StatQuest

Josh Turner

Linear Regression

We want to minimize the square of the distance between the observed values and the line

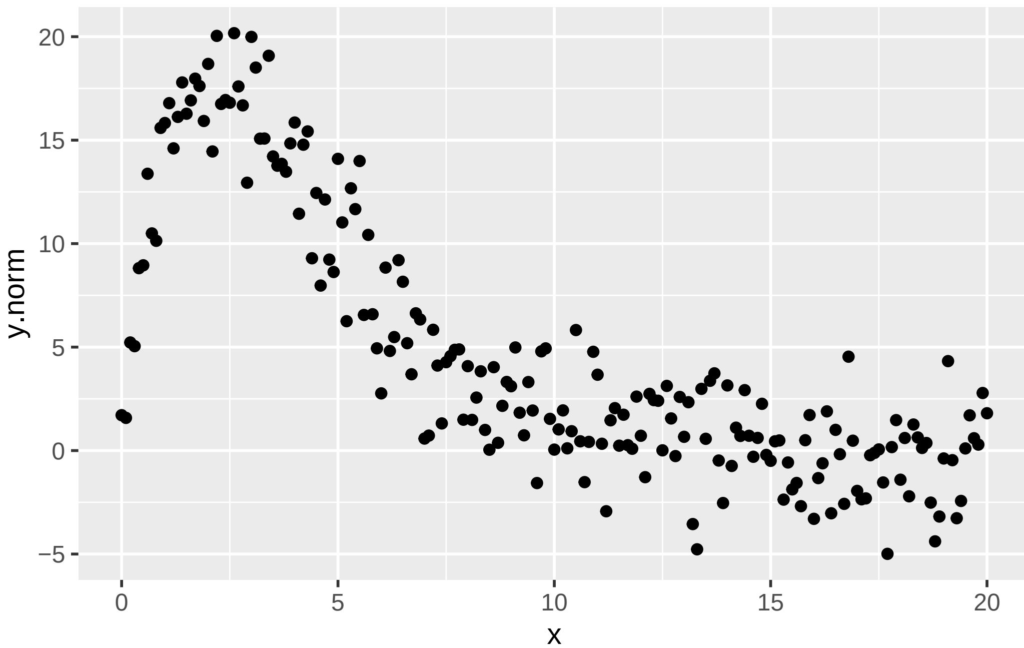
`lowess` vs `loess` smoothing

1. Use a type of sliding window to divide the data into smaller blobs.
2. At each data point, use a type of least squares to a line.

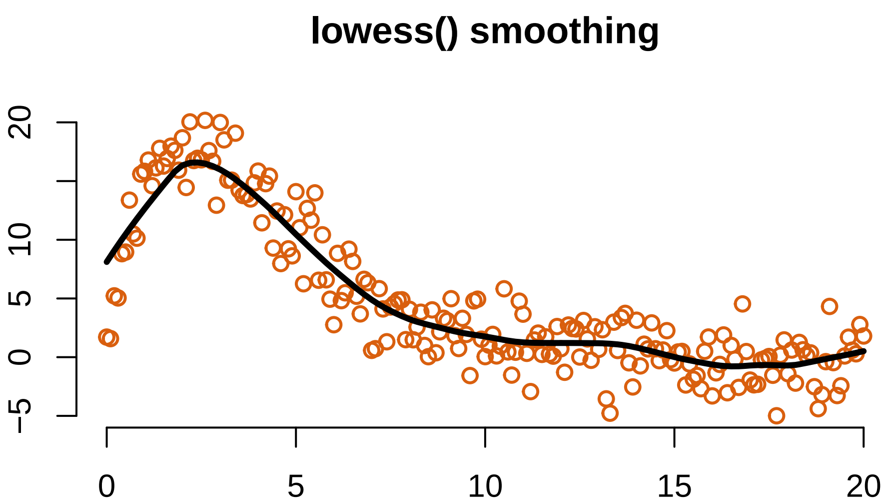
`lowess`	`loess`
Only fits line	Can fit a line or a parabola, default is fitting a parabols
Doesn't draw confidence interval around the curve	Draws confidence interval around the curve

```r
# noisy gamma distribution plot
x <- seq(from = 0, to = 20, by = 0.1)
y.gamma <- dgamma(x, shape = 2, scale = 2)
y.gamma.scaled <- y.gamma * 100

y.norm <- vector(length = 201)
for (i in 1:201) {
  y.norm[i] = rnorm(
      n = 1,
      mean = y.gamma.scaled[i],
      sd = 2
  )
}

data <- data.frame(x, y.norm)
```

```r
data |>
  ggplot(mapping = aes(
    x,
    y.norm
  )) +
  geom_point()
```

