BIFX 553 - Model Building Review

Randy Johnson

February 23, 2017

Setup

```
library(shrink)
library(car)
library(tidyverse)
library(broom)
theme_set(theme_classic() +
            theme(axis.line.x = element_line(color = 'black'),
                  axis.line.y = element_line(color = 'black'),
                  text = element_text(size = 15)))
```

Mulivariable Modeling Strategies

Prespecification of predictor complexity

- Deciding on a model before analysis of the data is the best policy when model building. Unless you have a good reason to remove something from the model after beginning the analysis, you probably shouldn't.
- If you don't know much about the system you are modelling, decide before hand how complex you want your model to be.

Variable selection

There are automated methods for building your model (e.g. foward or backward selection). These are best avoided because:

- r^2 values will be biased too high.
- Test statistics do not have the correct distribution.
- Standard errors / confidence intervals are biased smaller than they should be.
- p-values will be too small
- Collinearity makes variable selection arbitrary
- It allows you to turn off your brain.

Overfitting

Overfitting is likely to occur if the number of parameters is too high.

At some point, new parameters added to the model will fit sampling noise rather than the information we are trying to make inferrences about.

A common rule of thumb for picking the number of variables is $\frac{sample\ size}{10}$ or $\frac{sample\ size}{20}$ for continuous variables (see pg 61 of Harrell, 2001 for more information).

For genetic markers, this is more like $\frac{MAF}{10}$ or $\frac{MAF}{20}$, where MAF = minor allele frequency.

Regression to the Mean

When using a model to predict outcomes in independent data, predicted values that are far away from the mean will tend to over estimate the deviation from the mean, while predicted values near the mean will tend to be more accurate.

```
set.seed(28347) # this is the first seed I tried
n_pred <- 100
pred <- data_frame(x1 = rnorm(n_pred),
                   x2 = rnorm(n_pred),
                   x3 = rnorm(n_pred),
                   x4 = rnorm(n_pred),
                   x5 = rnorm(n_pred),
                   y = x1 + x2 + x3 + rnorm(n_pred))

model <- lm(y ~ x1 + x2 + x3 + x4 + x5,
            data = pred, y = TRUE, x = TRUE)
```

Regression to the Mean

```
tidy(model)
```

```
##          term    estimate  std.error  statistic      p.value
## 1 (Intercept) -0.09277074 0.09253871 -1.0025073 3.186723e-01
## 2          x1  1.06597935 0.09177622 11.6149837 7.225899e-20
## 3          x2  1.05377814 0.08049772 13.0907824 6.649894e-23
## 4          x3  1.07985388 0.08990097 12.0115926 1.082171e-20
## 5          x4 -0.07176524 0.09221576 -0.7782318 4.383869e-01
## 6          x5  0.24741839 0.09523223  2.5980530 1.088397e-02
```

Shrinkage

The regression coefficient for $x5$ is too big, and the coefficients for $x1$, $x2$ and $x3$ are all slightly too big. Predictions of y using this model will be slightly too big, and this effect will be magnified for points lying further from the mean.

Shrinkage can be used to compensate some for this (models with more parameters will bennefit more).

```
shrink(model)
```

```
## Shrinkage Factors (type=parameterwise, method=jackknife)
##          x1          x2          x3          x4          x5
##  0.9816159   1.0035074   0.9884095 -0.5527423   0.8634528
##
## Shrunken Regression Coefficients:
## (Intercept)          x1          x2          x3
## -0.09320754   1.04638229   1.05747420   1.06733782   0.03966
```

Collinearity

- Standard errors will be inflated.
- Correlated coefficients are difficult to estimate, because data provide limited information when holding another, correlated parameter constant.
- It is difficult to sort out the important variable(s)

Data reduction

Using what is known about the system you are modelling can be on of the best ways to reduce the number of parameters. Read the literature / consult experts.

Clustering or summarizing (e.g. using principal components) can be another powerful tool to reduce the dimensionality of your data (more on this to come in future discussions).

Overly influential variables

Sources include: - Data entry errors - Measurement error (if a particular variable is too noisy, it may be best to just throw out all obervations of that particular predictor) - Data need to be transformed - Not enough data to model a complex system - Model doesn't fit a specific subset of the data

Comparing two models

- Consider model assumptions / all of the above.
- A smaller model isn't always preferred if important statistically non-significant variables are omitted.
- Does it agree with our understanding of the system (e.g. our disease model)?

