Specification and Data Issues

BS1802 Statistics and Econometrics

Jiahua Wu

Part I

Functional form misspecification

The two tests for functional form misspecification, namely RESET (on slide 15) and Davidson MacKinnon test (on slide 19) are readily implemented in R in the *lmtest* library. The function for RESET test is *resettest*, which implements an F test for the expanded model in Step 2 with null hypothesis $H_0 : \delta_{\hat{y}^2} = 0, \delta_{\hat{y}^3} = 0$. If we reject null, we reject the null hypothesis that functional form is correctly specified. Smaller the p-value, stronger the evidence against null.

```
load("hprice1.RData")

# RESET test
house.m1 <- lm(price ~ lotsize + sqrft + bdrms, data)
resettest(house.m1, type = "fitted")
```

```
##
##  RESET test
##
## data:  house.m1
## RESET = 4.6682, df1 = 2, df2 = 82, p-value = 0.01202
```

```
house.m2 <- lm(log(price) ~ log(lotsize) + log(sqrft) + bdrms, data)
resettest(house.m2, type = "fitted")
```

```
##
##  RESET test
##
## data:  house.m2
## RESET = 2.565, df1 = 2, df2 = 82, p-value = 0.08308
```

Comparing the p-values from the two models above, there is stronger evidence of functional form misspecification in the *price* model. Thus, we would prefer the log-log model between the two.