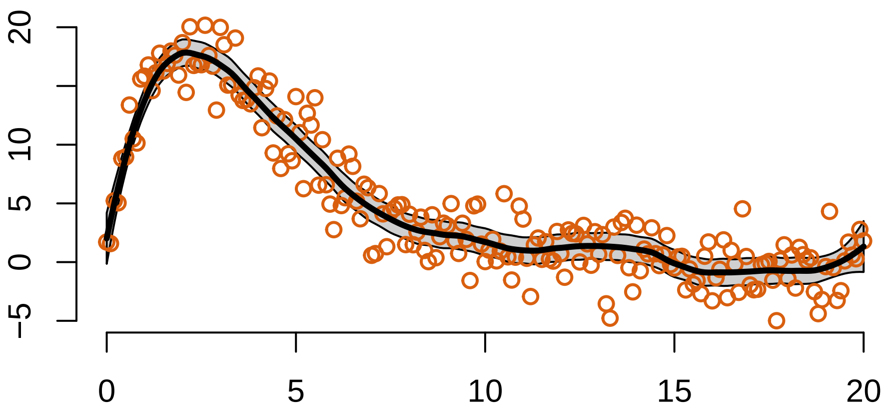



```r
# by default "lowess()" fits a line in each window using 2/3's
# of the data points
# y.norm ~ x says that y.norm is being modeled by x
# f is the fraction of points to use in each window

lo.fit.gamma <- lowess(y.norm ~ x, f = 1/5)
plot(data,
     frame.plot = F,
     xlab = "",
     ylab = "",
     col = "#d95f0e",
     lwd = 1.5
)
lines(x, lo.fit.gamma$y, col = "black", lwd = 3)
title("lowess() smoothing")
```



