Lab 6: Hands off my stack

Ben Thomas

11/4/2019

Data

Today, I'm looking at returns from stock ownership. First, let's download our dataset, which includes information about share prices, earnings, and returns.

```
d <- read.csv("https://bit.ly/36kibHZ")
```

Part 1: Inventing a variable

Let's now add a new variable, the P/E ratio, using the 10-year moving average of earnings.

```
library(tidyverse)
d <- d %>%
  mutate(MAPE = Price/Earnings_10MA_back)
```

Let's take a look at this new variable. Note that there are 120 N/A values. These are a result of there not being enough time for the 10-year moving average to be valid. For ease of calculation moving forward, I'll remove these values from the dataset.

```
summary(d$MAPE)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's 
##   4.785  11.708  15.947  16.554  19.959  44.196     120
```

```
data <- na.omit(d)
```

Now, let's try to predict the returns from owning the stock over the next ten years using this new variable.

```
m1 <- lm(Return_10_fwd ~ MAPE,
         data = data)
summary(m1)
```

```
## 
## Call:
## lm(formula = Return_10_fwd ~ MAPE, data = data)
## 
## Residuals:
##       Min        1Q    Median        3Q       Max 
## -0.116777 -0.029650  0.004347  0.028478  0.093157 
## 
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)    
## (Intercept)  0.1383475  0.0029889   46.29   <2e-16 ***
## MAPE        -0.0045885  0.0001727  -26.57   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04321 on 1482 degrees of freedom
## Multiple R-squared:  0.3226, Adjusted R-squared:  0.3221
## F-statistic: 705.8 on 1 and 1482 DF,  p-value: < 2.2e-16
```

Note that "MAPE" is significant at the 1% level, and has a coefficient of -0.0045885 with a standard error of 0.0001727. This model is really not too great, though, likely due to massive omitted variable bias.

Let's now try to estimate the MSE of this model using 5-fold CV. To do this, I'll first need to randomly assign each observation a fold number between 1-5. Note that I dropped the last integer of `partition_index` to match the number of observations in the dataset.

```
k = 5
partition_index <- rep(1:k, each = (nrow(data)+1)/k) %>%
  sample()
```

```
data <- data %>%
  mutate(p_i = partition_index[1:1484])
```

Now, let's use a for-loop to estimate the MSE.

```
MSE_i <- rep(NA, k)

for (i in 1:k) {
  data_train <- data[data$p_i !=i,]
  data_test <- data[data$p_i == i,]

  m1_ <- lm(Return_10_fwd ~ MAPE,
           data = data_train)

  MSE_i[i] = mean((data_test$Return_10_fwd - predict.lm(m1_, data_test)) ^ 2)
}

MSE_estimate <- mean(MSE_i)
MSE_estimate
```

```
## [1] 0.001870786
```

As we see here, the MSE of this model is 0.001868638.

Part 2: Inverting a variable

Let's now try this approach again, just inverting the MAPE variable. First, let's invert the variable and add it to the dataset.

```
data <- data %>%
  mutate(inverse_MAPE = 1/MAPE)
```

Now, let's use this inverted variable to estimate a linear model predicting returns.

```
m2 <- lm(Return_10_fwd ~ inverse_MAPE,
         data = data)
summary(m2)
```

```
##
## Call:
## lm(formula = Return_10_fwd ~ inverse_MAPE, data = data)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.106298 -0.030839  0.002955  0.028179  0.103866
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -0.007659   0.002878  -2.661  0.00788 **
## inverse_MAPE  0.995904   0.036513  27.275  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.04284 on 1482 degrees of freedom
## Multiple R-squared:  0.3342, Adjusted R-squared:  0.3338
## F-statistic: 743.9 on 1 and 1482 DF,  p-value: < 2.2e-16
```

Note that this new variable is significant at the 1% level, with a coefficient of 0.995904 and a standard error of 0.036513.

Let's now find the CV MSE of this new model and compare it to the original model. Since we've already partitioned the dataset, we can skip that step here.

```
MSE_i2 <- rep(NA, k)

for (i in 1:k) {
  data_train <- data[data$p_i !=i,]
  data_test <- data[data$p_i == i,]

  m2_ <- lm(Return_10_fwd ~ inverse_MAPE,
            data = data_train)

  MSE_i2[i] = mean((data_test$Return_10_fwd - predict.lm(m2_, data_test)) ^ 2)
}

MSE_estimate2 <- mean(MSE_i)
MSE_estimate2
```

```
## [1] 0.001870786
```

This is identical to the MSE estimated for the earlier model. This makes sense, as this is essentially the same exact model. We're using identical information to predict in both cases, just presented differently.