Davidson-MacKinnon test is implemented in the *jtest* function. The test allows us to compare two competing models, by adding fitted values from one model to the other. On slide 19, if we can reject the null $H_0 : \theta = 0$, we simply reject the null hypothesis that the first model is correctly specified.

```
# Davidson-MacKinnon test
lprice.m1 <- lm(log(price) ~ lotsize + sqrft + bdrms, data)
lprice.m2 <- lm(log(price) ~ log(lotsize) + log(sqrft) + bdrms, data)
jtest(lprice.m1, lprice.m2)
```

```
## J test
##
## Model 1: log(price) ~ lotsize + sqrft + bdrms
## Model 2: log(price) ~ log(lotsize) + log(sqrft) + bdrms
##                 Estimate Std. Error t value Pr(>|t|)
## M1 + fitted(M2)  0.82779    0.35337  2.3425  0.02155 *
```

```
## M2 + fitted(M1)  0.33201    0.43170  0.7691  0.44403
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In this example, we can reject the null hypothesis that coefficient of $fitted(M2)$ equals to 0, but fail to reject the null hypothesis that coefficient of $fitted(M1)$ equals to 0. We thus reject the first model, and prefer the second one.

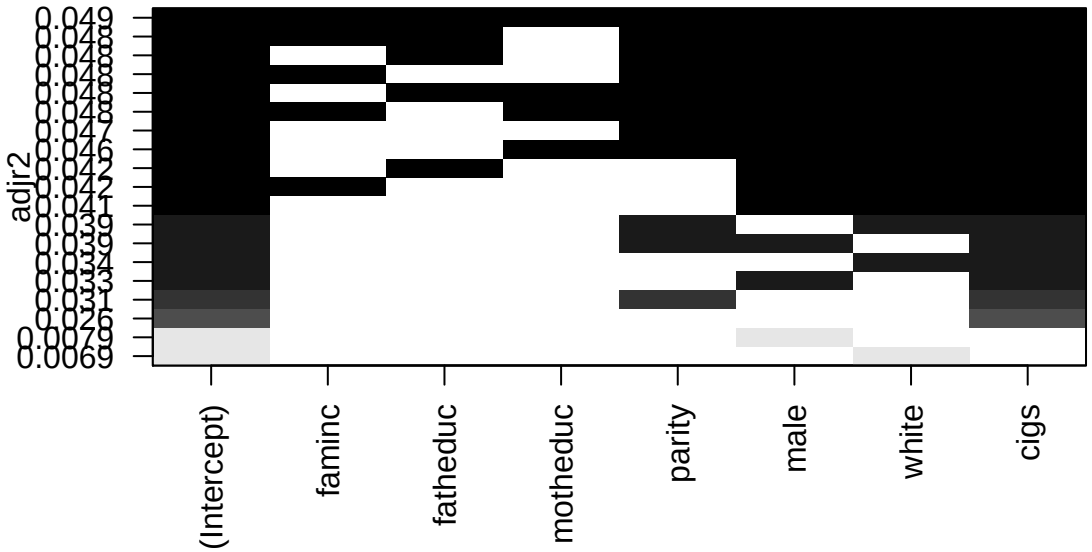
Variable Selection

In the class, we discuss two functions that will automatically evaluate different models, and report the best ones depending on goodness-of-fit measure of choice. The first function is *regsubsets* implemented in the *leaps* library. It would perform an exhaustive search, and thus computational complexity increases exponentionally (in the number of independent variables).

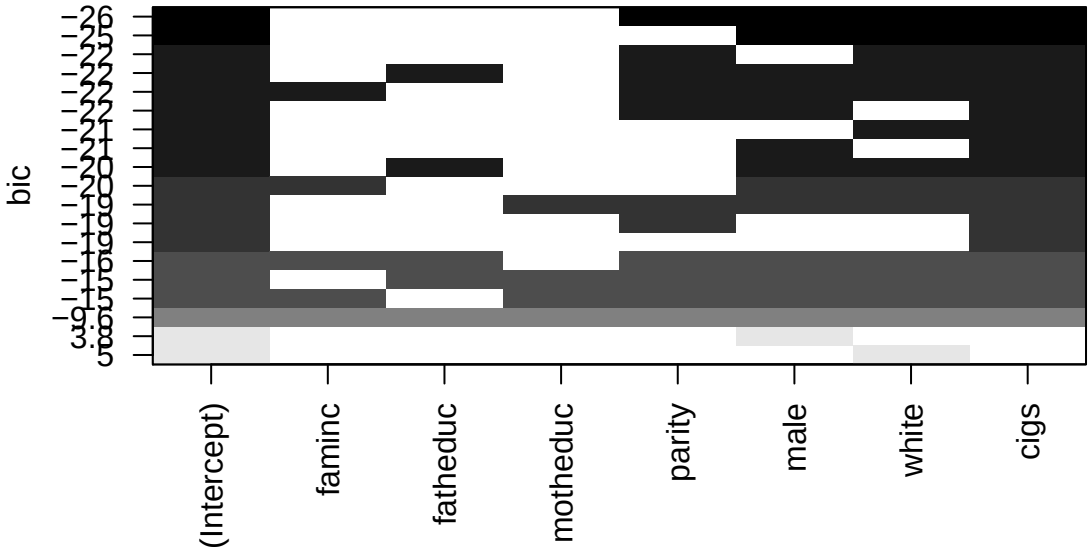
```
load("bwght.RData")
data.new <- na.omit(data)
bwght.model.search <- regsubsets(bwght ~ faminc + fatheduc + motheduc + parity
                                 + male + white + cigs, data.new, nbest = 3)
```

Before running *regsubsets*, we first remove any observations with missing value in the sample, such that different models are compared against the same sample. *nbest* specifies the number of subsets of each size to record. We next plot the models based on two goodness-of-fit measures, namely $\bar{R}^2$ and BIC.

```
plot(bwght.model.search, scale = "adjr2")
```




```
plot(bwght.model.search, scale = "bic")
```



The way to read the figures is the following: each row indicates one model and each column indicates one independent variable. If a block is black (or grey, they are the same), it indicates that the corresponding independent variable (column) is included in the corresponding model (row). One thing to mention is that the vertical axis in the BIC figure indicates the difference in BIC between the corresponding model and a model with only the intercept, rather than the model BIC value.

The other function we discussed for automatic variable selection is *step*. It would perform stepwise search to find the model with the best goodness-of-fit measure of choice. The function can perform $forward$ stepwise search, *backward* stepwise search, or *both* directions. With $forward$, at each iteration, the function would evaluate the model AICs by adding each of the independent variables that are not already included in the model, and add the one such that the resulting model has the lowest AIC. The algorithm stops when no more independent variables can be added to the model for a lower AIC. With *backward*, at each iteration, the function would evaluate the model AICs by removing each of the independent variables that are already in the model, and remove the one such that the resulting model has the lowest AIC. With *both*, at each iteration, the algorithm would evaluate the model AICs by adding each of the independent variables that are not already included in the model, and the model AICs by removing each of the independent variables that are already in the model. At the end of each iteration, it would return the model with the lowest AIC.

At the end of these automatic algorithms, it will return one model with the lowest AIC. However, as we discussed in the class, we may find several models with similar AICs (with difference less than 2), then we can decide whether to include those variables (in one model but not the other) based on subject knowledge, and etc. There is no fixed rule here. Model validation (which you will learn in the machine learning class) could also help you to pick the model.

