Lsn 20_AY_23

Clark

Admin

Recall that the general form for a linear regresison model is:

We can think about this model as having two components, a linear predictor $\beta_0 + \beta_1 x_{1,i} + \beta 2 x_{2,i} + \cdots + \beta n x_{n,i}$ and a random mechanism that perturbs us from the plane, ϵ_i.

However, sometimes, we might believe that the relationship between our covariates and response variable is not linear. This can be done for a few different reasons, perhaps the best reason is if we have some previous knowledge that suggests x and y have a nonlinear relationship. For example, if we remember the differential equation for radio active decay we had:

$$Y'(t) = -kY(t)$$

Which yielded a solution of:

$$Y(t) = Ce^{-kt}$$

Where C depended on the initial conditions, say $C = 3$ was the initial amount of substance. Then we had a non linear relationship between Y and t. If we measured time and amount of radioactive subtstance we would have to account for measurement error and perhaps we could fit the statistical model:

Therefore it might make sense to talk about a general form for statistical models of:

This model has two components, signal, and noise. While the best case is we are using a scientific mechanism to define the form of $f(x_i)$, in other cases we might just observe that clearly x_i and y_i don't have a linear relationship, so we might explore other forms of $f(x_i)$.

The simplest function outside of a linear relationship is if we assume $f(x_i) = \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{1,i}^2$. Or $x_{1,i}$ and y_i have a quadratic relationship. This is, what our text calls, a **polynomial statistical model**. While the relationship between $x_{1,i}$ and y_i is non-linear here, fitting the model can be achieved in the exact same way as a linear regression model. To see this, let's consider the Kentucky Derby data

```
ky.dat<-read.table("http://www.isi-stats.com/isi2/data/KYDerby18.txt",header=T,stringsAsFactors = T)
ky.dat %>% ggplot(aes(x=Year,y=Time))+geom_point()
```

That's kinda weird… But as it turns out, the distance changed in 1896, so we're comparing apples to oranges. Let's look at speed vs year

```
ky.dat %>% ggplot(aes(x=Year,y=speed))+geom_point()
```

Is there a story to the data? Unusual observations?

This isn't uncommon in athletic performace. We might think about there being a cap on the fastest a horse can possibly run. So it might make sense to fit a quadratic model. The model will be:

We can fit this by:

```
ky.dat <- ky.dat %>% mutate(Year.sq=Year^2)
poly.lm<-lm(speed~Year+Year.sq,data=ky.dat)
summary(poly.lm)
```

```
##
## Call:
## lm(formula = speed ~ Year + Year.sq, data = ky.dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9131 -0.3167  0.0314  0.3986  1.1499
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -9.888e+02  1.230e+02  -8.036 3.37e-13 ***
## Year         1.030e+00  1.265e-01   8.146 1.81e-13 ***
## Year.sq     -2.587e-04  3.249e-05  -7.964 5.02e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6024 on 141 degrees of freedom
## Multiple R-squared:  0.7522, Adjusted R-squared:  0.7487
## F-statistic:   214 on 2 and 141 DF,  p-value: < 2.2e-16
```

To check the fit we can look at:

```
ky.dat %>% ggplot(aes(x=Year,y=speed))+geom_point()+
  geom_line(aes(x=Year,y=.fitted),data=poly.lm,lwd=2,color="red")
```

Fit looks decent.

To check assumption on ϵ_i we have:

```
poly.lm %>% ggplot(aes(x=.fitted,y=.resid))+geom_point()
```

Any concerns?

The fitted or predicted model is:

If we want to predict from this model, we could do:

```
predict(poly.lm,data.frame(Year=2019,Year.sq=2019^2),interval="prediction")
```

```
##        fit      lwr      upr
## 1 36.65162 35.42299 37.88026
```

Let's look at this:

```
pred.df<-data.frame(y1=35.4,y2=37.8,x1=2019,x2=2019)
ky.dat %>% ggplot(aes(x=Year,y=speed))+geom_point()+
  geom_line(aes(x=Year,y=.fitted),data=poly.lm,lwd=2,color="red")+
  geom_segment(aes(x = x1, y = y1, xend = x2, yend = y2),lwd=2,colour = "blue", data = pred.df)
```

One concern we might have is colinearity or a relationship between our predictors. To examine this we again look at a pairs plot:

```
sub.df<-data.frame(speed=ky.dat$speed,Year=ky.dat$Year,Year.sq=ky.dat$Year.sq)
ggpairs(sub.df)
```

To fix this issue we can use what are called Orthogonal Polynomials. The scope of this is a bit beyond the course, but how this works is, our intercept is equal to 1, our first polynomial is equal to $x \equiv x - \bar{x}$.