Summary: Developing predictive models

- Collect lots of data.
- Do you homework when developing the disease model/hypothesis.
- Impute missing data if there isn't much. Will people using your model have trouble collecting all the variables you are using?
- Reduce the number of predictors if necessary/practical.
- Check all assumptions.
- Influential observations are especially bad in predictive models.
- Validate your final model.
- Use parameter shrinkage if possibility of over fitting.
- Develop some simplified alternative models.

Summary: Developing models for effect estimation

- Parsimony isn't as important in this context.
- Interactions should still receive carful consideration.
- Imputation of predictors is less helpful.
- Check all assumptions.

Summary Developing models for hypothesis testing

- Check all assumptions.
- Interactions should still receive careful consideration.
- Imputation might be helpful, but often results in no net benefit.

Assumptions Review

Example data

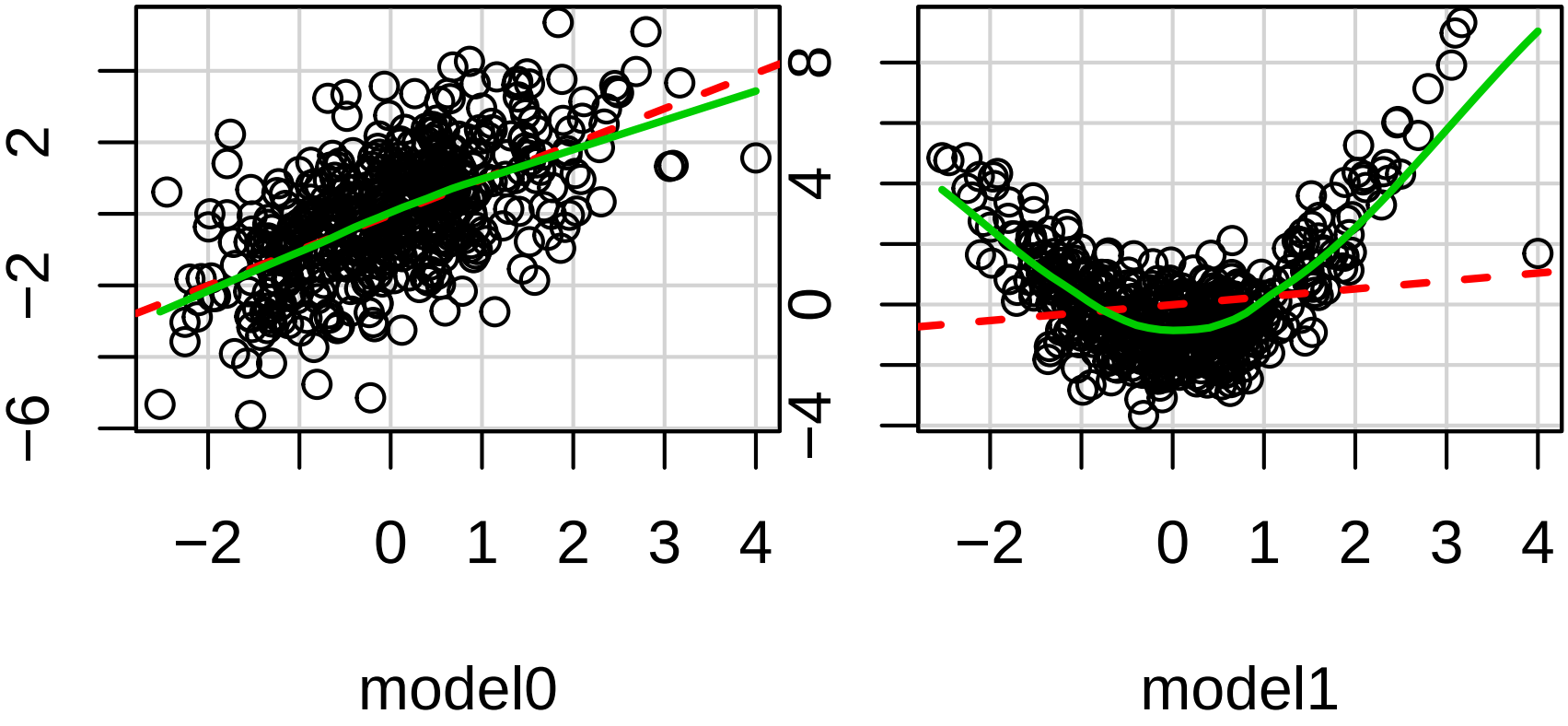
```
set.seed(923747)
n <- 500
dat <- data_frame(x = rnorm(n),
                  x2 = rnorm(n, x, 0.5),
                  x3 = rnorm(n),
                  x4 = 1:n,
                  y0 = x + x3 + rnorm(n),
                  nonlin_y = x^2 + rnorm(n),
                  auto_y = diffinv(rnorm(n-1)),
                  homo_y = x + rnorm(n, sd = (x + abs(min(x)))),
                  out_y = y0) %>%
       bind_rows(data_frame(x = c(0,4), out_y = rep(10,2)))


dat$x[n-1] <- max(dat$x) # outlier with leverage
dat$x[n] <- 0 # outlier without leverage

# OK
model0 <- lm(y0 ~ x, data = dat)
```

