, think about what you are going to use the new data frame for. Typically, you are going to be doing some kind of modelling. If you are likely to need the "what makes them different" column to be a factor, rather than just text, put the `factor_key=T` in your `gather`. That would apply in all three cases here: we're going to use it as a response[4] variable, so we can play it safe and make it into a factor.[5]

[3] In my opinion, the default output of `gather` should be a factor, but that's not the way it is, so we have to live with it.

[4] The same applies if it's going to be a factor explanatory variable in a regression. A text variable won't work here either.

[5] And thus ignoring the fact that `multinom` doesn't need a factor to work properly, for the sake of a uniform treatment.