The second polynomial is found via recursion:

$$P_2(x) = (x - \frac{< x^2, x >}{< x, x >})x - \frac{< x, x >}{n}$$

Each of the polynomial terms are then scaled by their l2 norm making them orthonormal polynomials.

Once this is done, these new covariates retain the polynomial shape of the raw polynomials, but are now uncorrelated with each other. This is done in R using `poly()`

```
orth.lm<-lm(speed~poly(Year,2),data=ky.dat)
summary(orth.lm)
```

```
##
## Call:
## lm(formula = speed ~ poly(Year, 2), data = ky.dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.9131 -0.3167  0.0314  0.3986  1.1499
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     35.8926     0.0502 714.959  < 2e-16 ***
## poly(Year, 2)1  11.5029     0.6024  19.094  < 2e-16 ***
## poly(Year, 2)2  -4.7979     0.6024  -7.964 5.02e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6024 on 141 degrees of freedom
## Multiple R-squared:  0.7522, Adjusted R-squared:  0.7487
## F-statistic:   214 on 2 and 141 DF,  p-value: < 2.2e-16
```

```
ky.dat %>% ggplot(aes(x=Year,y=speed))+geom_point()+
    geom_line(aes(x=ky.dat$Year,y=.fitted),data=orth.lm,lwd=2,color="red")
```

Fit is the exact same

```
predict(orth.lm,data.frame(Year=2019,Year.sq=2019^2),interval="prediction")
```

```
##        fit      lwr      upr
## 1 36.65162 35.42299 37.88026
```

Prediction is the same

```
new.df<-data.frame(speed=ky.dat$speed,v1=model.matrix(orth.lm)[,2],v2=model.matrix(orth.lm)[,3])
ggpairs(new.df)
```

Our book discusses standardizing our covariates which does something similar, but orthogonal polynomials are probably more common in practice. We can also add a cubic term to the model:

```
ky.dat <- ky.dat %>% mutate(Year.3=Year^3)
poly3.lm<-lm(speed~Year+Year.sq+Year.3,data=ky.dat)
```

Does this appear to significantly improve the fit?

```r
orth3.lm<-lm(speed~poly(Year,3),data=ky.dat)
summary(orth3.lm)
```

```
##
## Call:
## lm(formula = speed ~ poly(Year, 3), data = ky.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.91024 -0.34659  0.02968  0.36472  1.26550
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)      35.8926     0.0497 722.193  < 2e-16 ***
## poly(Year, 3)1  11.5029     0.5964  19.287  < 2e-16 ***
## poly(Year, 3)2  -4.7979     0.5964  -8.045 3.31e-13 ***
## poly(Year, 3)3  -1.1729     0.5964  -1.967   0.0512 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5964 on 140 degrees of freedom
## Multiple R-squared:  0.7589, Adjusted R-squared:  0.7537
## F-statistic: 146.9 on 3 and 140 DF,  p-value: < 2.2e-16
```

If we want to account for track condition we note that there's a ton of levels:

```r
levels(ky.dat$condition)
```

```
## [1] "dusty"   "fast"    "good"    "heavy"   "muddy"   "sloppy"  "slow"
## [8] "wetfast"
```

After adjusting for year, we could fit:

```r
condition.lm<-lm(speed~poly(Year,2)+condition,data=ky.dat)
summary(condition.lm)
```

```
##
## Call:
## lm(formula = speed ~ poly(Year, 2) + condition, data = ky.dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.0476 -0.2359  0.0044  0.2583  0.8571
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)        36.5262     0.2540 143.793  < 2e-16 ***
## poly(Year, 2)1     10.5473     0.4759  22.164  < 2e-16 ***
## poly(Year, 2)2     -5.1658     0.4502 -11.476  < 2e-16 ***
## conditionfast      -0.3988     0.2590  -1.540  0.12600
## conditiongood      -0.7694     0.2800  -2.748  0.00682 **
## conditionheavy    -1.6762     0.2883  -5.813 4.26e-08 ***
## conditionmuddy    -1.6356     0.3006  -5.442 2.43e-07 ***
## conditionsloppy   -0.9222     0.3017  -3.057  0.00270 **
## conditionslow     -1.5786     0.3068  -5.145 9.32e-07 ***
## conditionwetfast  -0.8175     0.5013  -1.631  0.10527
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4192 on 134 degrees of freedom
## Multiple R-squared:  0.8859, Adjusted R-squared:  0.8783
## F-statistic: 115.7 on 9 and 134 DF,  p-value: < 2.2e-16
```

