06 - Supervised Learning

SYS 4582/6018 | Spring 2019

06-supervised.pdf

Contents

1 Supervised Learning Intro

1.1 Required R Packages

We will be using the R packages of:

- `FNN` for k nearest neighbor models
- `tidyverse` for data manipulation and visualization

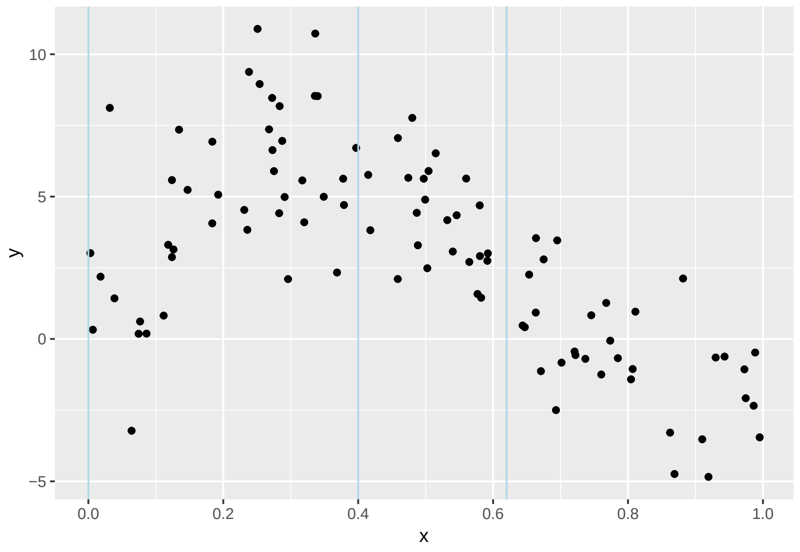
```r
library(FNN)
library(tidyverse)
```

1.2 Supervised Learning

- Up to this point, the data we used to for association analysis, network analysis, density estimation, clustering, and anomaly detection did not have any labels or target states we were trying to predict.
  - These are all examples of *unsupervised learning*
- In *supervised learning*, each observation can be partitioned into two sets: the predictor/independent/feature variables and the target/labels/response/dependent variable(s).
- Usually the predictor variables are represented by X and the response variables represented by Y
- The goal in supervised learning is to find the patterns and relationships between the predictors, X, and the response, Y.
  - Usually the goal is to *predict* the value of Y given X.

2 Example Data

Consider some data $D = \{(X_i, Y_i)\}_{i=1}^n$ with $Y_i \in \mathbb{R}$, $X_i \in [0, 1]$ and $n = 100$.



Your Turn #1

The goal is to predict new Y values if we are given the X's.
- If $x = .40$, predict Y.
- If $x = 0$, predict Y.

- If $x = .62$, predict Y.
- How should we build a *model* that will automatically predict Y?

3 Linear Models

- Linear models refer to a class of models where the output (predicted value) is a linear combination (weighted sum) of the input variables

$$f(x;\beta) = \beta_0 + \sum_{j=1}^{p} \beta_j x_j$$

where $x = [x_1, \ldots, x_p]^\mathsf{T}$ is a vector of features/variables/attributes and $\hat{Y}|x = f(x;\hat{\beta})$ is the predicted response at $X = x$

- the coefficients (or weights), $\hat{\beta}$ are often selected by minimizing the squared residuals of the *training data* (may also be described as *ordinary least squares*)
  - But, there are other (and better) ways to estimate the parameters in linear regression that we will discuss later in the course

3.1 Simple Linear Regression

- single predictor variable $x \in \mathbb{R}$
- $f(x;\beta) = \beta_0 + \beta_1 x$
- Use *training data*: $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^n$
- OLS uses the weights/coefficients that minimize the RSS loss function over the training data

$$\hat{\beta} = \arg\min_{\beta} \ \text{RSS}(\beta)$$

- where RSS is the *residual sum of squares*

$$\begin{aligned} \text{RSS}(\beta) &= \sum_i^n (y_i - f(x_i,\beta))^2 \\ &= \sum_i^n (y_i - \beta_0 - \beta_1 x_i)^2 \\ &= \sum_i^n \hat{\epsilon}_i^2 \qquad \text{where } \hat{\epsilon}_i = y_i - \hat{y}_i \text{ is the residual} \end{aligned}$$

