Bias Variance Tradeoff

Matt Richey

2/11/2019

Introduction

The Bias-variance tradeoff is a statement of expected values. It links together the Mean Squared Error, the Variance of the prediction, the square of the bias of the prediction, and the inherent "noise" in the modeling process. The true model has the form

$$y = f(x) + \epsilon$$

where ϵ we will assume is normally distributed with mean 0 and variance σ^2. Our goal is to estimate $f(x)$ with algorithm which we will call $\hat{f}(x)$.

Imagine a fixed prediction value x_0 and a realized value $y_0 = f(x_0) + \epsilon$. Bias-variance tradeoff says:

$$E[(y_0 - \hat{f}(x_0))^2] = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

Let's look closely at each of these components. Keep in mind where the randomness is: the training set used to construct $\hat{f}$ and the realized value y_0.

- $E[(y_0 - \hat{f}(x_0))^2]$ is the expected (average) value $(y_0 - \hat{f}(x_0))^2$.
- $\text{Var}(\hat{f}(x_0))$ is the variance of the values $\hat{f}(x_0)$ generated from each training set.
- $\text{Bias}(\hat{f}(x_0))]$. The Bias of $\hat{f}(x_0)$ is the expected value of $\hat{f}(x_0) - f(x_0)$ over the training data.
- The last term is the noise, in this case $\text{Var}(\epsilon) = \sigma^2/$

Keep in mind, the randomness in all of these scenarios comes from the training sets and the realized value of y_0. To model this effect we will be repeatedly generating random training data sets and random values of $y_0 = f(x_0) + \epsilon$

Bias-Variance of a linear model.

Let's build some synthetic data with a known underlying (true) model.

A collection of underlying "true" functions.

```
f1 <- function(x) 1+x
f2 <- function(x) (x-1)*(x+1)
f3 <- function(x) x+ sin(5*pi*x)
f4 <- function(x) -x+sqrt(3)*sin(pi^(3/2)*x^2)
```

Pick one to use.

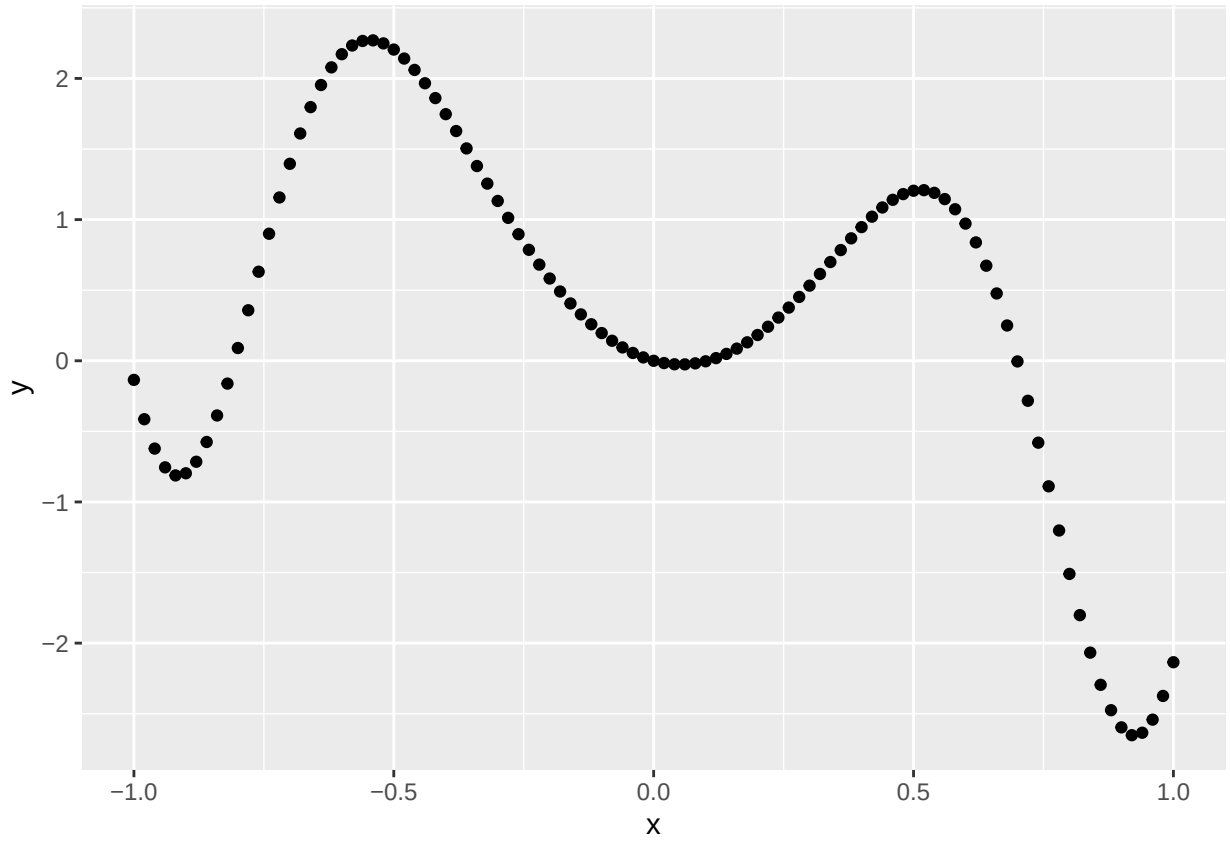
```
f <- f4
```

For convenience, let's have a fixed set of input values to use for predictions and other tasks.

```
K <- 101
## Range
xMin <- -1
xMax <- 1
xVals<-seq(xMin, xMax,length=K)
trueF.df <- data.frame(x=xVals,y=f(xVals))
```

Here's what we are looking at for an underlying model

```
trueF.df %>%
  ggplot()+
  geom_point(aes(x,y))
```