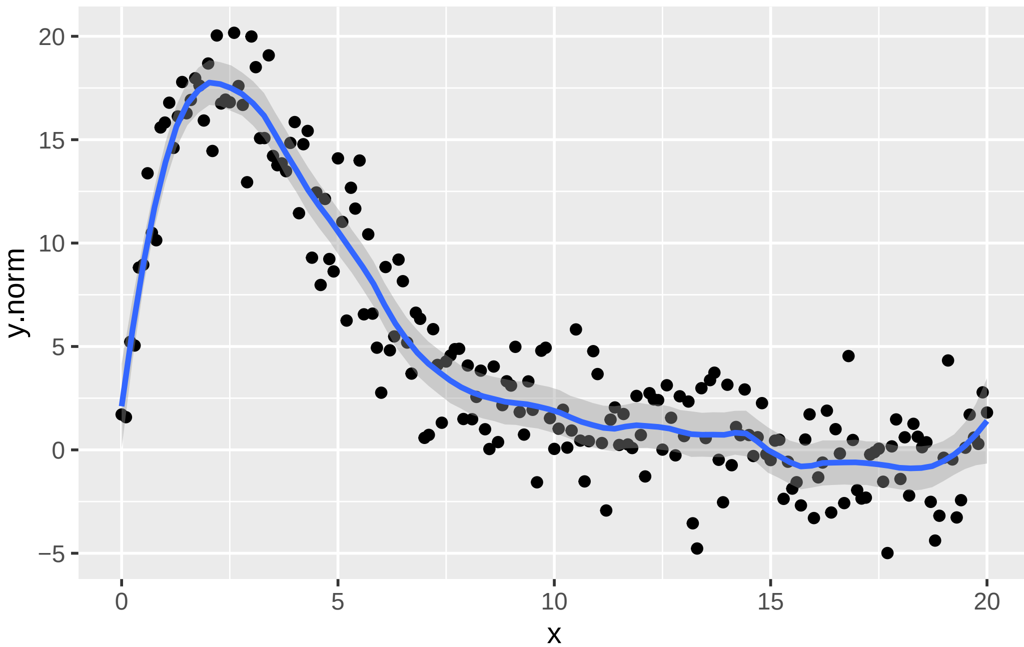

```
# by default "loess()" fits a parabola in each window using
# 75% of the data points

plx<-predict(
  loess(y.norm ~ x,
        span=1/5,
        degree=2,
        family="symmetric",
        iterations=4
  ),
  se=T
)

## Now let's add a confidence interval to the loess() fit...
plot(
  data,
  type="n",
  frame.plot=FALSE,
  xlab="",
  ylab="",
  col="#d95f0e",
  lwd=1.5
)
polygon(
  c(x, rev(x)),
  c(
    plx$fit + qt(0.975,plx$df)*plx$se,
    rev(plx$fit - qt(0.975,plx$df)*plx$se)
```

```
  ),
  col="#99999977"
)
points(
  data,
  col="#d95f0e",
  lwd=1.5
)
lines(
  x,
  plx$fit,
  col="black",
  lwd=3
)
```




```
data |>
  ggplot(aes(x, y.norm)) +
  geom_point() +
  geom_smooth(span = 1/5)
```

```
`geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



General Linear Models - Part 1

There are at least three parts to it,

1. Use least squares to fit a line to the data
2. Calculate R^2
3. Calculate a p-value for the R^2

The distance from a line to a data point is called a "**residual**". Fitting the line to the data is fairly straightforward. Pick a random line. Then adjust the line's parameters so that it better fits the data.

In the form, $y = a + bx$, if the b is non-zero, that means if a particular observation of x is provided, it is possible to make a guess about the y value. Quantification of how good the guess is can be achieved by two values,

1. R^2
2. p-value for the R^2

SS(mean) $\rightarrow$ Sum of Squares around the mean (of the responses, y) $= (\text{data} - \text{mean})^2$

Var(mean) $= \frac{SS(\text{mean})}{n}$

SS(fit) $\rightarrow$ Sum of Squares around the least-squares fit $= (\text{data} - \text{fit})^2$

Var(fit) $= \frac{SS(\text{fit})}{n}$

If and when $Var(\text{mean}) > Var(\text{fit})$, some of the variation is explained by the least-squared line. R^2 tells us how much of the variation in mouse size can be explained by taking mouse weight into account. $R^2 = \frac{Var(\text{mean}) - Var(\text{fit})}{Var(\text{mean})}$. For example, if $R^2 = 60\%$, it is said that x explains 60% of the variation in y.

```r
## Here's the data from the video
mouse.data <- data.frame(
  weight=c(0.9, 1.8, 2.4, 3.5, 3.9, 4.4, 5.1, 5.6, 6.3),
  size=c(1.4, 2.6, 1.0, 3.7, 5.5, 3.2, 3.0, 4.9, 6.3))

mouse.data # print the data to the screen in a nice format
```

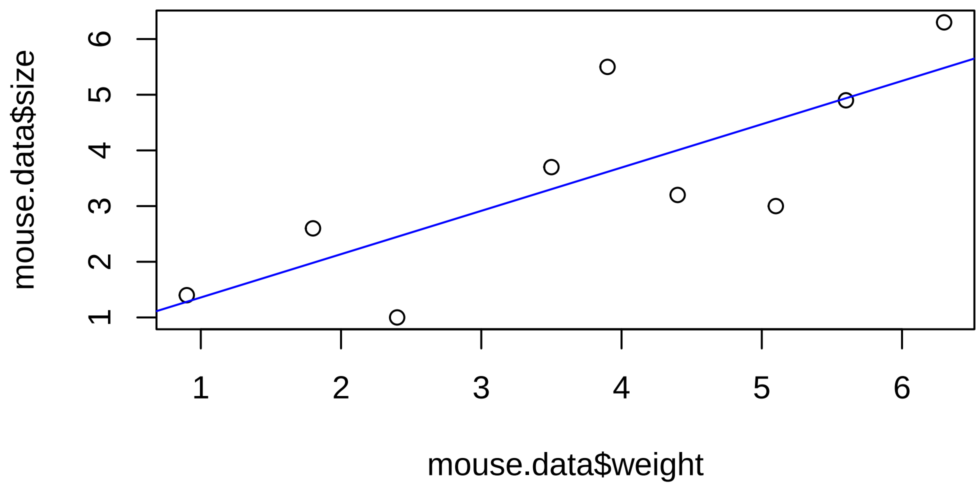
```
  weight size
1    0.9  1.4
2    1.8  2.6
3    2.4  1.0
4    3.5  3.7
5    3.9  5.5
6    4.4  3.2
7    5.1  3.0
8    5.6  4.9
9    6.3  6.3
```