- The solutions are

$$\begin{aligned} \hat{\beta}_0 &= \bar{y} - \beta_1 \bar{x} \\ \hat{\beta}_1 &= \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \end{aligned}$$

- Definitions:

$$\begin{aligned} \text{MSE}(\beta) &= \frac{1}{n}\text{RSS}(\beta) \\ &= \frac{1}{n}\sum_{i=1}^n (y_i - f(x_i;\beta))^2 \\ \text{RMSE} &= \sqrt{\text{MSE}} = \sqrt{\text{RSS}}/\sqrt{n} \end{aligned}$$

3.2 OLS Linear Models in R

3.2.1 Estimation with `lm()`

In R , the function `lm()` fits an OLS linear model

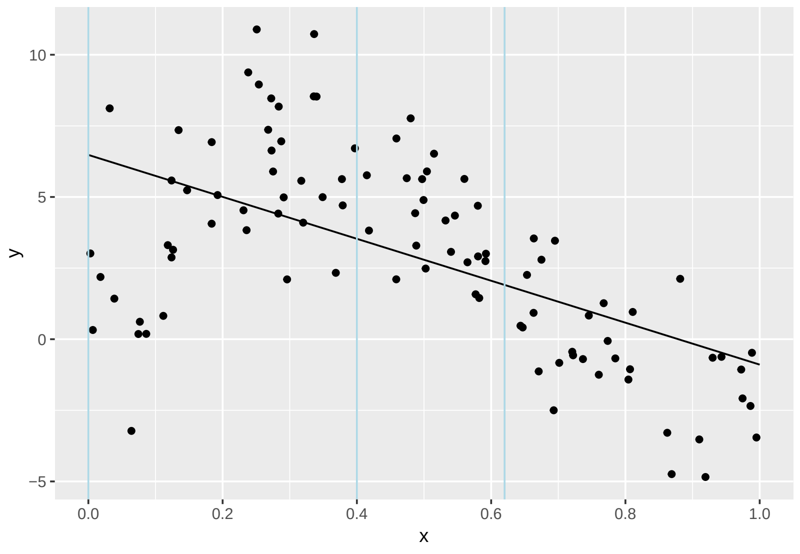
```
data.train = data.frame(x,y)
m1 = lm(y~x, data=data.train) # fit simple OLS
summary(m1)                   # summary of model
#>
#> Call:
#> lm(formula = y ~ x, data = data.train)
#>
#> Residuals:
#>    Min     1Q Median     3Q    Max
#> -9.229 -1.635  0.019  1.940  6.728
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept)    6.478      0.584   11.09  < 2e-16 ***
#> x             -7.372      1.058   -6.97  3.7e-10 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.91 on 98 degrees of freedom
#> Multiple R-squared:  0.331,  Adjusted R-squared:  0.325
#> F-statistic: 48.6 on 1 and 98 DF,  p-value: 3.69e-10
```

- `lm()` uses the `formula` interface, which includes the intercept by default. Some examples here.

3.2.2 Prediction with `predict()`

The function `predict()` is used to get the predicted values.

```
xseq = seq(0, 1, length=200)         # sequence of equally spaced values from 0 to 1
xeval = data.frame(x = xseq)         # make into a data.frame object
yhat1 = predict(m1, newdata=xeval)   # vector of yhat's (predictions)
```



3.2.3 Questions

Your Turn #2

1. How did we do? If X_{new} is close to 0, or close to 0.4, or close to .62?
2. How to make it better?

4 Polynomial inputs

- In the *simple* linear regression model, we had 2 parameters that we needed to estimation, β_0 and β_1. Thus, the model complexity is minimal.
  - The only thing simpler is an intercept only model.
- But the data appears to have a more *complex* structure than linear.
- A *parametric approach* to add complexity is to incorporate *polynomial terms* into the model.
  - A quadratic model is $f(x;\beta) = \beta_0 + \beta_1 x + \beta_2 x^2$