Example data

```r
# Nonlinear + Non-normal
model1 <- lm(nonlin_y ~ x, data = dat)

# Multicollinearity
model2 <- lm(y0 ~ x + x2 + x3, data = dat)
model3 <- update(model2, . ~ . - x2) # OK

# Autocorrelation
model4 <- lm(auto_y ~ x4, data = dat)

# Homoscedasticity
model5 <- lm(homo_y ~ x, data = dat)

# with outliers
model6 <- lm(out_y ~ x, data = dat)
```

Linearity

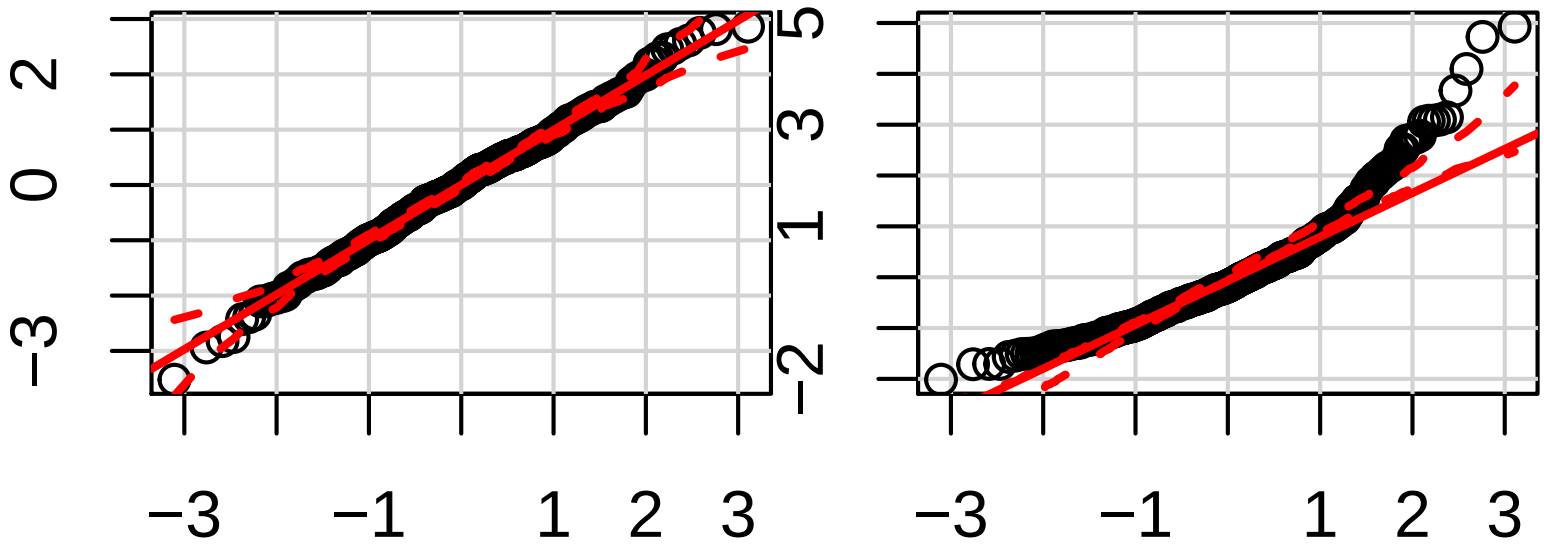
Component Residual Plots, `crPlots()`, shows the relationship between each predictor and the outcome, after accounting for all of the other variables in the model. Since this doesn't work so well with interactions, simply remove the interactions to view these relationships.