Part 3: A simple model

One simple way of modeling returns would indicate that returns should exactly equal the inverse of the 10-year moving average price-to-earnings ratio. Let's estimate the training MSE of this simple model and compare it to the earlier models.

```
MSE_estimate3 <- mean((data$Return_10_fwd - data$inverse_MAPE) ^ 2)
MSE_estimate3
```

```
## [1] 0.001896346
```

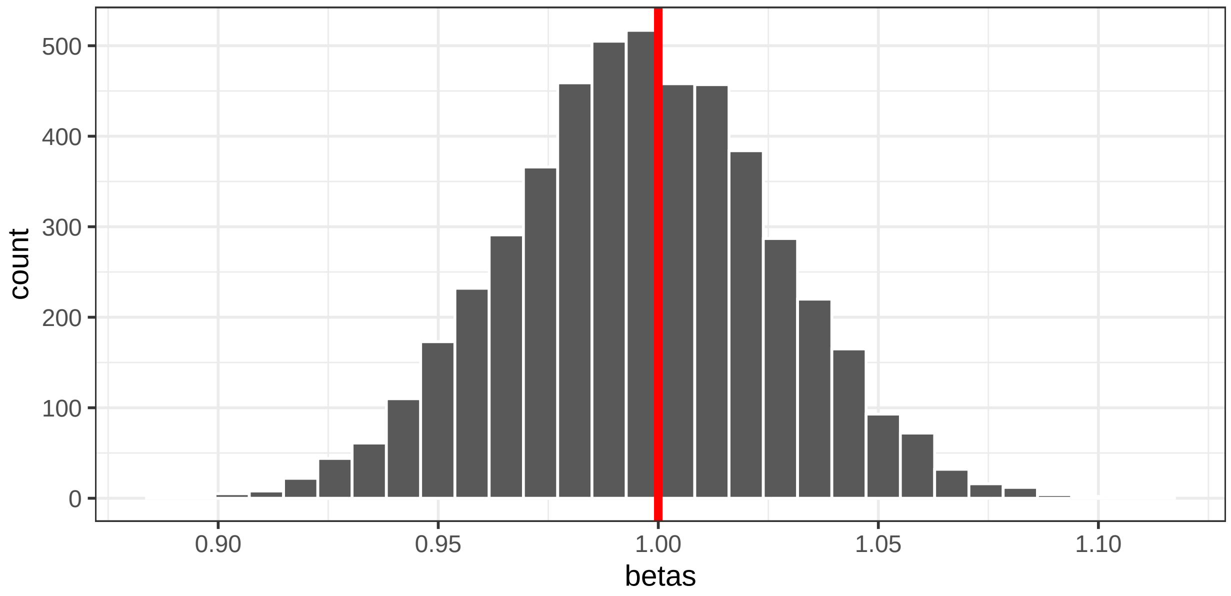
Note that this training MSE (which is slightly higher than that of our first two models) would be equivalent to the test MSE we would get through 5-fold CV. This is because our predictions do not really use a training dataset; they simply equate `inverse_MAPE` and `Return_10_fwd`.

Part 4: Is simple sufficient?

The second model we fit (in part 2) is very similar to the one we made in part 3. Let's compare the two! One method we haven't yet tried to do this is bootstrapping.

To compare the two models using the bootstrapping method, I'll simulate model 2 across many samples of the dataset, with replacement. Then I'll plot the distribution of coefficient estimates to see where the coefficient of the simple model (1) falls.

```
betas <- rep(NA, 5000)
for(i in 1:5000) {
  boot_ind <- sample(1:nrow(data),
                     size = nrow(data),
                     replace = TRUE)
  data_boot <- data[boot_ind, ]
  betas[i] <- coef(lm(Return_10_fwd ~ inverse_MAPE,
                      data = data_boot))[2]
}
```



As we can see here, the coefficient of the simple model, 1, indicated by the vertical red line, is nearly at the center of the distribution of model 2. To find the 95% confidence interval, we can use the following equation:

$$\bar{x} \pm z * \sigma$$

Plugging these values into R results in the following 95% confidence interval.

```
x <- mean(betas)
sigma <- sd(betas)
z <- 1.96
left_bound <- x - z * (sigma)
right_bound <- x + z * (sigma)
betas_conf_int <- c(left_bound, right_bound)
betas_conf_int
```

```
## [1] 0.9362952 1.0559653
```

We can compare this confidence interval to the one generated by our second model, using `confint()`. I've done so below.

```
confint(m2)
```