4.1 Estimation

- OLS uses the weights/coefficients that minimize the RSS loss function over the training data

$$\begin{aligned}\hat{\beta} &= \arg\min_{\beta}\ \mathrm{RSS}(\beta) \qquad \text{Note: } \beta \text{ in this problem is a } \textit{vector}\\ &= \arg\min_{\beta} \sum_{i=1}^{n}(y_i - f(x_i;\beta))^2\\ &= \arg\min_{\beta} \sum_{i=1}^{n}(y_i - \beta_0 - \beta_1 x_i - \beta_2 x_i^2)^2\end{aligned}$$

4.1.1 Matrix notation

- **Model**

$$f(\mathbf{x};\beta) = \mathbf{x}^{\mathsf{T}}\beta$$

$$\mathbf{x} = \begin{bmatrix}1\\ x\\ x^2\end{bmatrix} \qquad \beta = \begin{bmatrix}\beta_0\\ \beta_1\\ \beta_2\end{bmatrix}$$

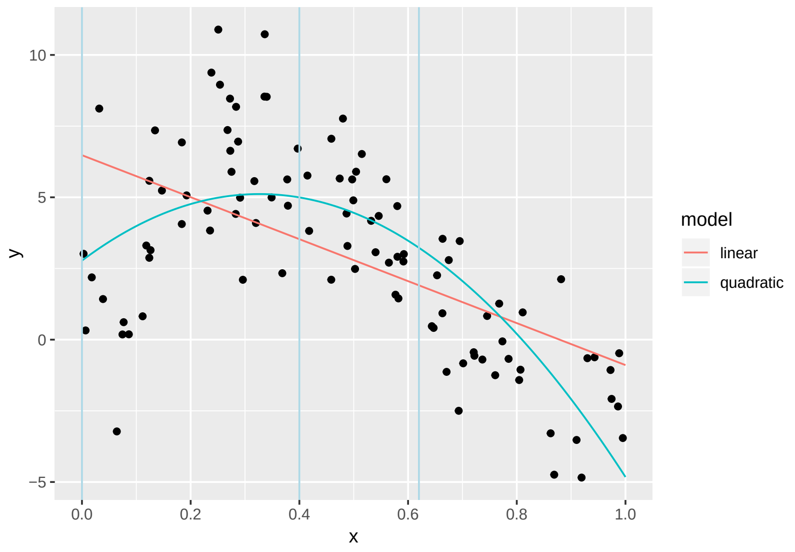
Your Turn #3 : Matrix Notation

Solve for $\hat{\beta}$ using matrix notation.

4.1.2 R implementation

In R , the function `poly()` is a convenient way to get polynomial terms

```
m2 = lm(y~poly(x, degree=2), data=data.train)
yhat2 = predict(m2, newdata=xeval)
```



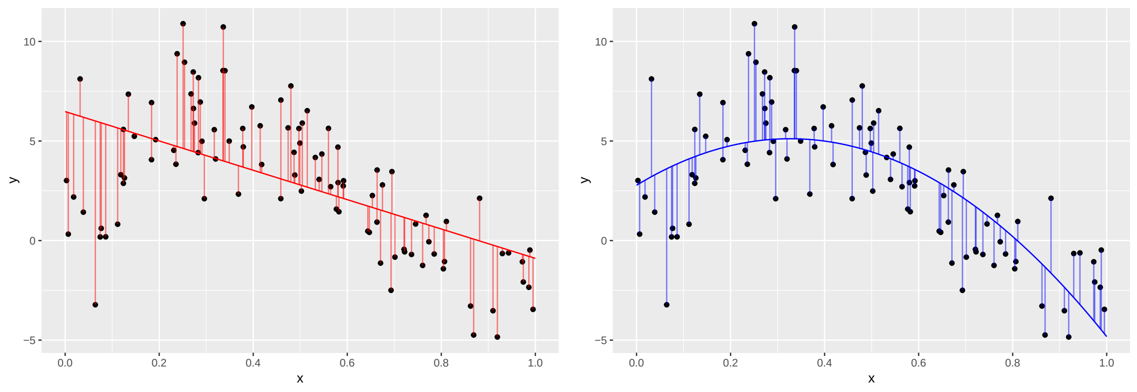
Your Turn #4

1. How did we do? If X_{new} is close to 0, or close to 0.4, or close to $.62$?
2. But does the quadratic model fit better *overall*?
3. What is the *complexity* of the quadratic model?