```r
## plot a x/y scatter plot with the data
plot(mouse.data$weight, mouse.data$size)

## create a "linear model" - that is, do the regression
mouse.regression <- lm(size ~ weight, data=mouse.data)
## generate a summary of the regression
summary(mouse.regression)
```

```
Call:
lm(formula = size ~ weight, data = mouse.data)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.5482 -0.8037  0.1186  0.6186  1.8852 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
```

```
(Intercept)   0.5813     0.9647   0.603   0.5658
weight        0.7778     0.2334   3.332   0.0126 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.19 on 7 degrees of freedom
Multiple R-squared:  0.6133,    Adjusted R-squared:  0.558
F-statistic:  11.1 on 1 and 7 DF,  p-value: 0.01256
```

```r
## add the regression line to our x/y scatter plot
abline(mouse.regression, col="blue")
```



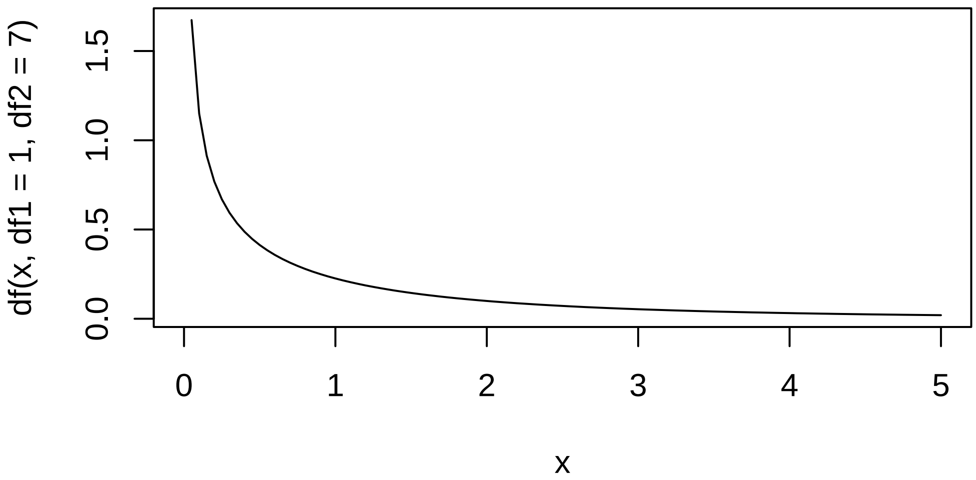
Equations with more parameters will never make SS(fit) worse than equation with fewer parameters.

R^2 is the proportion of variance described by the model. The p-value of R^2 comes from F-statistic,

$$F = \frac{\text{The variation of } y \text{ explained by } x}{\text{The variation of } y \text{ not explained by } x}$$

$$F = \frac{(SS(\text{mean}) - SS(\text{fit}))/(p_{\text{fit}} - p_{\text{mean}})}{SS(\text{fit})/(n - p_{\text{fit}})}$$

If the fit is good, F-statistic will be large. If the fit is small, F-statistic will be small. To get the p-value from this F-statistic, look at the F-curve of DF $= p_{\text{mean}} - p_{\text{fit}}$ and DF $= n - p_{\text{fit}}$. Area of the more extreme portion is the p-value. For example, F-curve of DF = 1 and 7 is,



The p-value will be smaller when there are more samples relative to the number of parameters in the fit equation.