```
# stepwise search
bwght.null <- lm(bwght ~ 1, data.new)
bwght.full <- lm(bwght ~ faminc + fatheduc + motheduc + parity + male + white
                 + cigs, data.new)
step(bwght.null, scope = list(lower = bwght.null, upper = bwght.full),
     direction = "forward")
step(bwght.full, direction = "backward")
step(bwght.null, scope = list(lower = bwght.null, upper = bwght.full),
     direction = "both")
```

If we want to evaluate models based on BIC, we can simply add the argument $k = log(n)$ in the *step* function.

```
n <- nrow(data.new)
step(bwght.null, scope = list(lower = bwght.null, upper = bwght.full),
     direction = "forward", k = log(n))
```

Prediction

The function for model prediction in R is *predict*. One argument that is worth mentioning is *interval*. If we specify $interval =$ "$confidence$", then we are predicting for an average person - error term u no longer plays a role here as on average it is equal to 0. On the other hand, if we specify $interval =$ "$predict$", we are predicting wage for a particular person. In this case, we need to account for two sources of variation: sampling variation (as we don't really know the true population parameter), and variance in the error term (as we don't observe it for this particular individual). As such, the prediction interval would be much wider.

```
load("wage1.RData")
wage.m1 <- lm(wage ~ educ + exper, data)
newdata <- data.frame(educ = 12, exper = 8)

# prediction for an average person
predict(wage.m1, newdata, interval = "confidence", level = 0.95)
```

```
##        fit      lwr      upr
## 1 4.901489 4.546359 5.256618
```

```
# prediction for a particular individual
predict(wage.m1, newdata, interval = "predict", level = 0.95)
```

```
##        fit       lwr      upr
## 1 4.901489 -1.506855 11.30983
```

Lastly, when we predict y in a model where the dependent variable is $log(y)$, we need to be aware that simply exponentiating the fitted value wouldn't give us the right answer. We need to scale it up by a sample estimate of $E(exp(u))$, which is given by the sample average of exponential of residuals.