4.2 Performance Comparison (on Training Data)

Comparing the two models (according to RSS loss), the quadratic model does much better!

degree	RSS	npars
1	829.1	2
2	557.8	3



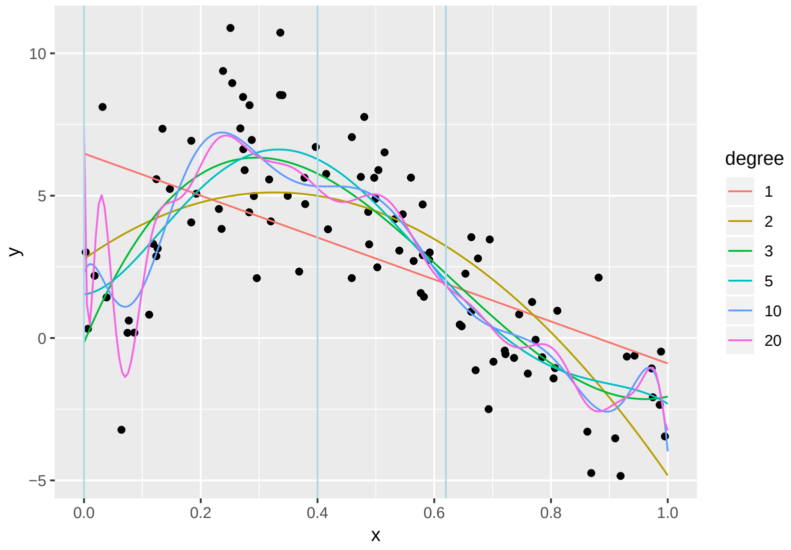
As my kids always reason, “if a little is good, than a lot must be better”. So why not try more complex models by increasing the polynomial degree.

- Polynomial of degree d

$$f_{\text{poly}}(x; \beta, d) = \beta_0 + \sum_{j=1}^{d} \beta_j x^j$$

degree	RSS	npars
1	829.1	2
2	557.8	3
3	428.1	4
5	410.2	6
10	364.9	11
20	315.6	21

And its always good to observe the plot



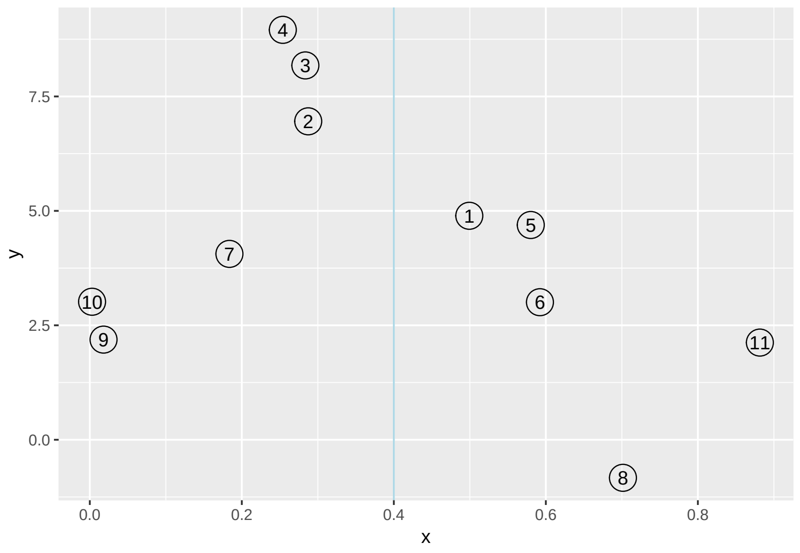
- For `degree=20`, the behavior at the end points are a bit erratic.
- Using a higher degree would further reduce the RSS, but the fitted curve would be less "smooth"