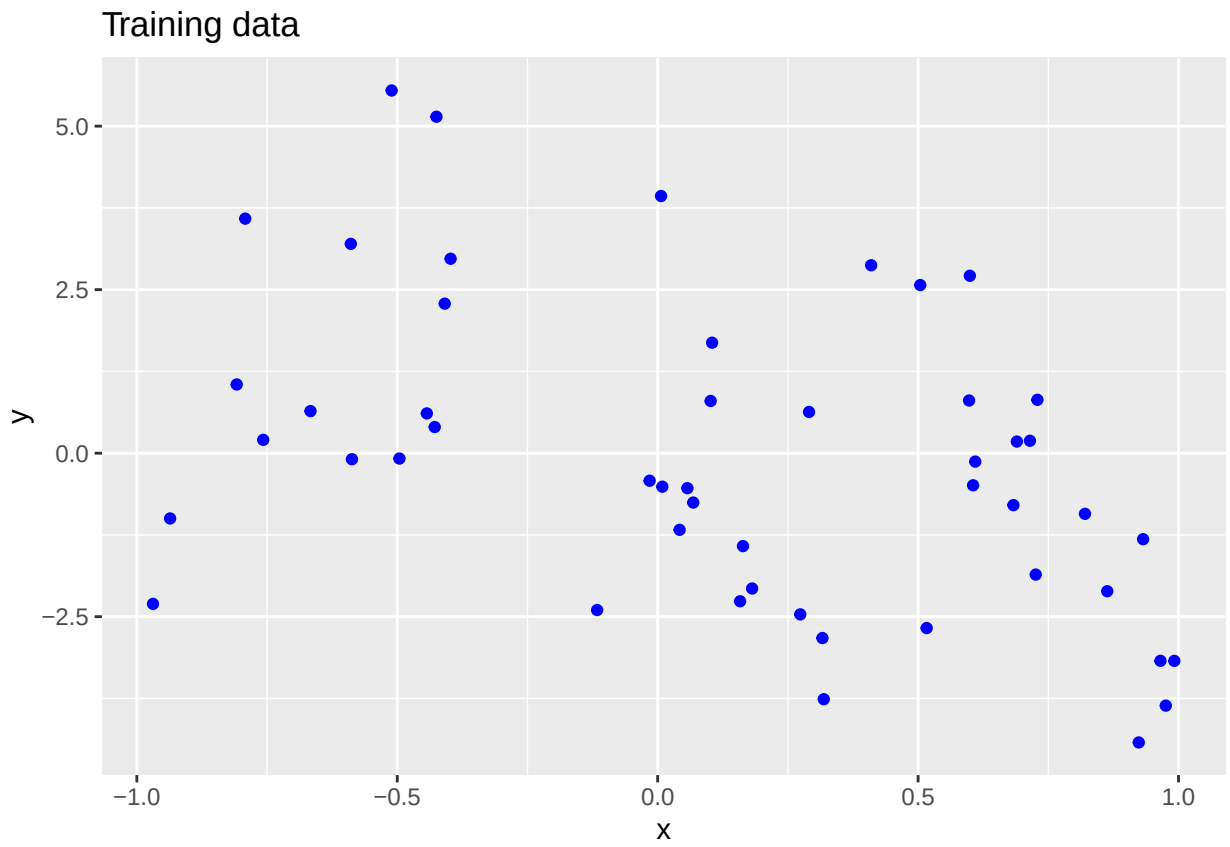
We can generate training data by adding some "noise".

```
sizeDS <-50 # number of data points
sig <-1.75 # for the noise

##predictor
x<-runif(sizeDS,xMin, xMax) # inputs
## Repsonse
y<-f(x)+rnorm(sizeDS,0,sig) #realized values f(x)+noise
## Put in a data frame
train.df<-data.frame(x,y)
```

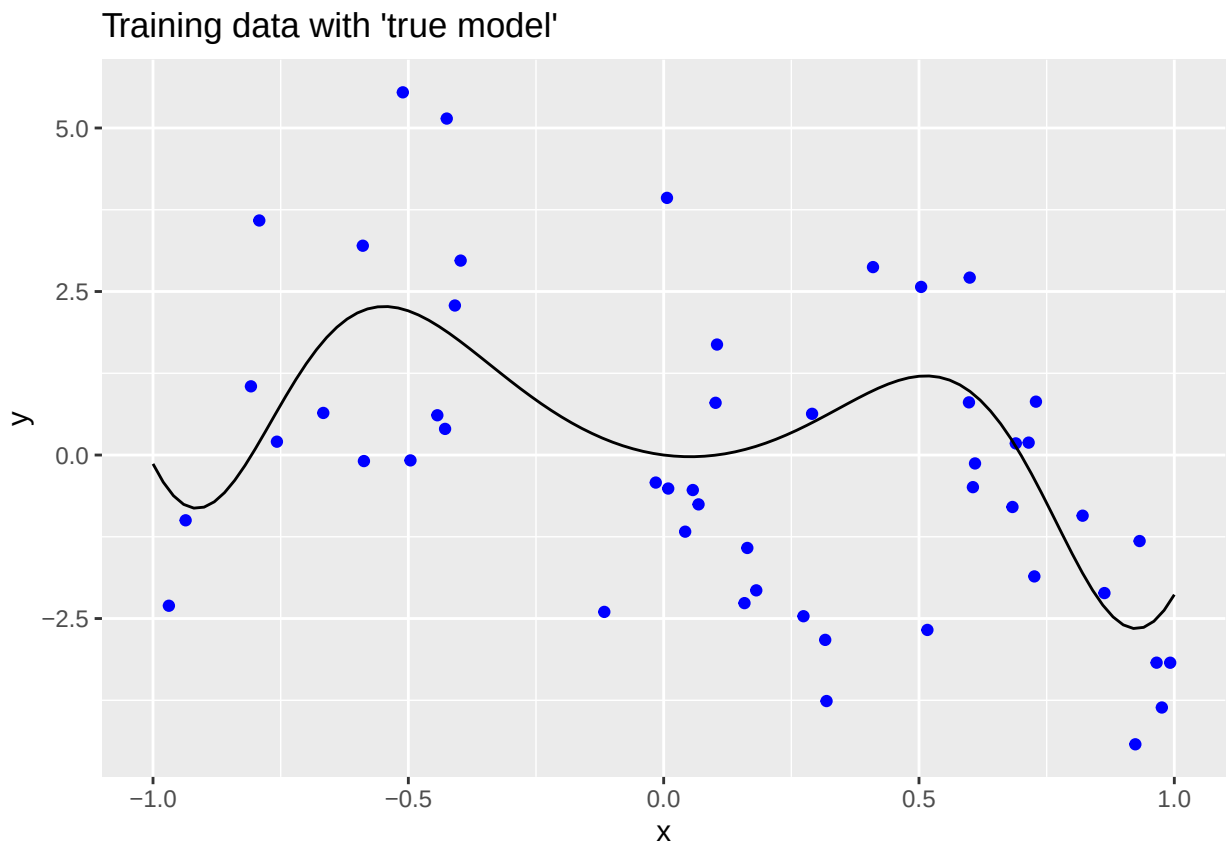
Plot the data

```
train.df %>%
  ggplot()+
  geom_point(aes(x,y),color="blue")+
  ggtitle("Training data")
```



For what it's worth, here's the same plot with the underlying "true" f(x).

```
train.df %>%
  ggplot()+
  geom_point(aes(x,y),color="blue")+
  geom_line(data=trueF.df,aes(x,y),color="black")+
  ggtitle("Training data with 'true model'")
```