Multivariate Normality

There are a few ways to check for multivariate normality.

- `shapiro.test()`
- `qqPlot()`
- Plotting a histogram of the residuals (redundant)



model0 (p=0.44) model1 (p=1.9e−15)

Little/No Multicollinearity

If the variance inflation factors are too big (e.g. greater than 2), they are likely to be collinear with another predictor. Removing one of the collinear variables is usually sufficient to meet this assumption.

```
vif(model2)
```

```
##        x       x2       x3
## 4.700179 4.700470 1.000400
```

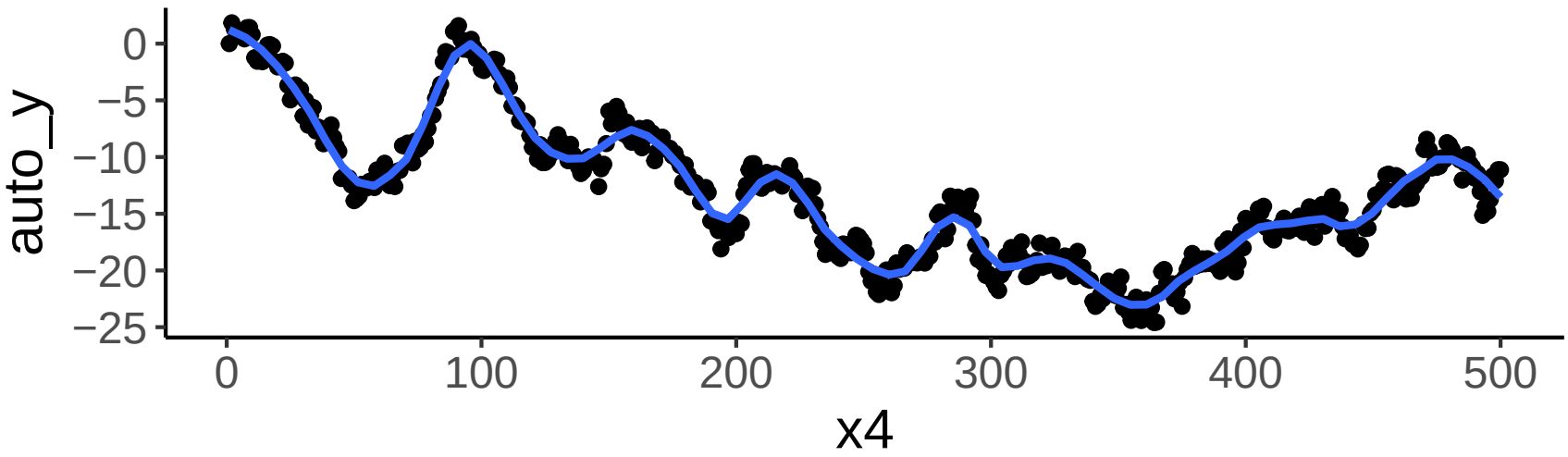
```
vif(model3)
```

```
##        x       x3
## 1.000336 1.000336
```

No Autocorrelation

Autocorrelation occurs when there is correlation between the elements of a series that are separated by a given interval (e.g. 1 day).

```
## Warning: Removed 2 rows containing non-finite values (st

## Warning: Removed 2 rows containing missing values (geom_
```