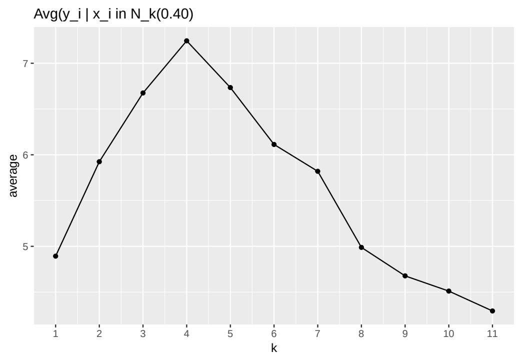
5 k-nearest neighbor models

- The k-NN method is a non-parametric *local* method, meaning that to make a prediction $\hat{y}|x$, it only uses the training data in the *vicinity* of x.
  - contrast with OLS linear regression, which uses all X's to get prediction.
- The model is simple to describe

$$\begin{aligned} f_{\text{knn}}(x;k) &= \frac{1}{k}\sum_{i:x_i \in N_k(x)} y_i \\ &= \text{Avg}(y_i \mid x_i \in N_k(x)) \end{aligned}$$

  - $N_k(x)$ are the set of k nearest neighbors
  - only the k closest y's are used to generate a prediction
  - it is a *simple mean* of the k nearest observations
  - What is the estimate if $k = n$?
- Consider the following example where we wish to estimate $Y \mid X = 0.40$





x	y	k	D	$\hat{f}_{\text{knn}}(x;k)$
0.50	4.89	1	0.10	4.89
0.29	6.96	2	0.11	5.92
0.28	8.18	3	0.12	6.68
0.25	8.95	4	0.15	7.25
0.58	4.69	5	0.18	6.73
0.59	3.00	6	0.19	6.11
0.18	4.06	7	0.22	5.82
0.70	-0.83	8	0.30	4.99
0.02	2.19	9	0.38	4.68
0.00	3.01	10	0.40	4.51
0.88	2.12	11	0.48	4.29

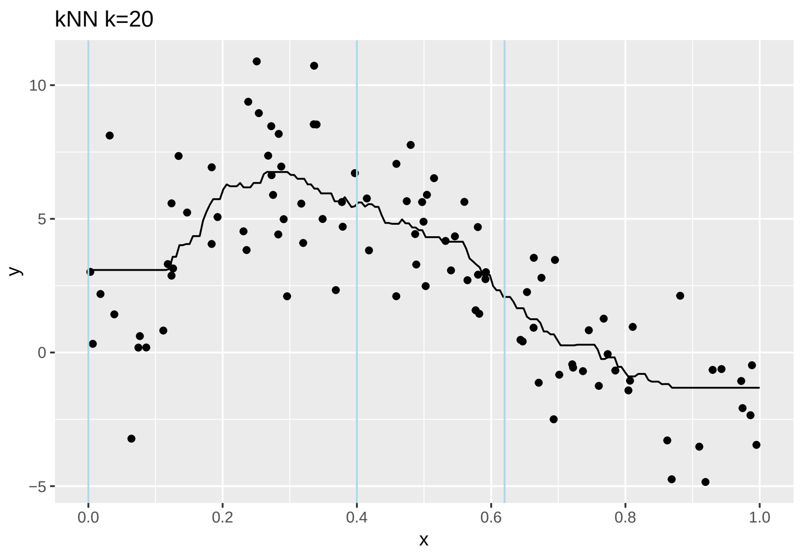
- One drawback of knn methods is that all the training data must be stored in order to make predictions
- Another drawback comes from the *curse of dimensionality* when the distance to neighbors grows exponentially