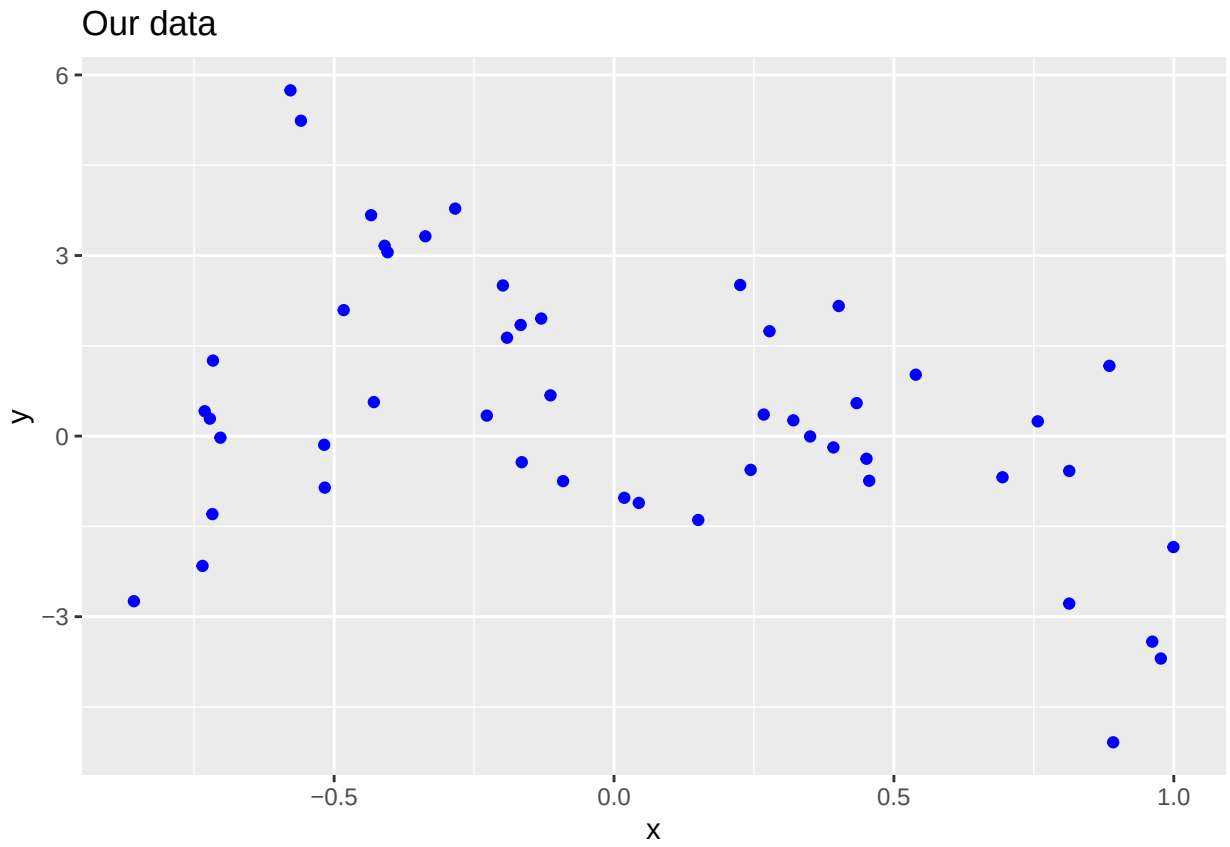


We are going to be repeating the process of building training data, so make a simple function

```
buildData <- function(func,sizeDS,sig,xMin = -1, xMax = 1){
  ##predictor
  x<-runif(sizeDS,xMin, xMax) # inputs
  ## Repsonse
  y<-func(x)+rnorm(sizeDS,0,sig) #realized values f(x)+noise
  ## Put in a data frame
  data.frame(x,y)
}
```

Now we can build and plot the data very easily.

```
buildData(f,sizeDS,sig) %>%
 ggplot()+
  geom_point(aes(x,y),color="blue")+
  labs(title="Our data")
```



A linear model Now build a simple linear model.

The data

```
train.df <- buildData(f,sizeDS,sig)
```

The linear model

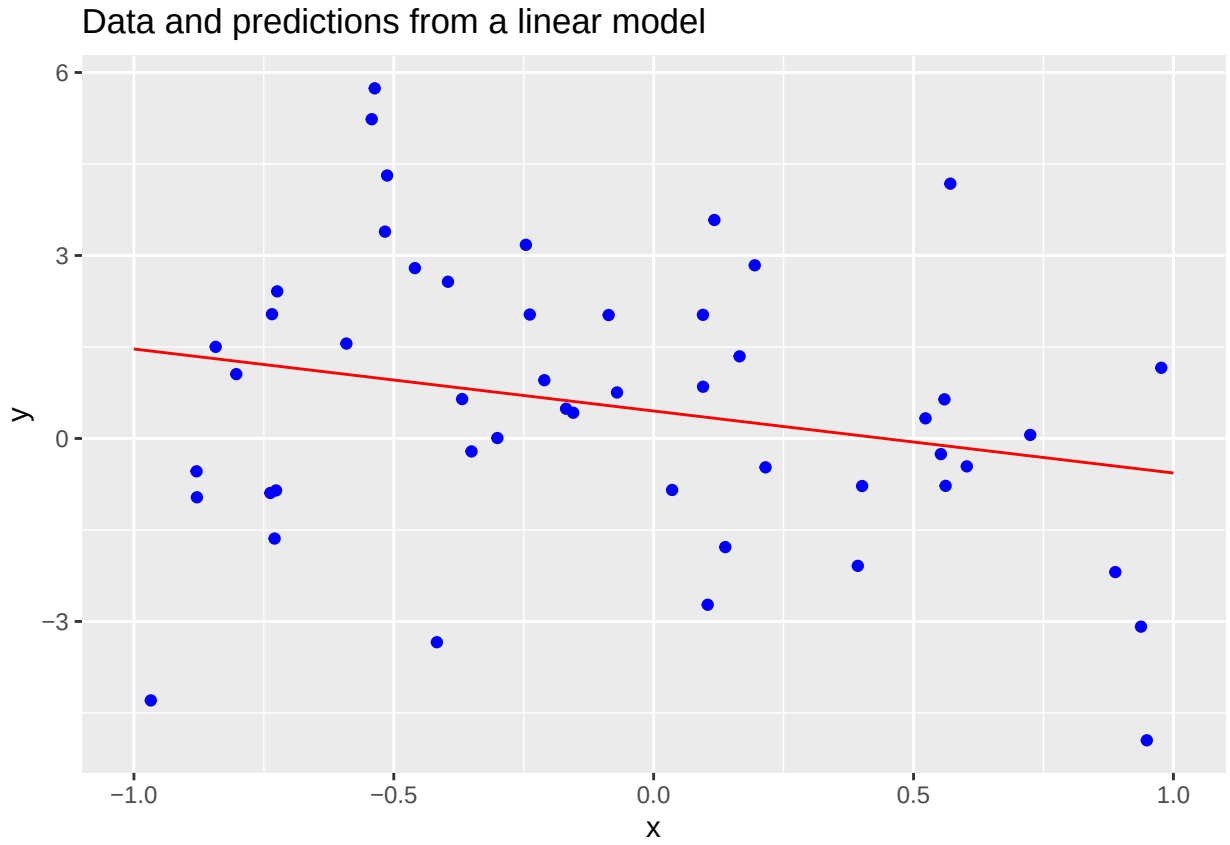
```
mod <- lm(y ~ x, data = train.df)
```

The predictions from the model

```
predVals<-predict(mod,newdata=data.frame(x=xVals))
```

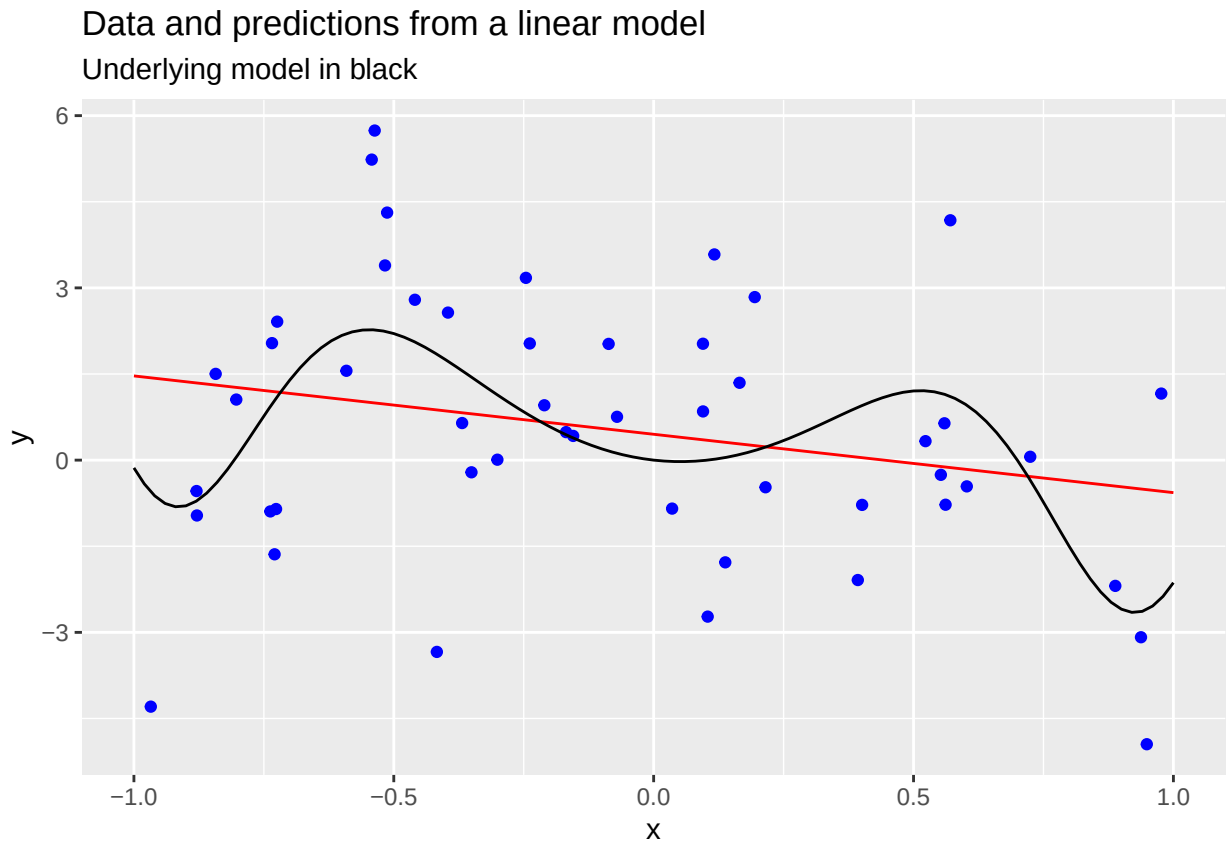
Plot the prediction and the data

```
ggplot()+
  geom_point(data=train.df,aes(x,y),color="blue")+
  geom_line(data=NULL,aes(x=xVals,y=predVals),color="red")+
  labs(title="Data and predictions from a linear model")
```