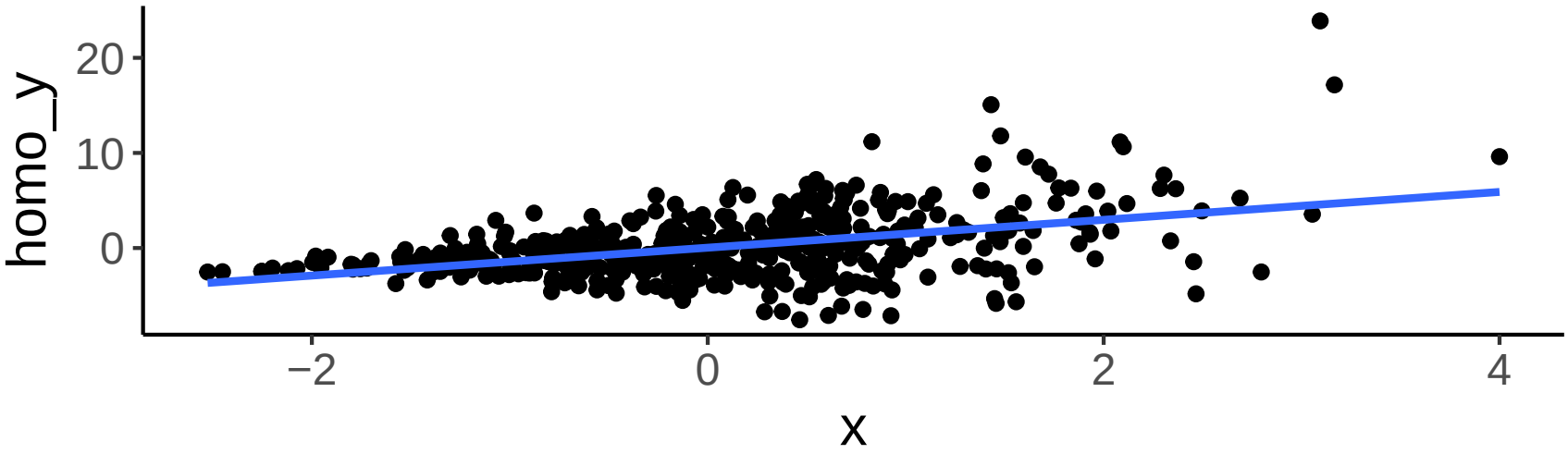

```
##  lag Autocorrelation D-W Statistic p-value
##    1       0.9726548    0.04455744       0
```

Homoscedasticity

Heteroscedasticity occurs when the variance about the regression line is not constant. We can test this assumption with `ncvTest()` or `spreadLevelPlot()`.

```
## Warning: Removed 2 rows containing non-finite values (st
```

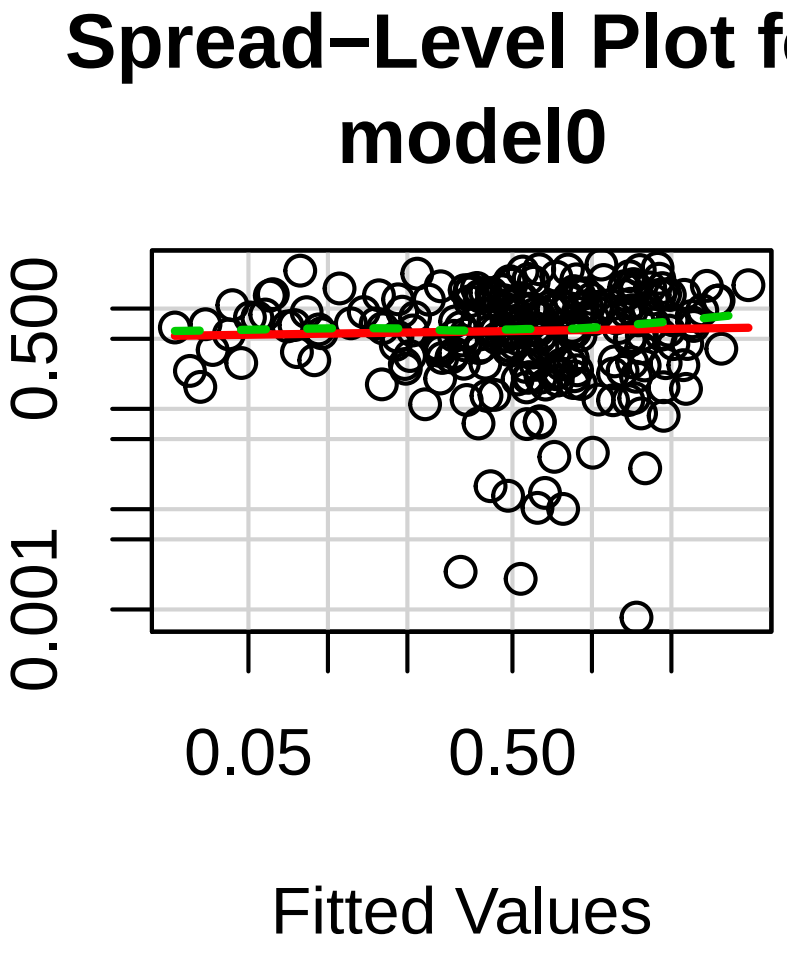
```
## Warning: Removed 2 rows containing missing values (geom_
```