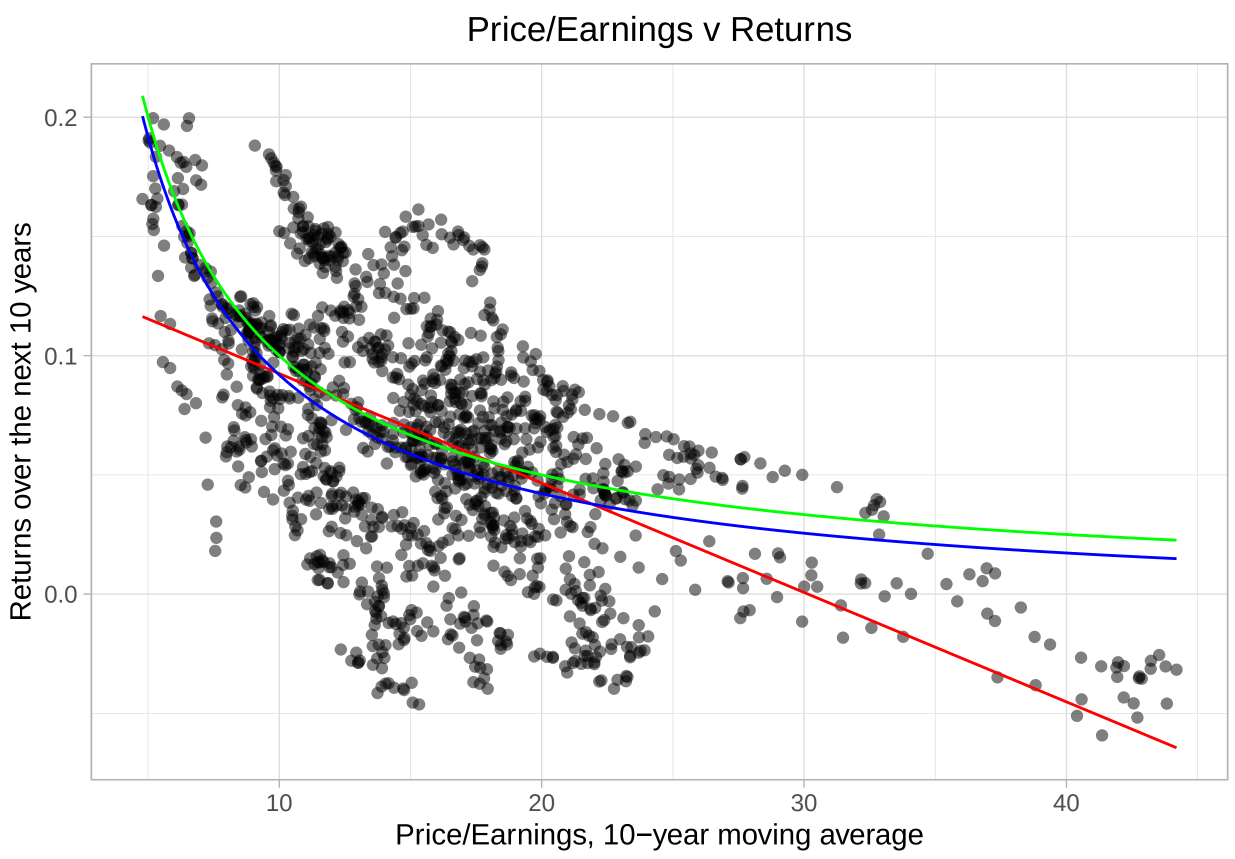
```
##                    2.5 %       97.5 %
## (Intercept)  -0.01330433 -0.002013051
## inverse_MAPE  0.92428102  1.067526198
```

Note that the confidence interval of our second model is almost identical, though a little wider in the confint() estimate.

Part 5: One big happy plot

Let's graphically look at the relationship between returns and price/earnings. Note that our first model, in red, predicts a linear relationship. The second and third models instead predict a non-linear trend.

```r
ggplot(data = data) +
  labs(title="Price/Earnings v Returns",
       y="Returns over the next 10 years",
       x="Price/Earnings, 10-year moving average") +
  theme_light() + theme(plot.title = element_text(hjust = 0.5)) +
  geom_point(aes(x = MAPE, y = Return_10_fwd), alpha = 0.5) +
  geom_line(aes(x = MAPE, y = predict(m1)), color = "red") +
  geom_line(aes(x = MAPE, y = predict(m2)), color = "blue") +
  geom_line(aes(x = MAPE, y = inverse_MAPE), color = "green")
```



The big picture

1. CV for model selection

Based on the CV MSEs, I'd use one of the first two models to predict returns. Between the two, I'd choose the linear model to predict the returns. While this model has less strength toward the lower end of the P/E ratio, it captures the end tail better than the other models.