Add the underlying model for reference. In reality, you never know what this is.

```
ggplot()+
  geom_point(data=train.df,aes(x,y),color="blue")+
  geom_line(data=NULL,aes(x=xVals,y=predVals),color="red")+
  geom_line(data=trueF.df,aes(x,y),color="black")+
  labs(title="Data and predictions from a linear model",
       subtitle="Underlying model in black")
```



Variablility.

We will repeat this process a large number of times, each time using new (random) data set and then producing a linear model

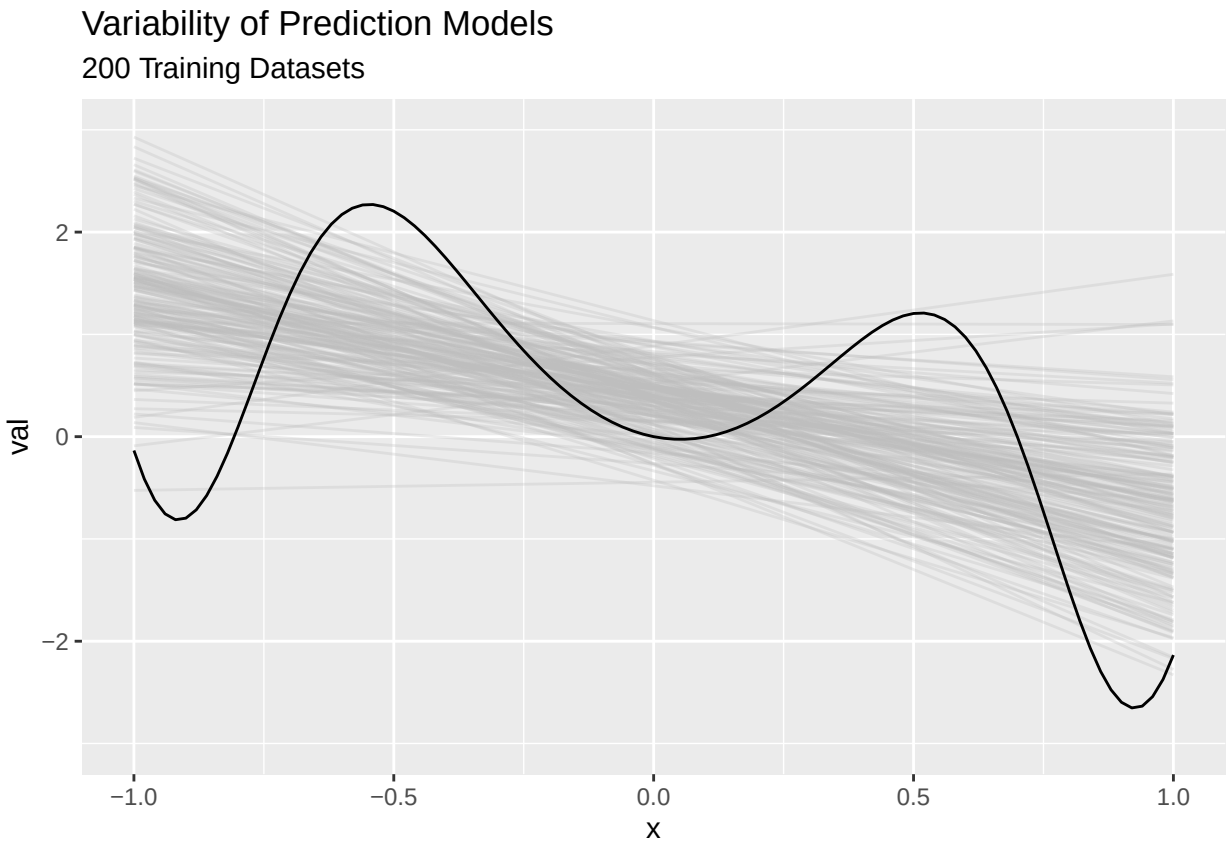
```
numDS <- 200 ##number of runs
##Place to put all the predictions
allVals <- matrix(nrow=K,ncol=numDS+1)
allVals[,1] <- xVals ##the xvalues

for(m in 1:numDS) {
  train.df <- buildData(f,sizeDS, sig)
  ##mod <- lm(y~1,data=train.df)
  mod <- lm(y ~ x, train.df)
  ##mod <- lm(y ~ x + I(x ^ 2), data = train.df)
  ##mod <- lm(y ~ x + I(x ^ 2) + I(x ^ 3) + I(x ^ 4) + I(x ^ 5), data = train.df)
  ##mod <- lm(y~x+I(x^2)+I(x^3)+I(x^4)+I(x^5)+I(x^6)+I(x^7)+I(x^8)+I(x^9)+I(x^10)+I(x^11)+I(x^12),train
  pred <- predict(mod, newdata = data.frame(x = xVals))
  allVals[, m + 1] <- pred
}

allVals.df <- data.frame(allVals)
names(allVals.df) <- c("x", paste0("run", 1:numDS))
```

What does it look like?

```
lim0 <- 3
allVals.df %>%
  gather(run,val,2:(numDS+1)) %>%
  ggplot()+
  geom_line(aes(x,val,group=run),color="grey",alpha=.3)+
  geom_line(data=trueF.df,aes(x,y),color="black")+
  scale_y_continuous(limits=c(-lim0,lim0))+
  labs(title="Variability of Prediction Models",
       subtitle=sprintf("%s Training Datasets",numDS))
```