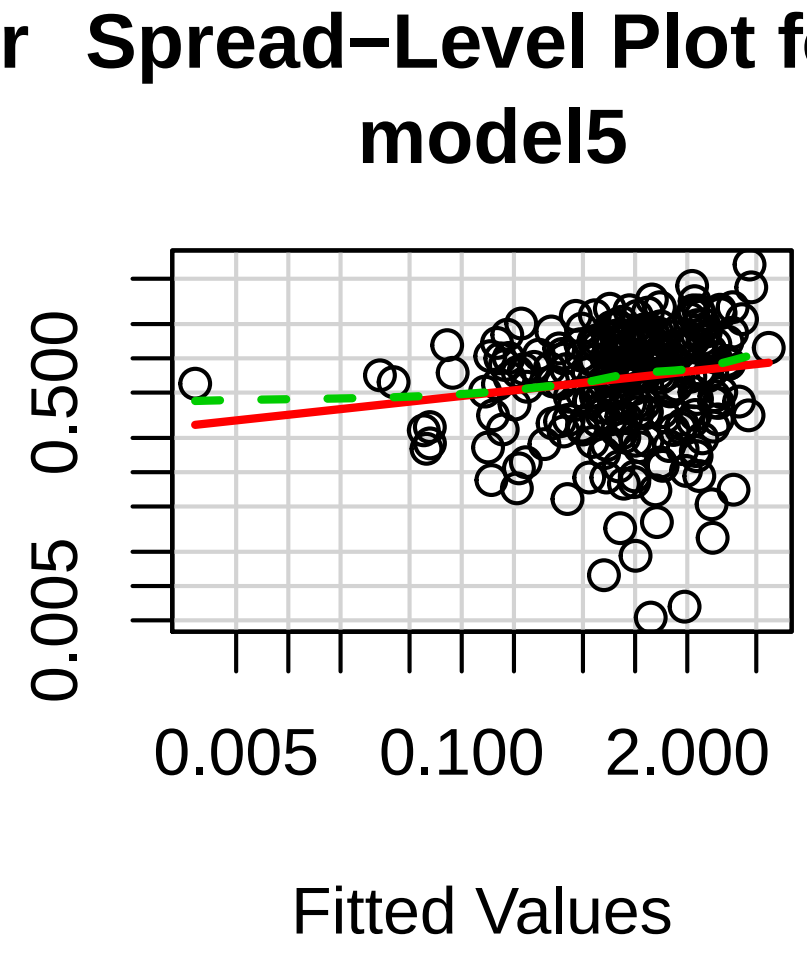



```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
```

Homoscedasticity

We want both trend lines to be as flat as possible (i.e. straight line wth slope $= 0$).