2. Bootstrapping for uncertainty estimation

Based on our earlier bootstrapping estimation, the simple model is absolutely plausible, and well within the 95% confidence interval.

Chapter 5 exercises

Problem 4

We could estimate the standard deviation of a prediction using the following steps.

1. Create new samples of X and Y using bootstrapping;
2. Create new estimates of the prediction using the new datasets;
3. Find the mean of these estimates;
4. Find the average distance from the mean in these estimates. This is your estimate of the standard deviation!

Problem 8

Part A

```r
set.seed(1)
x=rnorm(100)
y=x-2*x^2+rnorm (100)
```

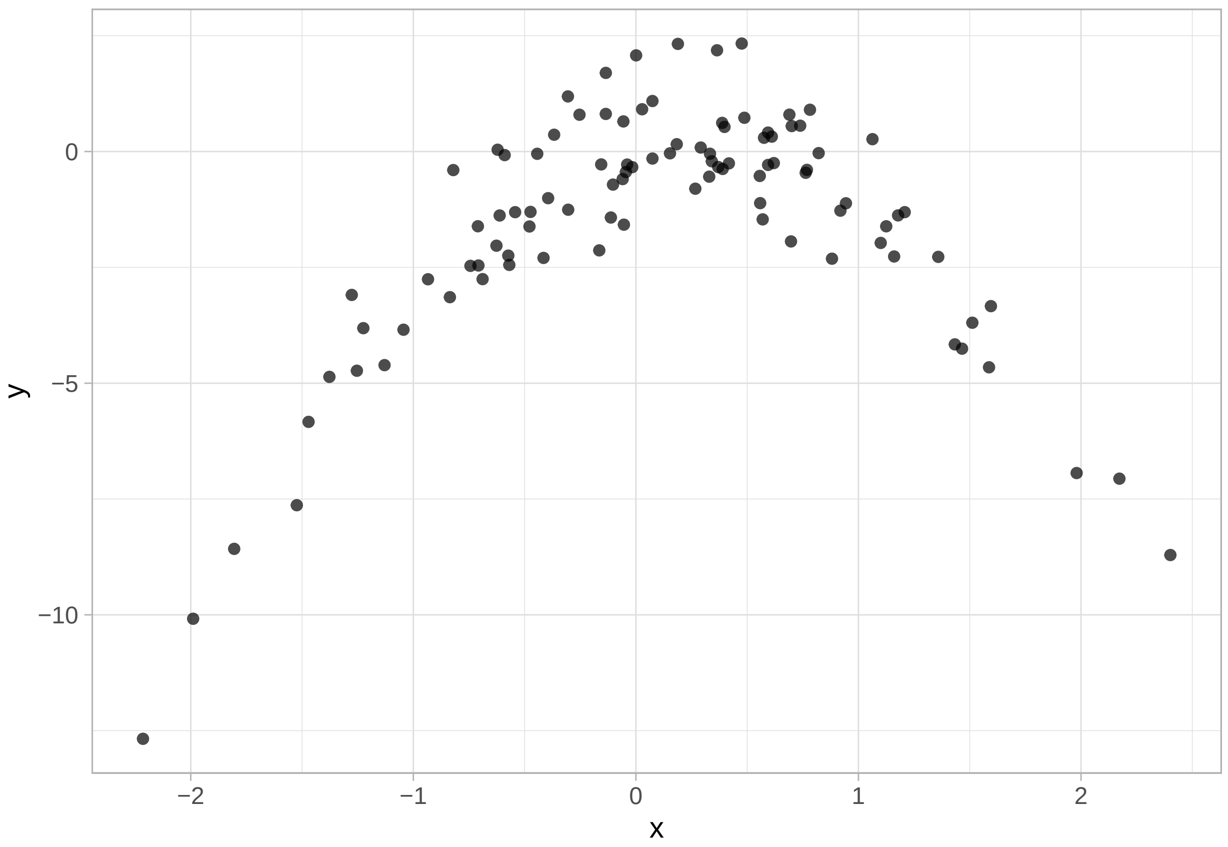
In this scenario, n = 100 and p = 1. In equation form, the model used to generate the data is:

$$y = x - 2x^2 + \epsilon$$

Part B

Let's create a scatterplot of X and Y.

```r
random_data <- tibble(x, y)
ggplot(random_data, aes(x=x, y=y)) +
  theme_light() +
  geom_point(alpha = .7)
```



As we see here, the relationship between the two is initially positive, but turns negative after reaching a peak around x = 0.25.