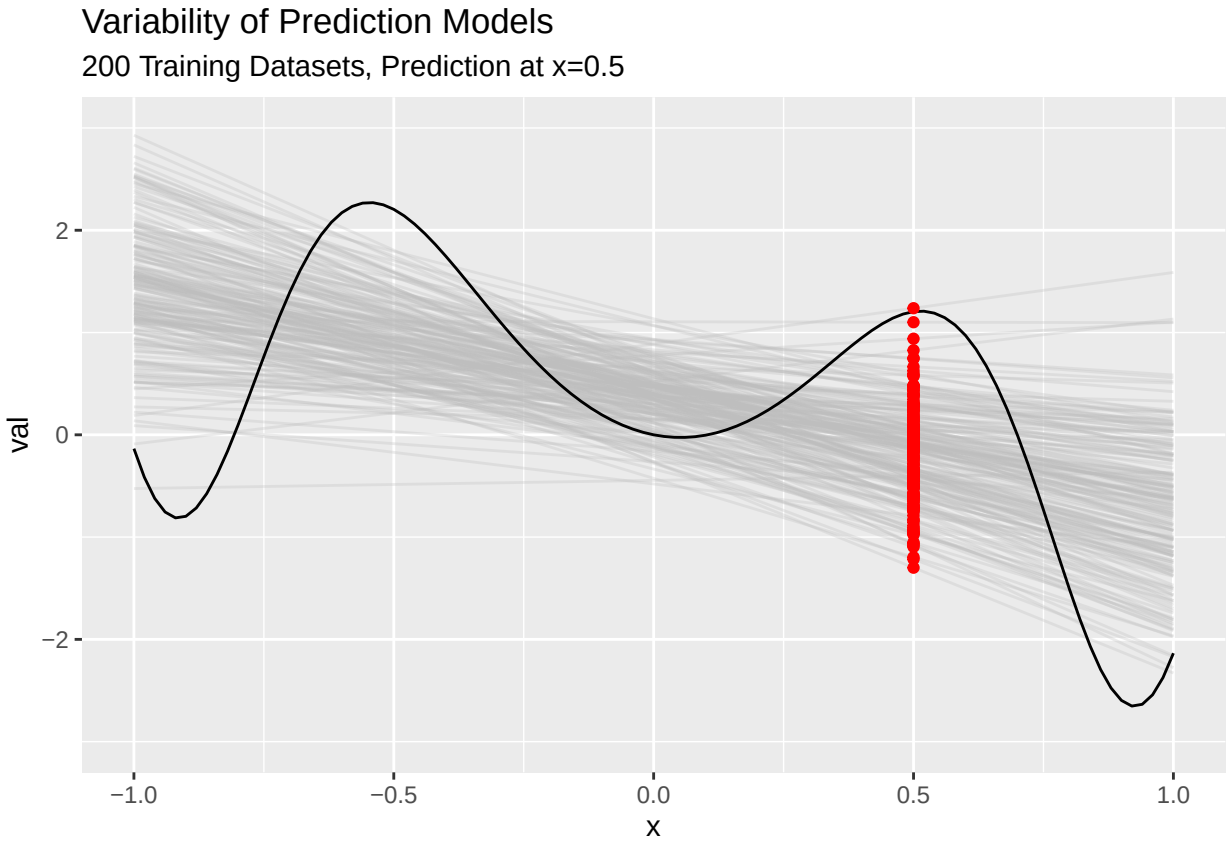
Predictions at a point x0

Yank out all the predictions at a particular value x0 = 0.5

```
x0 <- 0.5
predVals0.df <- allVals.df %>%
  filter(x == x0) %>%
  gather(run,y,2:(numDS+1)) %>%
  select(-run)
```

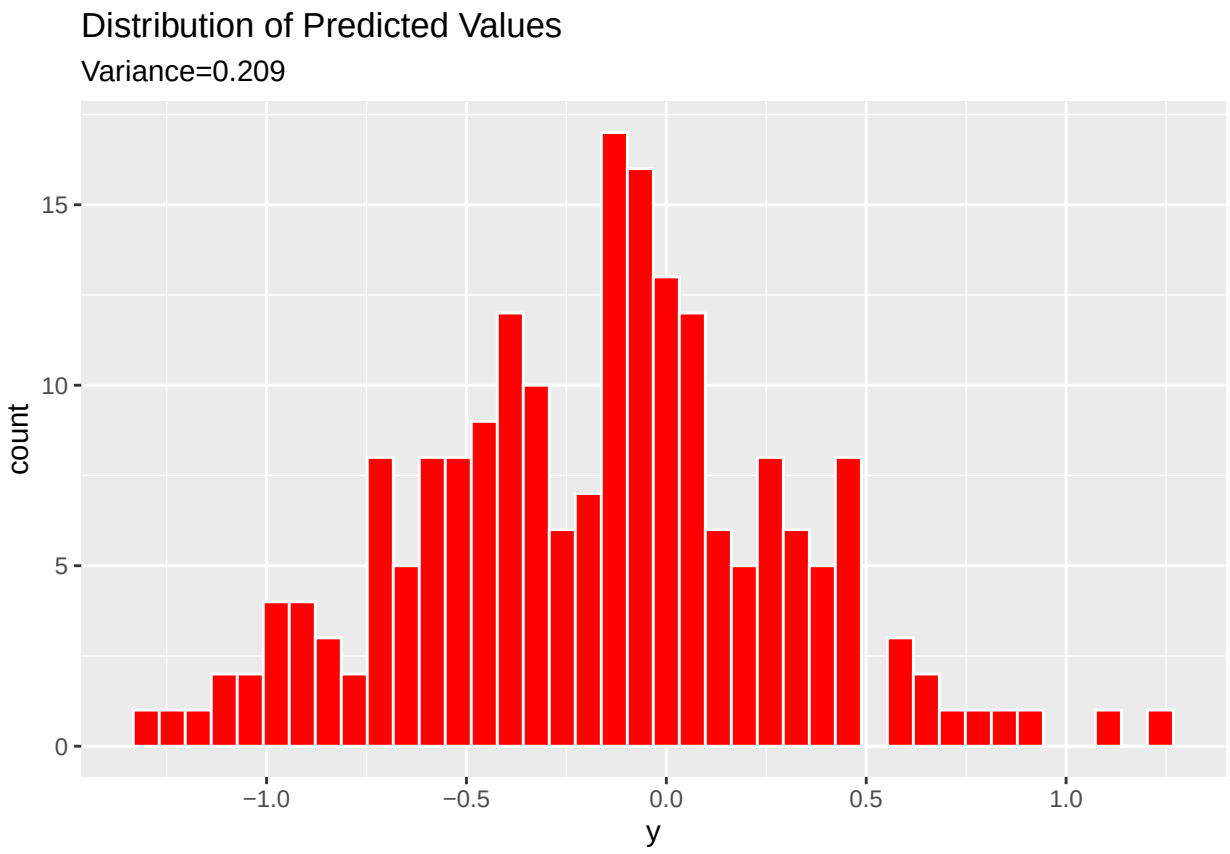
Add the predicted values to our plot.

```
allVals.df %>%
  gather(run,val,2:(numDS+1)) %>%
  ggplot()+
  geom_line(aes(x,val,group=run),color="grey",alpha=.3)+
  geom_line(data=trueF.df,aes(x,y),color="black")+
  geom_point(data=predVals0.df,
             aes(x,y),color="red")+
  scale_y_continuous(limits=c(-lim0,lim0))+
  labs(title="Variability of Prediction Models",
       subtitle=sprintf("%s Training Datasets, Prediction at x=%s",numDS,x0))
```



How variable are these point predictions?

```
theVar <- with(predVals0.df,var(y))
predVals0.df %>%
  ggplot()+
  geom_histogram(aes(y),color="white",fill="red",bins=40)+
  labs(title=sprintf("Distribution of Predicted Values"),
                     subtitle=sprintf("Variance=%s",round(theVar,3)))
```