Outliers and Leverage

Use `outlierTest()` to get a list of outliers.

```
outlierTest(model6)
```

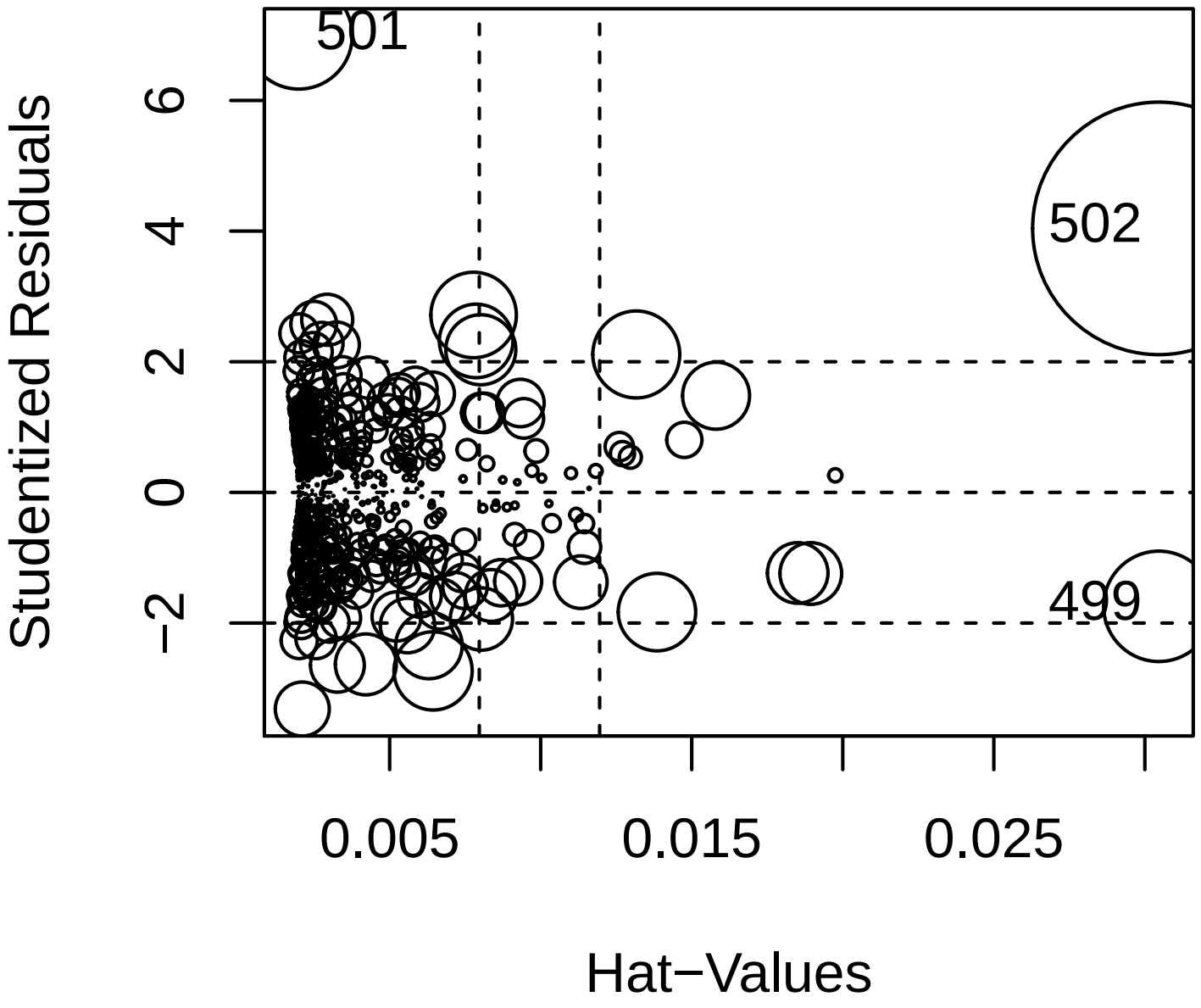
```
##     rstudent unadjusted p-value Bonferonni p
## 501 6.991280          8.7797e-12   4.4074e-09
## 502 4.041096          6.1598e-05   3.0922e-02
```

Remember that some outliers affect our model more than others. `influencePlot()` will give us a nice picture of which outliers we need to worry about.

Outliers and Leverage

- Hat-values: A measure of the amount of influence each data point has on the outcome predictions.
- Residuals: The difference between the observed and predicted value of the outcome variable.
- Studentized residuals: Scaled residuals, such that they have mean $= 0$ and variance $= 1$.
- Cook's distance: A measure of the effect of each data point on the regression coefficients.
- `influencePlot()` gives reference lines at
  - vertical: `mean(.hat)*c(2,3)`
  - horizontal: 0 and ± 2 SDs

Outliers and Leverage



Outliers and Leverage