```
wage.m2 <- lm(log(wage) ~ educ + exper, data)
predicted.logwage <- predict(wage.m2, newdata, interval = "none")
predicted.wage <- mean(exp(wage.m2$residuals)) * exp(predicted.logwage)
```

Part II

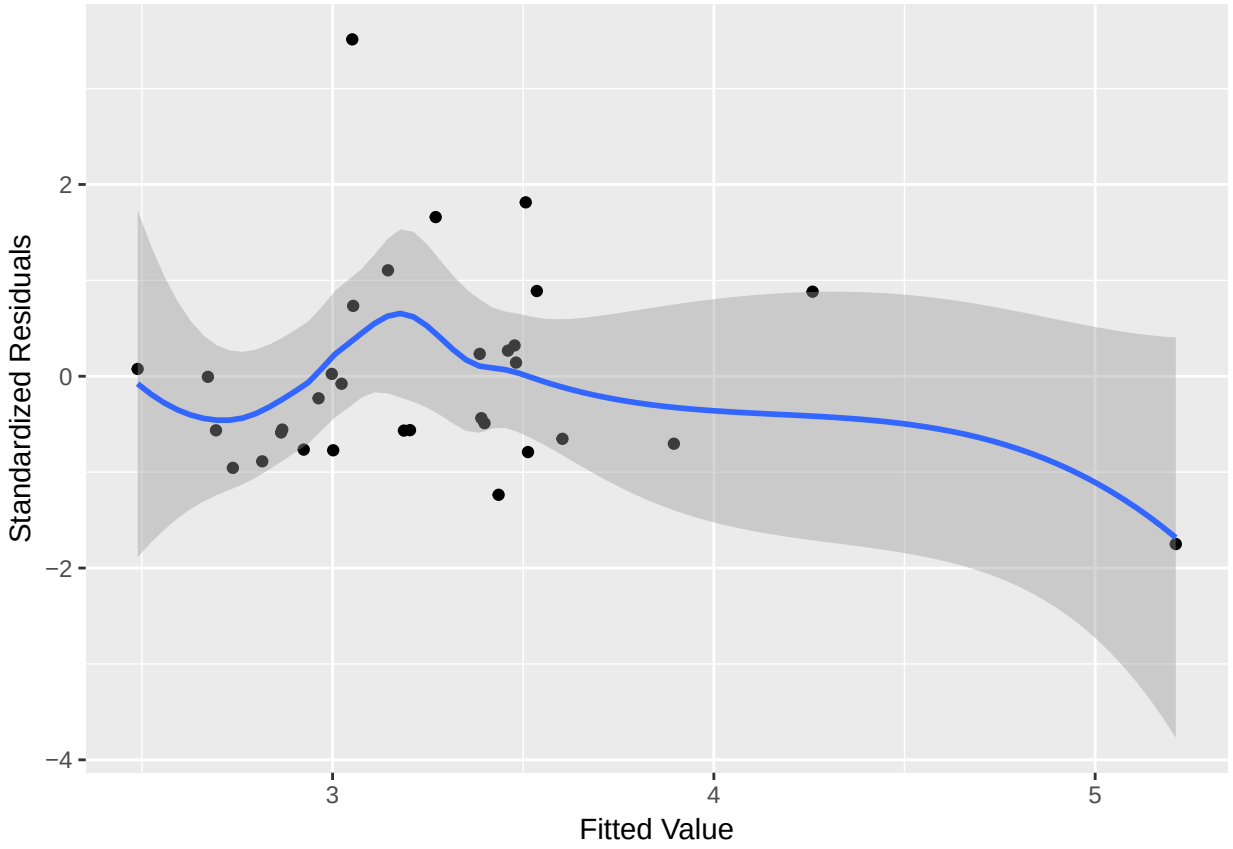
Outlier and leverage points

One common way to determine whether there exist outliers/leverage points in the sample is through visual inspection. Plots that involve standardized residues, leverage values or Cook's distance can all help us with this regard. We draw two plots for this particular example. The first one is "standardized residuals vs fitted values", which can help us identify outliers (unusual y values). The rule of thumb that is commonly used in practice is to find those observations that lie 3 standard deviations away from the mean. Judging from the plot, we don't have outliers in this example.

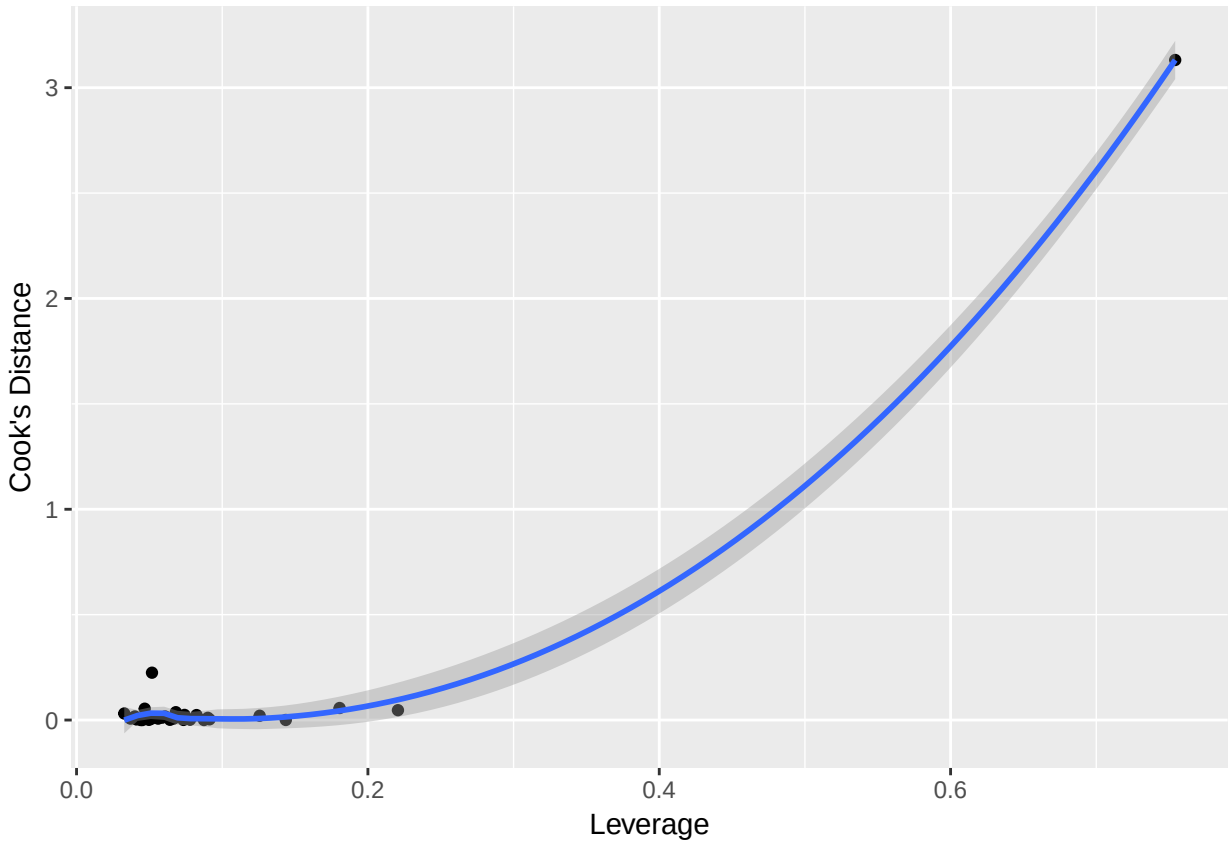
The second plot is "Cook's distance vs leverage values", which can help us identify high leverage points (unusual x values). Again, we think an observation has high leverage if its leverage value is greater than twice of the mean $(2(k+1)/n)$ and/or Cook's distance is greater than 1.