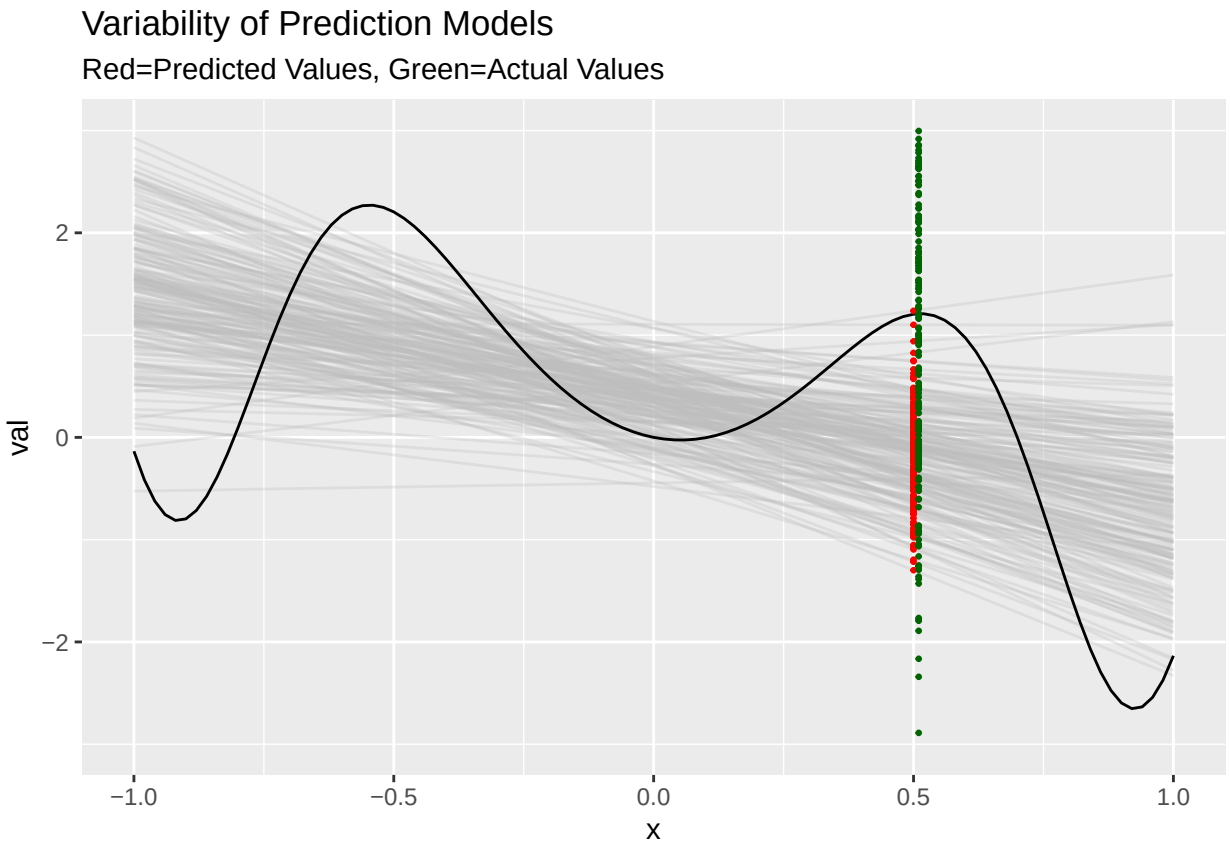
What about the "true" values, i.e, y=f(x)+noise. Notice we can generate these independently of everything else we've done up to this point.

```
trueVals.df <- data.frame(x=x0,y=f(x0)+rnorm(numDS,0,sig))
```

Add these values to the plot

```
allVals.df %>%
  gather(run,val,2:(numDS+1)) %>%
  ggplot()+
  geom_line(aes(x,val,group=run),color="grey",alpha=.3)+
  geom_line(data=trueF.df,aes(x,y),color="black")+
  geom_point(data=predVals0.df,
             aes(x,y),color="red",size=.5)+
  geom_point(data=trueVals.df,aes(x=x+.01,y),color="darkgreen",size=.5)+
  scale_y_continuous(limits=c(-lim0,lim0))+
  labs(title="Variability of Prediction Models",
       subtitle="Red=Predicted Values, Green=Actual Values")
```

```
## Warning: Removed 27 rows containing missing values (geom_point).
```



This is an important plot, it shows how the variability of the prediction process is related to the variabililty of the underlying process that produces the data.

Estimating the MSE, Variance, and Bias.

We can now emperically verify the Bias-Variance Tradeoff Equation

$$MSE = E[(y_0 - \hat{f}(x_0))^2] = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

- MSE: The average squared difference between the true value and the predicted value at x0 = 0.5.

MSE

Start be computing the loss function, i.e., the mean squared error between the predicted values and the true values

```
mse <- bind_cols(predVals0.df,trueVals.df) %>% ##combine these two
  mutate(sqDiff=(y-y1)^2) %>%
  with(mean(sqDiff))
mse
```

```
## [1] 4.394821
```

Now the variance of the predicted values.

```
varPred <- with(predVals0.df,var(y))
varPred
```

```
## [1] 0.2091277
```

The Bias (squared).

The bias is how far the predicted values are from the true values. It's similar to squared error, only with out the square.

```
bias2 <- bind_cols(predVals0.df,trueVals.df) %>%
  mutate(diff=(y-y1)) %>%
  with(mean(diff)^2)
bias2
```

```
## [1] 1.424128
```

Lastly, the variance of the noise. This is just sig^2. It's fixed by what is missing from the model.

```
varNoise <- sig^2
varNoise
```

```
## [1] 3.0625
```

How did we do?

$$MSE = E[(y_0 - \hat{f}(x_0))^2] = \text{Var}(\hat{f}(x_0)) + [\text{Bias}(\hat{f}(x_0))]^2 + \text{Var}(\epsilon).$$

```
c(mse,
varPred+bias2+varNoise)
```

```
## [1] 4.394821 4.695756
```

Very close! (As it should be.)

You should repeat this with different values of sig and perhaps more repititions (numDS).

Making this smoother

```
numDS <- 2000 ##number of runs
##Place to put all the predictions
allVals <- matrix(ncol=2,nrow=numDS)

for(m in 1:numDS){
  mod <- lm(y~x,buildData(f,sizeDS,sig))
  pred <- predict(mod,newdata=data.frame(x=x0))
  allVals[m,1] <- pred
  }
allVals[,2] <- f(x0)+rnorm(numDS,0,sig)

allVals.df <- data.frame(pred=allVals[,1],true=allVals[,2])
```

```
(mse <- with(allVals.df,mean((pred-true)^2)))
```

```
## [1] 5.051922
```

```
(var0 <- with(allVals.df,var(pred)))
```

```
## [1] 0.1612237
```

```
(bias2 <- with(allVals.df,mean(pred-true))^2) ##careful: bias^2 is the square of the bias
```

```
## [1] 1.835218
```

```
(noise <- sig^2)
```

```
## [1] 3.0625
```

In the end,

```
c(mse,var0+bias2+noise)
```

```
## [1] 5.051922 5.058942
```

Build a function that computes all these values in one fell swoop.

```
biasVarTO <- function(sizeDS,numDS,x0){
  allVals <- matrix(ncol=2,nrow=numDS)
  for(m in 1:numDS){
    mod <- lm(y~x,buildData(f,sizeDS,sig))
    pred <- predict(mod,newdata=data.frame(x=x0))
    allVals[m,1] <- pred
  }
  allVals[,2] <- f(x0)+rnorm(numDS,0,sig)

  allVals.df <- data.frame(pred=allVals[,1],true=allVals[,2])
  mse <- with(allVals.df,mean((pred-true)^2))
  var0 <- with(allVals.df,var(pred))
  bias2 <- with(allVals.df,mean(pred-true))^2 ##careful: bias^2 is the square of the bias
  noise <- sig^2
  c(mse,var0,bias2,noise)
}
```

Try it out a few times. Use a data set with, say, 100 samples and run, say, 200 separate repititions of the modeling process.

```
sizeDS <- 50
numDS <- 200
(vals <- biasVarTO(sizeDS,numDS,x0))
```

```
## [1] 5.9830801 0.1594678 2.3656133 3.0625000
```

```
c(vals[1],sum(vals[2:4]))
```

```
## [1] 5.983080 5.587581
```

Again, close enough.