5.1 knn in action

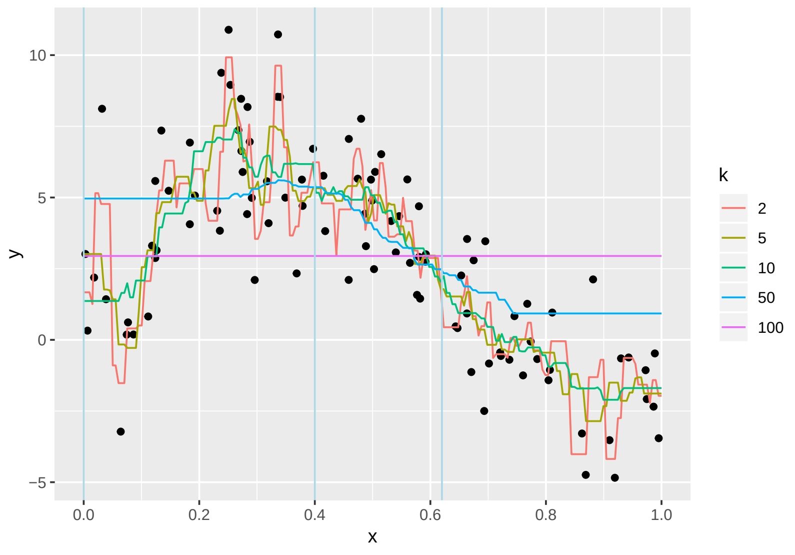
In R , the function `knn.reg()` from the `FNN` package will fit a knn regression model. Here is a $k = 20$ nearest neighbor model

```r
# install.packages("FNN")       # to install FNN package
library(FNN)                    # library() loads the package

#- fit a k=20 knn regression
knn.20 = knn.reg(data.frame(x), test=xeval, y=y, k=20)
```



- The *complexity* of a knn model increases as k decreases.
- The least model, which is a constant occurs when $k = n$
- The most complex model when $k = 1$
- The effective degrees of freedom or *edf* for a knn model is n/k
  - this is a measure of the model *complexity*. It is about how many parameters are estimated in the model (to allow comparison with parametric models)



5.1.1 Performance of the knn models (on training data)

k	RSS	edf
100	1240.2	1
50	686.5	2
10	386.1	10
5	316.0	20
2	183.9	50

6 Predictive Model Comparison (or how to choose the best model)

6.1 Predictive Model Evaluation

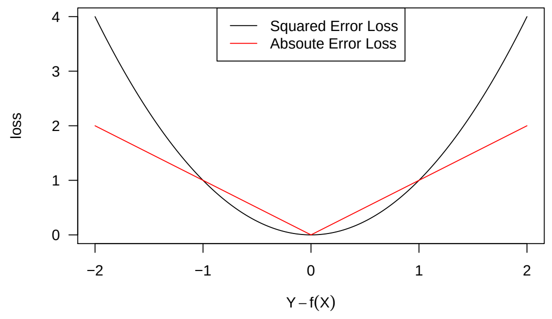
Our goal is prediction, so we should evaluate the models on their *predictive performance*.

- So we need to use new data (i.e., data not used to fit the model) to evaluate how well our models do in prediction
- Call these data *test data* $D_{\text{test}} = \{(X_j, Y_j)\}_{j=1}^{J}$
  - Note: assume that the test data comes from the same distribution as the training data
  - Or $P_{\text{test}}(X, Y) = P_{\text{train}}(X, Y)$
  - **both** Y and X from same distribution
- Later in the course we will cover ways to do this when we only have training data (e.g., cross-validation)
- but for today, we have an unlimited about of *test data* at our disposal (since we know how the data were generated)

6.2 Statistical Decision Theory

- In a prediction context, we want a *point estimate* for the value of an unobserved r.v. $Y \in \mathbb{R}$ given an input feature $X \in \mathbb{R}$.
- Let $f(X)$ be the prediction of Y given X.
- Define a *loss function* $L(Y, f(X))$ that indicates how bad it is if we estimate the value Y by $f(X)$
  - E.g. Y is the number of customers complaints in a call center and X is the day of week
  - If we guess $f(X) = 500$, but there are really $Y = 2000$, how bad would that be?
- A common loss function is *squared error*

$$L(Y, f(X)) = (Y - f(X))^2$$



- The best predictor is the one that minimizes the *expected loss* or Risk or Expected Prediction Error (EPE)

$$\text{Risk} = \text{EPE} = \text{E[loss]}$$

- For *squared error*, the *risk* for using the model f is:

$$\begin{aligned} R(f) &= E_{XY}[L(Y, f(X))] \\ &= E_{XY}[(Y, f(X))^2] \end{aligned}$$

where the expectation is w.r.t. the *test values* of X, Y.

