Design Matrices in R

WILD 502 - Jay Rotella

The work we'll do with design matrices is also very relevant to statistical analyses you do in other statistical software. In R, 'model.matrix' is a useful tool for seeing the design matrices that are in play when you build regression models.

Build a simple data frame

First, build a simple data frame with *time* as a factor and *Time* as a continuous, numeric variable. The two variables look alike when you print the data frame. But, if you summarize the data, you see that they are different.

```
d <- data.frame(time = factor(1:4), Time = 1:4)
d
```

```
##   time Time
## 1    1    1
## 2    2    2
## 3    3    3
## 4    4    4
```

```
summary(d)
```

```
##  time       Time
##  1:1   Min.   :1.00
##  2:1   1st Qu.:1.75
##  3:1   Median :2.50
##  4:1   Mean   :2.50
##        3rd Qu.:3.25
##        Max.   :4.00
```

Design matrices: different types

First, let's look at *treatment contrasts*, which are the default type in R where the reference level (control) is the first level of the factor variable, which is the level with the lowest number or letter among the values provided. In Program MARK, you can set the reference level to be for any level of the factor you wish, e.g., you might have a specific year or group or site that you think of as a reference level. Note that the $\hat{\beta}'s$ associated with the other columns (time2, time3, time4) are then intercept-adjustment terms.

```
model.matrix(~time, data = d)
```

```
##   (Intercept) time2 time3 time4
## 1           1     0     0     0
## 2           1     1     0     0
## 3           1     0     1     0
## 4           1     0     0     1
## attr(,"assign")
## [1] 0 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$time
## [1] "contr.treatment"
```

We obtain an identity matrix if we remove the intercept from the model. Now each of $\hat{\beta}'s$ operates alone to provide the estimate for each time period. **Note:** be aware that while removing the intercept makes sense for a model that only includes factor variables, it should not typically be done if continuous variables are in the model (see below).

```
model.matrix(~-1 + time, data = d)
```

```
##   time1 time2 time3 time4
## 1     1     0     0     0
## 2     0     1     0     0
## 3     0     0     1     0
## 4     0     0     0     1
## attr(,"assign")
## [1] 1 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$time
## [1] "contr.treatment"
```

We can also request *sums contrasts*, for which the intercept provides an average and the other $\hat{\beta}'s$ represent adjustments away from that average for each time period. These are quite useful if you want to know the overall average and how far each time period's value is away from that average.

```
model.matrix(~time, data = d, contrasts.arg = list(time = "contr.sum"))
```

```
##   (Intercept) time1 time2 time3
## 1           1     1     0     0
## 2           1     0     1     0
## 3           1     0     0     1
## 4           1    -1    -1    -1
## attr(,"assign")
## [1] 0 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$time
## [1] "contr.sum"
```

Note: All three of the design matrices above would yield the exact same estimates of survival rates if applied to the same dataset. They would, of course, yield different estimates of the β's.

We can also examine a design matrix with continuous covariates

```
model.matrix(~Time, data = d)
```

```
##   (Intercept) Time
## 1           1    1
## 2           1    2
## 3           1    3
## 4           1    4
## attr(,"assign")
## [1] 0 1
```

Consider what it means to run a model with a continous covariate without an intercept. You've now declared that you don't want to estimate an intercept, which means that the intercept = 0. If the design matrix below is for a logistic regression model of survival rate, you are declaring that when Time = 0, the survival rate is 0.5: $\frac{exp(0+\beta_1 \cdot 0)}{(1+exp(0+\beta_1 \cdot 0))}$, which simplifies to $\frac{exp(0)}{(1+exp(0))}$ or $\frac{1}{2} = 0.5$. This will affect the estimates of all the $\beta's$ as they are affected by what the intercept value is. It is very hard to imagine a situation where you would want to do that!

```
model.matrix(~-1 + Time, data = d)
```

```
##   Time
## 1    1
## 2    2
## 3    3
## 4    4
## attr(,"assign")
## [1] 1
```