Part C

Let’s compute the LOOCV error for 4 models using this data, each with a higher-order polynomial term.

```r
library(boot)
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following object is masked from 'package:tidyr':
##
##     expand

## Loading required package: foreach

##
## Attaching package: 'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when
## Loaded glmnet 2.0-16
```

```r
set.seed(234098724)
cv.error.1 <- rep(0, 4)
for (i in 1:4){
  glm.fit = glm(y ~ poly(x,i), data=random_data)
  cv.error.1[i] = cv.glm(random_data, glm.fit)$delta[1]
}
cv.error.1
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

As we see here, there is a sharp decrease in the LOOCV error moving from 1 to 2 terms, but no real change after.

Part D

Let's make this computation again with a different seed.

```r
library(boot)
library(glmnet)
set.seed(35)
cv.error.2 <- rep(0, 4)
for (i in 1:4){
  glm.fit2 = glm(y ~ poly(x,i), data=random_data)
  cv.error.2[i] = cv.glm(random_data, glm.fit2)$delta[1]
}
cv.error.2
```

```
## [1] 7.2881616 0.9374236 0.9566218 0.9539049
```

My results (LOOCV errors) are exactly the same as before. This is because we aren't making any random variables at this stage, so everything is the same between the two.

Part E

The model with the lowest LOOCV in Part C was the model with two terms. This makes sense, as this is the closest to the theoretical model.

Part F

Let's compare these models by hand.

```r
set.seed(234098724)
glmfit1 <- glm(y ~ x, data = random_data)
summary(glmfit1)
```

```
##
## Call:
## glm(formula = y ~ x, data = random_data)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -9.5161  -0.6800   0.6812   1.5491   3.8183
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -1.6254     0.2619  -6.205 1.31e-08 ***
## x             0.6925     0.2909   2.380   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 6.760719)
##
##     Null deviance: 700.85  on 99  degrees of freedom
## Residual deviance: 662.55  on 98  degrees of freedom
## AIC: 478.88
##
## Number of Fisher Scoring iterations: 2
```

```r
set.seed(234098724)
random_data <- random_data %>%
  mutate(sqrx = x*x,
         tripx = x*x*x,
         quadx = x*x*x*x)
glmfit2 <- glm(y ~ x + sqrx, data = random_data)
summary(glmfit2)
```

```
##
## Call:
## glm(formula = y ~ x + sqrx, data = random_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9650  -0.6254  -0.1288   0.5803   2.2700
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.05672    0.11766   0.482    0.631
## x            1.01716    0.10798   9.420  2.4e-15 ***
## sqrx        -2.11892    0.08477 -24.997  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9178258)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  89.029  on 97  degrees of freedom
## AIC: 280.17
##
## Number of Fisher Scoring iterations: 2
```

```r
set.seed(234098724)
glmfit3 <- glm(y ~ x + sqrx + tripx, data = random_data)
summary(glmfit3)
```

```
##
## Call:
## glm(formula = y ~ x + sqrx + tripx, data = random_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.9765  -0.6302  -0.1227   0.5545   2.2843
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.06151    0.11950   0.515    0.608
## x            0.97528    0.18728   5.208 1.09e-06 ***
## sqrx        -2.12379    0.08700 -24.411  < 2e-16 ***
## tripx        0.01764    0.06429   0.274    0.784
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9266599)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  88.959  on 96  degrees of freedom
## AIC: 282.09
##
## Number of Fisher Scoring iterations: 2
```

```r
set.seed(234098724)
glmfit4 <- glm(y ~ x + sqrx + tripx + quadx, data = random_data)
summary(glmfit4)
```

```
##
## Call:
## glm(formula = y ~ x + sqrx + tripx + quadx, data = random_data)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.0550  -0.6212  -0.1567   0.5952   2.2267
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.156703   0.139462   1.124    0.264
## x            1.030826   0.191337   5.387 5.17e-07 ***
## sqrx        -2.409898   0.234855 -10.261  < 2e-16 ***
## tripx       -0.009133   0.067229  -0.136    0.892
## quadx        0.069785   0.053240   1.311    0.193
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.9197797)
##
##     Null deviance: 700.852  on 99  degrees of freedom
## Residual deviance:  87.379  on 95  degrees of freedom
## AIC: 282.3
##
## Number of Fisher Scoring iterations: 2
```

As we see here, in the first model, x is significant at the 0.05 level. In all models afterward, x and x^2 are both significant at the 0.001 level, and none else are significant at all. These results do agree with those from the CV, which indicated that the model with x and x^2 in it was the best. These two are the only significant terms, so should be the only ones included.