To see if this matters we can compare the smaller (nested) model via:

```
anova(poly.lm,condition.lm)
```

```
## Analysis of Variance Table
##
## Model 1: speed ~ Year + Year.sq
## Model 2: speed ~ poly(Year, 2) + condition
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1    141 51.172
## 2    134 23.553  7    27.619 22.448 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

So, again, here we have been talking about models like:

Where the $f(x_i)$ that we considered was the class of polynomial functions. Another class of regression models is fitting:

Why would we want to do this? Well, we might do it because we have a reason to believe that this is the underlying relationship. Another common reason is that our assumptions are violated. For instance, speed and stopping distance look like:

```
stopping.dat<-read_table("http://www.isi-stats.com/isi2/data/stopping.txt")%>%drop_na()
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   speed = col_double(),
##   stoppingdistance = col_double()
## )
```

```
stopping.dat %>% ggplot(aes(x=speed,y=stoppingdistance))+
  geom_point()
```

Here we see that a model relating speed to stopping distane linearly might not be appropriate. Why?

To account for this we might consider transforming y. One common transformation is $\log(y)$ (Note: From here on out in life, when someone writes log assume that it is natural log)

Why $\log(y)$? This comes from the delta method in statistics. The assumption we are making when we use the log transformation is that the variance of y is increasing as the expected value of y increases. That is:

The delta method says that any transformation of a random variable can be expressed as:

Similarly $\sqrt{y}$ can be used if we have y that has variance of:

So, any power transformation can be done if we want to correct or stabilize our variance and our variance is a function of our mean.

So why not transform?

Well, it changes the relationship between x_i and y_i. Note that previously we had assumed x_i and y_i had a linear relationship. Now, let's take $\log(y_i)$ and fit a regression model. The model is now:

So now the relationship between x_i and y_i becomes:

Overall, my recommendation is, if I don't care about exploring a linear relationship between x_i and y_i then transform away in order to fix assumptions. If we DO care about the linear relationshp, then we shouldn't do this.

Here we could do:

```
stopping.dat=stopping.dat %>% mutate(log.stop=log(stoppingdistance))
log.lm<-lm(log.stop~speed,data=stopping.dat)

log.lm %>% ggplot(aes(x=.fitted,y=.resid))+geom_point()
```

If we are satisfied with this we could find:

```
predict(log.lm,data.frame(speed=4),interval="prediction")
```

```
##        fit      lwr      upr
## 1 1.854145 1.104573 2.603718
```

But remember that this isn't a PI for stopping. It is for log(stopping), so our 95 % PI for stopping is

```
exp(1.1)
```

```
## [1] 3.004166
```

```
exp(2.6)
```

```
## [1] 13.46374
```

Let's see what happens though if we choose a different transformation:

```
stopping.dat=stopping.dat %>% mutate(sq.stop=sqrt(stoppingdistance))
sq.lm<-lm(sq.stop~speed,data=stopping.dat)

sq.lm %>% ggplot(aes(x=.fitted,y=.resid))+geom_point()
```

Fit looks better

```
predict(sq.lm,data.frame(speed=4),interval="prediction")
```

```
##        fit       lwr      upr
## 1 1.928557 0.4527392 3.404374
```

Again, this is for square root of stopping time, so to find actual interval we need:

```
.452^2
```

```
## [1] 0.204304
```

```r
3.4^2
```

```
## [1] 11.56
```

So certainly the transformation matters. How do we know that we have the *right* transformation? Well, we cannot do:

```r
anova(sq.lm,log.lm)
```

```
## Warning in anova.lmlist(object, ...): models with response '"log.stop"' removed
## because response differs from model 1

## Analysis of Variance Table
##
## Response: sq.stop
##           Df Sum Sq Mean Sq F value    Pr(>F)
## speed      1 386.06  386.06  746.22 < 2.2e-16 ***
## Residuals 61  31.56    0.52
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In fact it gives us a warning. Our models are NOT nested, so statistics doesn't help us here.

One way is to note that all of the transformations are special cases of what are known as Box-Cox transformations.

So we can find the value of λ that maximizes the log-likelihood of this.

```r
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```r
boxcox(stoppingdistance~speed,data=stopping.dat)
```

Here $\lambda = 0$ is log transformation and $\lambda = .5$ is square root, $\lambda = 1$ is no transformation. So it looks like $\lambda \approx .5$ would be preferable here.

So the model would be:

Note here there is not a nice clean linear or multiplicative relationship between x_i and y_i, but there is **a** relationship. If we want a predictive model this would suffice but if we want to explore a linear relationship between x_i and y_i this would not be appropriate.