Let's do this several times.

```
L <- 10
allVals <- matrix(nrow=L,ncol=4)
sizeDS <- 50
numDS <- 200
for(i in 1:L){
  allVals[i,] <- biasVarTO(sizeDS,numDS,x0)
}

data.frame(mse=allVals[,1],
                 var=allVals[,2],
                 bias2=allVals[,3],
                 noise=allVals[,4]) %>%
  mutate(tot=var+bias2+noise)
```

```
##         mse       var    bias2  noise      tot
## 1  4.443316 0.1824190 1.360680 3.0625 4.605599
## 2  5.430924 0.1527196 2.326067 3.0625 5.541286
## 3  5.029993 0.1657678 1.700045 3.0625 4.928313
## 4  5.078139 0.1668643 1.604061 3.0625 4.833425
## 5  5.055748 0.1574929 1.719375 3.0625 4.939368
## 6  4.464177 0.1720527 1.418826 3.0625 4.653379
## 7  5.570117 0.1753409 1.741260 3.0625 4.979101
## 8  4.882153 0.1838154 1.510203 3.0625 4.756518
## 9  4.384666 0.1726122 1.515858 3.0625 4.750970
## 10 4.984337 0.1890719 1.571444 3.0625 4.823016
```

Your turn: Adding More Flexibility

We could repeat this process by adding more flexibility to our model. For a linear model, this usually means adding higher order powers of the input variables.

For example, here's a quadratic model.

```
train.df <- buildData(f,sizeDS,sig)
mod2 <- lm(y ~ x + I(x ^ 2),data=train.df)
summary(mod2)
```

```
##
## Call:
## lm(formula = y ~ x + I(x^2), data = train.df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
```

```
## -4.1005 -1.3884  0.0243  0.9493  5.1759
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   1.6401     0.4480   3.661 0.000635 ***
## x            -1.0332     0.5157  -2.004 0.050895 .
## I(x^2)       -3.0655     1.0043  -3.052 0.003730 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.099 on 47 degrees of freedom
## Multiple R-squared:  0.2071, Adjusted R-squared:  0.1733
## F-statistic: 6.137 on 2 and 47 DF,  p-value: 0.004286
```

Yo can repeat everything above with the more flexible model. Perhaps the most expeditiouus way to proceed to is simply modify the biasVarTO function to account for more flexibility (i.e., change the one line where the model is created).

```
biasVarTO2 <- function(sizeDS,numDS,x0){
  allVals <- matrix(ncol=2,nrow=numDS)
  for(m in 1:numDS){
    mod <- lm(y~x+I(x^2)+I(x^3),buildData(f,sizeDS,sig))
    pred <- predict(mod,newdata=data.frame(x=x0))
    allVals[m,1] <- pred
  }
  allVals[,2] <- f(x0)+rnorm(numDS,0,sig)

  allVals.df <- data.frame(pred=allVals[,1],true=allVals[,2])
  mse <- with(allVals.df,mean((pred-true)^2))
  var0 <- with(allVals.df,var(pred))
  bias2 <- with(allVals.df,mean(pred-true))^2
  noise <- sig^2
  c(mse,var0,bias2,noise)
}
```

How the quadratice model look?

```
L <- 10
allVals <- matrix(nrow=L,ncol=4)
for(i in 1:L){
  allVals[i,] <- biasVarTO2(100,200,x0)
}

data.frame(mse=allVals[,1],
                 var=allVals[,2],
                 bias2=allVals[,3],
                 noise=allVals[,4]) %>%
  mutate(tot=var+bias2+noise)
```

```
##         mse        var    bias2  noise      tot
## 1  4.544964 0.13433231 2.129508 3.0625 5.326340
## 2  4.521130 0.09517919 1.432298 3.0625 4.589977
## 3  4.250485 0.14102288 1.052428 3.0625 4.255951
```

```
## 4  3.987522 0.13166275 1.429596 3.0625 4.623759
## 5  5.318458 0.12955986 1.528830 3.0625 4.720889
## 6  5.015357 0.12384702 1.986678 3.0625 5.173025
## 7  4.229684 0.11525323 1.465573 3.0625 4.643326
## 8  4.123399 0.11474395 1.303088 3.0625 4.480332
## 9  4.521193 0.14487946 1.635407 3.0625 4.842786
## 10 4.043580 0.11574610 1.317174 3.0625 4.495421
```

At a glance, the MSE appears to have decreased. What about the variance, bias^2?

Generalize even more

Build a nifty function that allows us to include any "formula."

A formula is something of the form "y ~ x +I(x) + ..."

```
biasVarTO3 <- function(form,sizeDS,numDS,x0){
  allVals <- matrix(ncol=2,nrow=numDS)
  for(m in 1:numDS){
    ##the
    mod <- lm(formula(form),buildData(f,sizeDS,sig))
    pred <- predict(mod,newdata=data.frame(x=x0))
    allVals[m,1] <- pred
  }
  allVals[,2] <- f(x0)+rnorm(numDS,0,sig)

  allVals.df <- data.frame(pred=allVals[,1],true=allVals[,2])
  mse <- with(allVals.df,mean((pred-true)^2))
  var0 <- with(allVals.df,var(pred))
  bias2 <- with(allVals.df,mean(pred-true))^2
  noise <- sig^2
  c(mse,var0,bias2,noise)
}
```

Here's the linear version

```
myForm <- "y ~x "
biasVarTO3(myForm, sizeDS, numDS, 0.5)
```

```
## [1] 5.9043461 0.1549244 1.9836079 3.0625000
```

Here's the quadratice

```
myForm <- "y ~x + I(x^2) "
biasVarTO3(myForm, sizeDS, numDS, 0.5)
```

```
## [1] 4.770757 0.147116 1.668821 3.062500
```

Cubic....

```
myForm <- "y ~x + I(x^2) + I(x^3) "
biasVarTO3(myForm, sizeDS, numDS, 0.5)
```

```
## [1] 5.1251906 0.2142452 1.6295108 3.0625000
```

Let's do a whole bunch of these, say up to degree=15.

