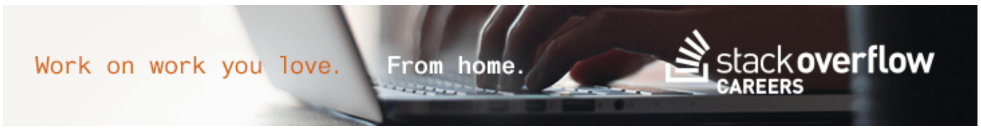
Converting coefficient names to a formula in R



When using formulas that have factors, the fitted models name the coefficients XY, where X is the name of the factor and Y is a particular level of it. I want to be able to create a formula from the names of these coefficients.

The reason: If I fit a lasso to a sparse design matrix (as I do below) I would like to create a new formula object that only contains terms for the nonzero coefficients.

```
require("MatrixModels")
require("glmnet")
set.seed(1)
n <- 200
Z <- data.frame(letter=factor(sample(letters,n,replace=T),letters),
                x=sample(1:20,200,replace=T))
f <- ~ letter + x:letter + I(x>5):letter
X <- sparse.model.matrix(f, Z)
beta <- matrix(rnorm(dim(X)[2],0,5),dim(X)[2],1)
y <- X %*% beta + rnorm(n)

myfit <- glmnet(X,as.vector(y),lambda=.05)
fnew <- rownames(myfit$beta)[which(myfit$beta != 0)]
 [1] "letterb"              "letterc"              "lettere"
 [4] "letterf"              "letterg"              "letterh"
 [7] "letterj"              "letterm"              "lettern"
[10] "lettero"              "letterp"              "letterr"
[13] "letters"              "lettert"              "letteru"
[16] "letterw"              "lettery"              "letterz"
[19] "lettera:x"            "letterb:x"            "letterc:x"
[22] "letterd:x"            "lettere:x"            "letterf:x"
[25] "letterg:x"            "letterh:x"            "letteri:x"
[28] "letterj:x"            "letterk:x"            "letterl:x"
[31] "letterm:x"            "lettern:x"            "lettero:x"
[34] "letterp:x"            "letterq:x"            "letterr:x"
[37] "letters:x"            "lettert:x"            "letteru:x"
[40] "letterv:x"            "letterw:x"            "letterx:x"
[43] "lettery:x"            "letterz:x"            "letterb:I(x > 5)TRUE"
[46] "letterc:I(x > 5)TRUE" "letterd:I(x > 5)TRUE" "lettere:I(x > 5)TRUE"
[49] "letteri:I(x > 5)TRUE" "letterj:I(x > 5)TRUE" "letterl:I(x > 5)TRUE"
[52] "letterm:I(x > 5)TRUE" "letterp:I(x > 5)TRUE" "letterq:I(x > 5)TRUE"
[55] "letterr:I(x > 5)TRUE" "letteru:I(x > 5)TRUE" "letterv:I(x > 5)TRUE"
[58] "letterx:I(x > 5)TRUE" "lettery:I(x > 5)TRUE" "letterz:I(x > 5)TRUE"
```

From this I would like to have a formula

```
~ I(letter=="d") + I(letter=="e") + ...(etc)
```

I checked out formula() and all.vars() to no avail. Also, writing a function to parse this is a bit of a pain because of the different types of terms that can arise. For example, for x:letter when x is a numeric value and letter is a factor, or I(x>5):letter as another annoying case.

So am I not aware of some function to convert between formula and its character representation and back again?

r formula sparse-matrix

edited Nov 26 '10 at 9:44
Gavin Simpson
94.2k 9 168 276

asked Nov 25 '10 at 21:37
Christopher DuBois
11.1k 17 41 85

That isn't a formula I recognise in R – Gavin Simpson Nov 25 '10 at 22:00

1 Perhaps I misunderstand, but you appear to not fully grep R's model formulae. You don't include in the formula the XY bits, you include the X and `model.matrix()` and `model.frame()` do their thing to expand the levels of the X to the relevant model matrix columns, the XY. – Gavin Simpson Nov 25 '10 at 22:06

Could you explain why you want the formula? What is the end use? – Gavin Simpson Nov 25 '10 at 22:14

I have a design matrix with 200k columns created with model f. After fitting the lasso, I have a model g with, say, 5k terms. I would like to build new design matrix with g rather than f. After letting model.matrix() and model.frame() do their thing, I would like to get a formula corresponding to some subset of the model matrix columns (after the levels have been expanded) – Christopher DuBois Nov 26 '10 at 3:14

2 Answers

When I ran the code, I got something a bit different, since set.seed() had not been specified. Instead of using the variable name "letter", I used "letter_" as a convenient splitting marker:

```
> fnew <- rownames(myfit$beta)[which(myfit$beta != 0)]

> fnew
 [1] "letter_c" "letter_d" "letter_e" "letter_f" "letter_h" "letter_k" "letter_l"
 [8] "letter_o" "letter_q" "letter_r" "letter_s" "letter_t" "letter_u" "letter_v"
[15] "letter_w"
```

Then made the split and packaged into a character matrix:

```
> fnewmtx <- cbind( lapply(sapply(fnew, strsplit, split="_"), "[[", 2),
+ lapply(sapply(fnew, strsplit, split="_"), "[[", 1))
```

> fnewmtx [,1] [,2]
> letter_c "c" "letter" letter_d "d" "letter" letter_e "e" "letter" letter_f "f" "letter" snipped the rest