```
load("rdchem.RData")
rdchem.m1 <- lm(rdintens ~ sales + profmarg, data)

ggplot(rdchem.m1, aes(.fitted, .stdresid)) + geom_point() + stat_smooth(method = "loess") +
  xlab("Fitted Value") + ylab("Standardized Residuals")
```




```
ggplot(rdchem.m1, aes(.hat, .cooksd)) + geom_point() + stat_smooth(method = "loess") +
  xlab("Leverage") + ylab("Cook's Distance")
```



In the example, we clearly have one observation that satisfies both conditions (leverage value is around 0.75, and Cook's distance around 3.13), and another one with leverage value slightly greater than twice the average (around 0.22). We need to examine them further.

```
# leverage value
hatvalues(rdchem.m1)
```

```
##          1          2          3          4          5          6
## 0.05178072 0.05917460 0.07793055 0.14364715 0.09129965 0.04097883
##          7          8          9         10         11         12
## 0.08744683 0.03698317 0.18058297 0.75418395 0.05875135 0.04359415
##         13         14         15         16         17         18
## 0.05599620 0.09012704 0.04412758 0.05115140 0.22063132 0.04518525
##         19         20         21         22         23         24
## 0.12571730 0.06412486 0.08234326 0.04678718 0.07338371 0.06816198
##         25         26         27         28         29         30
## 0.04968940 0.04001925 0.06678673 0.06077687 0.03275102 0.07411315
##         31         32
## 0.04100128 0.04077131
```

```
# cooks distance
cooks.distance(rdchem.m1)
```

```
##            1            2            3            4            5
## 2.246774e-01 1.310179e-02 2.010315e-03 3.230196e-04 3.456981e-03
##            6            7            8            9           10
## 4.574012e-03 1.140738e-06 6.890940e-03 5.710895e-02 3.131360e+00
##           11           12           13           14           15
## 1.217157e-02 2.907136e-03 6.790536e-03 1.049967e-02 9.016596e-06
```

```
##           16           17           18           19           20
## 5.674131e-03 4.672589e-02 9.645603e-05 2.041507e-02 1.242497e-03
##           21           22           23           24           25
## 2.363966e-02 5.381593e-02 5.362091e-04 3.726535e-02 9.197927e-04
##           26           27           28           29           30
## 1.696774e-02 7.374571e-03 1.697529e-02 3.109594e-02 2.435721e-02
##           31           32
## 3.412649e-03 8.445907e-03
```