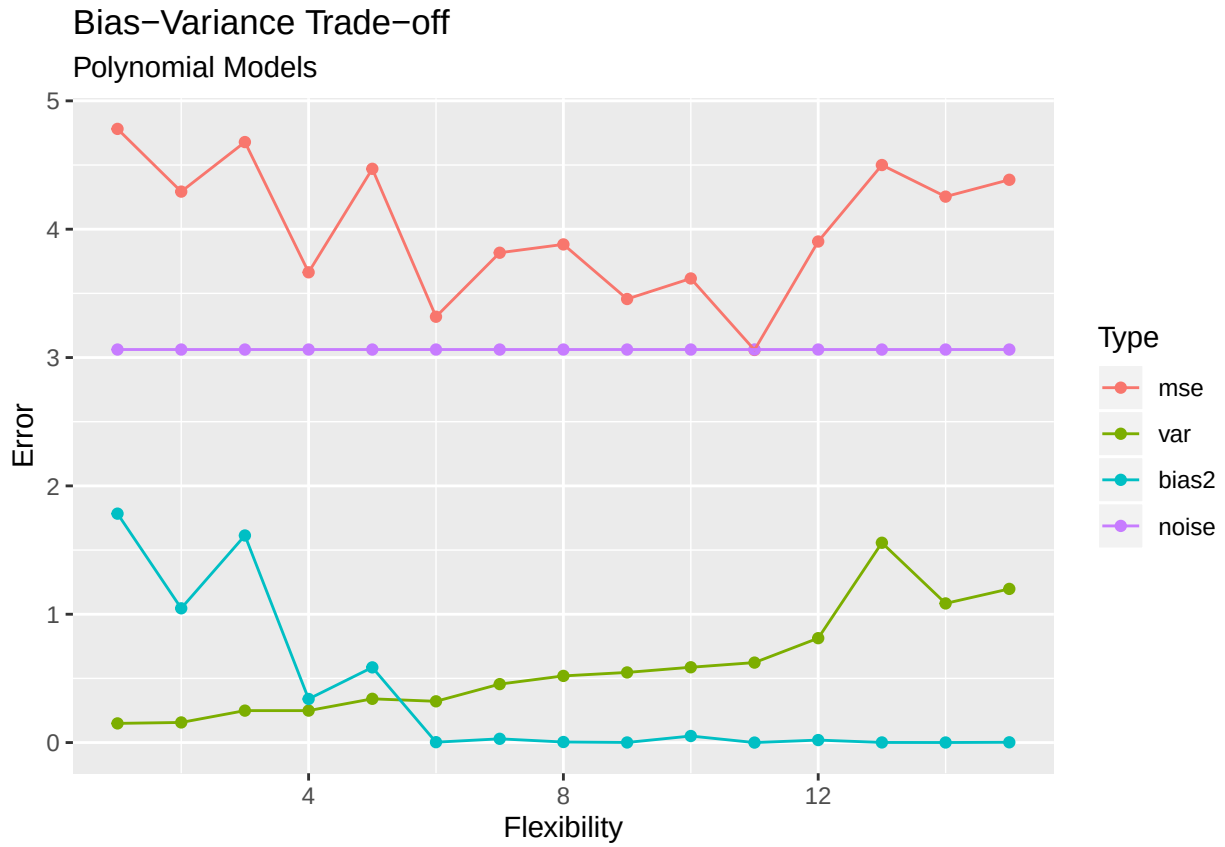
```
sizeDS <- 50
numReps <- 250 ## Increase this for more accuracy of the estimates
##Starter Formula
form0 <- "y ~ "
maxDegree <- 15
##A place to stash the results
res <- matrix(nrow=maxDegree,ncol=4)
for(k in 1:maxDegree){
  ##Build up the formula
  form0 <- sprintf("%s + I(x^%s)",form0,k)
  print(form0)
  res[k,] <- biasVarTO3(form0,sizeDS,numReps,0.5)
  print(res[k,])
}
```

```
## [1] "y ~  + I(x^1)"
## [1] 4.781584 0.149418 1.783724 3.062500
## [1] "y ~  + I(x^1) + I(x^2)"
## [1] 4.2928249 0.1567847 1.0460305 3.0625000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3)"
## [1] 4.6789943 0.2489285 1.6136118 3.0625000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4)"
## [1] 3.6641940 0.2490698 0.3396485 3.0625000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5)"
## [1] 4.4704796 0.3408295 0.5858179 3.0625000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6)"
## [1] 3.318037145 0.321473157 0.003002927 3.062500000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7)"
## [1] 3.81673855 0.45550447 0.02919384 3.06250000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8)"
## [1] 3.881525639 0.519351976 0.004269687 3.062500000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9)"
## [1] 3.4561276757 0.5464303548 0.0008210062 3.0625000000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I(x^10)"
## [1] 3.61608211 0.58675664 0.05102241 3.06250000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I(x^10)"
## [1] 3.059255e+00 6.231367e-01 1.183512e-06 3.062500e+00
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I(x^10)"
## [1] 3.90421773 0.81337235 0.01947006 3.06250000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I(x^10)"
## [1] 4.4999991433 1.5567739065 0.0009247388 3.0625000000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I(x^10)"
## [1] 4.2538571796 1.0841830463 0.0004853465 3.0625000000
## [1] "y ~  + I(x^1) + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7) + I(x^8) + I(x^9) + I(x^10)"
## [1] 4.385207593 1.197379197 0.002068722 3.062500000
```

Build a plot from this information.

```
res.df <- data.frame(flex=1:maxDegree,res)
names(res.df) <- c("flex","mse","var","bias2","noise")

res.df %>%
  gather(Type,err,mse:noise) %>%
  ##put these in order
  mutate(Type=factor(Type,levels=c("mse","var","bias2","noise"))) %>%
  ggplot()+
  geom_point(aes(flex,err,color=Type))+
    geom_line(aes(flex,err,color=Type))+
  labs(x="Flexibility",
       y="Error",
       title="Bias-Variance Trade-off",
      subtitle="Polynomial Models")
```



It looks as if the minimal MSE occurs somewhere around degree=4 (or maybe a bit larger).

Note: compared to the "cartoon" Figure 2.12, this picture is probably be pretty jerky.

Assignments

Build a stand-alone RMarkdown document that contains both of these exercises. Make each exercise a separate section, with subsections describing the steps you go through to complete the assignment.

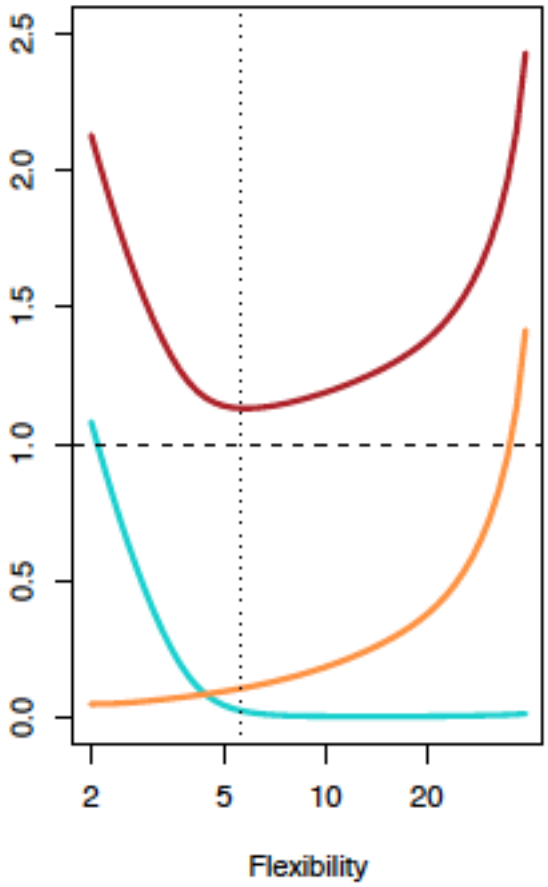


Figure 1: Figure 2.12

Assignment 1

- For the underlying "true" model, use polynomials of degree =1..4. In each case, repeat the process above with linear models of degree up to about 15 or so. Does the "optimal" flexibility correspond to the degree of the underlying true model?

Of course, build a ggplot version of Figure 2.12 of ISLR using your results from above.

Assignment 2

- Repeat using KNN regression (i.e., using knn.reg). Note that in this case, flexibility increases as the control paramenter k (=number of neighbors) decreases. Again, your goal is to build a version of Figure 2.12 of ISLR. Note, we usually put flexibilty on the horizontal axis with the lowest flexibility on the left and the highest on the right.

Be careful, the knn.reg function requires that you put the input data is a very specific form. This was described in the RMarkdown document from the first day of class.