And wrapped the paste function(s) output in as.formula() which is half of the answer to how to "convert between formula and its character representation and back." The other half is as.character()

```
form <- as.formula( paste("~",
             paste(
               paste(" I(", fnewmtx[,2], "_ ==", "'",fnewmtx[,1],"') ", sep="") ,
             sep="", collapse="+")
                 )
           )  # edit: needed to add back the underscore
```

And the output is now an appropriate class object:

```
> class(form)
[1] "formula"
> form
~I(letter_ == "c") + I(letter_ == "d") + I(letter_ == "e") +
    I(letter_ == "f") + I(letter_ == "h") + I(letter_ == "k") +
    I(letter_ == "l") + I(letter_ == "o") + I(letter_ == "q") +
    I(letter_ == "r") + I(letter_ == "s") + I(letter_ == "t") +
    I(letter_ == "u") + I(letter_ == "v") + I(letter_ == "w")
```

I find it interesting that the as.formula conversion made the single-quotes around the letters into double-quotes.

Edit: Now that the problem has an additional dimension or two, my suggestion is to skip the recreation of the formula. Note that the rownames of myfit$beta are exactly the same as the column names of X, so instead use the non-zero rownames as indices to select columns in the X matrix:

```
> str(X[ , which( colnames(X) %in% rownames(myfit$beta)[which(myfit$beta != 0)] )] )
Formal class 'dgCMatrix' [package "Matrix"] with 6 slots
  ..@ i       : int [1:429] 9 54 91 157 166 37 55 68 117 131 ...
  ..@ p       : int [1:61] 0 5 13 20 28 36 42 50 60 68 ...
  ..@ Dim     : int [1:2] 200 60
  ..@ Dimnames:List of 2
  .. ..$ : chr [1:200] "1" "2" "3" "4" ...
  .. ..$ : chr [1:60] "letter_b" "letter_c" "letter_e" "letter_f" ...
  ..@ x       : num [1:429] 1 1 1 1 1 1 1 1 1 1 ...
  ..@ factors : list()

> myfit2 <- glmnet(X[ , which( colnames(X) %in% rownames(myfit$beta)[which(myfit$beta !=
0)] )] ,as.vector(y),lambda=.05)
> myfit2

Call:  glmnet(x = X[, which(colnames(X) %in% rownames(myfit$beta)[
                                           which(myfit$beta != 0)])],
              y = as.vector(y), lambda = 0.05)

     Df   %Dev Lambda
[1,] 60 0.9996   0.05
```

edited Nov 26 '10 at 5:03 answered Nov 25 '10 at 22:57
42-
148k 5 117 233

Thanks DWin! This is close, but not quite a general solution. I've updated my example. Is there a quick fix that extends your solution? – Christopher DuBois Nov 25 '10 at 23:10

Also, when creating fnewmtx, shouldn't the strsplit use split="_" instead of split=""? – Christopher DuBois Nov 25 '10 at 23:18

Quite right. I will edit. I am seeing if I can cope with the added complexity of an interaction formula but don't expect a quick reply on that..... update: tried to edit but the underscores are in the code segments of the original and not being displayed on the SO page. Re-"coded" and perhaps now better. – 42- Nov 25 '10 at 23:48

1 @Christopher Not sure to which X you are referring above. DWin's solution is indexing your **sparse** model matrix `X`. From your Question, I got the distinct impression you *could* form that matrix in order to fit the model with `glmnet()`. DWin's subsetting yields a new *sparse* matrix containing only the columns where the coefs were non-zero. Do you want to create the full design matrix (as opposed to the sparse version)

from this reduced *sparse* design matrix? If so, try: `newX <- as.matrix(X[ , which( colnames(X) %in% rownames(myfit$beta)[which(myfit$beta != 0)] )])` . `newX` has 60 cols. – Gavin Simpson Nov 26 '10 at 9:42

1 Sorry for not being clear. Let's say I create my first sparse model matrix X_train and fit it with glmnet(). (This takes 5-15 minutes for my particular problem.) Now I want to create a new sparse matrix using new data, X_test, but only with the columns whose coefficients were nonzero. For computational reasons, I want to avoid building up a sparse matrix for X_test that has all the columns, so the subseting solution above doesn't work for me (unless I'm missing something). – Christopher DuBois Nov 27 '10 at 2:55

Christopher, what you are asking for appears, after some consideration and examination of `sparse.model.matrix` etc, to be somewhat involved. You haven't explain *why* you do not want to form the full sparse model matrix for `X_test` so it is difficult to advise a way forward other than the two options below.

If you have a large number of observations in `X_test` and hence do not want to produce the full sparse matrix for use in `predict()` for computational reasons, it might be more expedient to split `X_test` into two or more chunks of samples and form the sparse model matrices for each one in turn, discarding it after after use.

Failing that, you will need to study code from the Matrix package in detail. Start with `sparse.model.matrix` and note that it then calls `Matrix:::model.spmatrix` and locate calls to `Matrix:::fac2Sparse` in that function. You will probably need to co-opt code from these functions but use a modified `fac2Sparse` to achieve what you want to achieve.

Sorry I cannot provide an off-the-shelf script to do this, but that is a substantial coding task. If you go down that route, check out the *Sparse Model Matrices* vignette in the Matrix package and get the package sources (from CRAN) to see if the functions I mention are better documented in the source code (there are no Rd files for `fac2Sparse` for example). You can also ask the authors of Matrix (Martin Maechler and Doug Bates) for advice, although note that both of these chaps have had a particularly heavy teaching load this term.

Good luck!

edited Dec 3 '10 at 12:05

answered Dec 1 '10 at 17:10
Gavin Simpson
94.2k 9 168 276