  - Note under squared error loss, the risk is also known as the *mean squared error* (MSE)

- To simplify a bit, let's examine the risk of model f at a given input $X = x$. This removes the uncertainty in X, so we only have uncertainty coming from Y.

$$\begin{aligned} R_x(f) &= E[L(Y, f(x)) \mid X = x] \\ &= \mathrm{E}[(Y - f(x))^2 \mid X = x] \qquad \text{for squared error loss} \end{aligned}$$

where the expectation is taken with respect to $Y|X = x$

- The best prediction $f^*(x)$, given $X = x$, is the value that minimizes the risk

$$\begin{aligned} f^*(x) &= \arg\min_c R_x(c) \\ &= \arg\min_c \mathrm{E}[(Y - c)^2 \mid X = x] \end{aligned}$$

Your Turn #5

What is the optimal prediction at $X = x$, $f^*(x)$?

- **Conclusion:** If quality of prediction is measured by squared error, then the best predictor is the (conditional) expected value $f^*(x) = \mathrm{E}[Y|X = x]$.
  - And the minimum Risk/MSE is $R_x(f^*) = \mathrm{V}[Y|X = x]$
- Under *squared error loss* the Risk is

$$\begin{aligned} R_x(f) &= \mathrm{E}[L(Y, f(X)) \mid X = x] \\ &= \mathrm{E}[(Y - f(x))^2 \mid X = x] \\ &= \mathrm{V}[Y \mid X = x] + (E[Y \mid X = x] - f(x))^2 \\ &= \text{Irreducible Variance} + \text{squared error} \end{aligned}$$

6.2.1 kNN and Polynomial Regression

- The kNN model estimates the conditional expectation by using the data in a *local region* around x

$$\hat{f}_{\text{knn}}(x; k) = \mathrm{Ave}(\mathrm{y_i} \mid \mathrm{x_i} \in \mathrm{N_k(x)})$$

This assumes that the true $f(x)$ can be well approximated by a *locally constant* function

- Polynomial (linear) regression, on the other hand, assumes that the true $f(x)$ is well approximated by a *globally polynomial* function

$$\hat{f}_{\text{poly}}(x; p) = \beta_0 + \sum_{j=1}^{p} \beta_j x^j$$

6.2.2 Empirical Risk

- The actual Risk/EPE is based on the error from *test data* (out-of-sample), or data that was not used to estimate $\hat{f}$

$$\begin{aligned} R(f) &= E_{XY}[L(Y, f(X))] \\ &= E_{XY}[(Y, f(X))^2] \qquad \text{for squared error loss} \end{aligned}$$

  where X, Y are from $\Pr(X, Y)$ (i.e., test data)

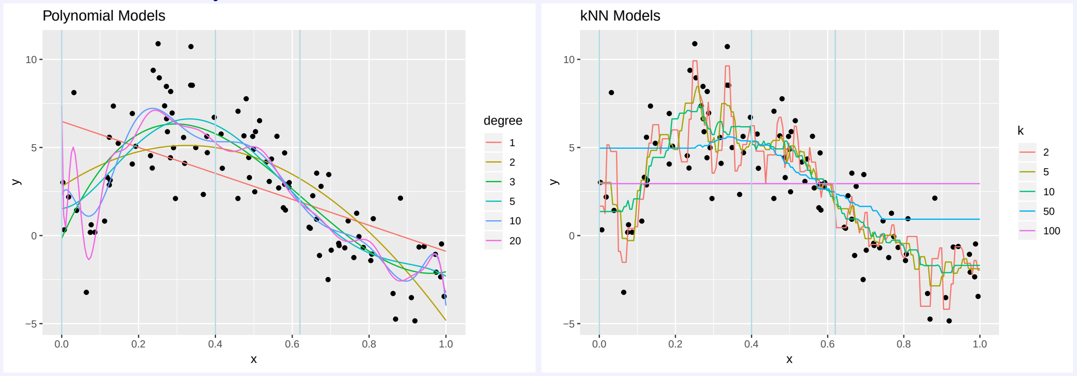
- But is it a bad idea to choose the best model according to *empirical risk* or *training error*?

$$\begin{aligned} R_n(f) &= \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) \\ &= \frac{1}{n}\sum_{i=1}^{n} (y_i - f(x_i))^2 \qquad \text{for squared error loss} \end{aligned}$$

6.3 Choose the best *predictive* model

Your Turn #6

Which model will you choose?



Polynomial

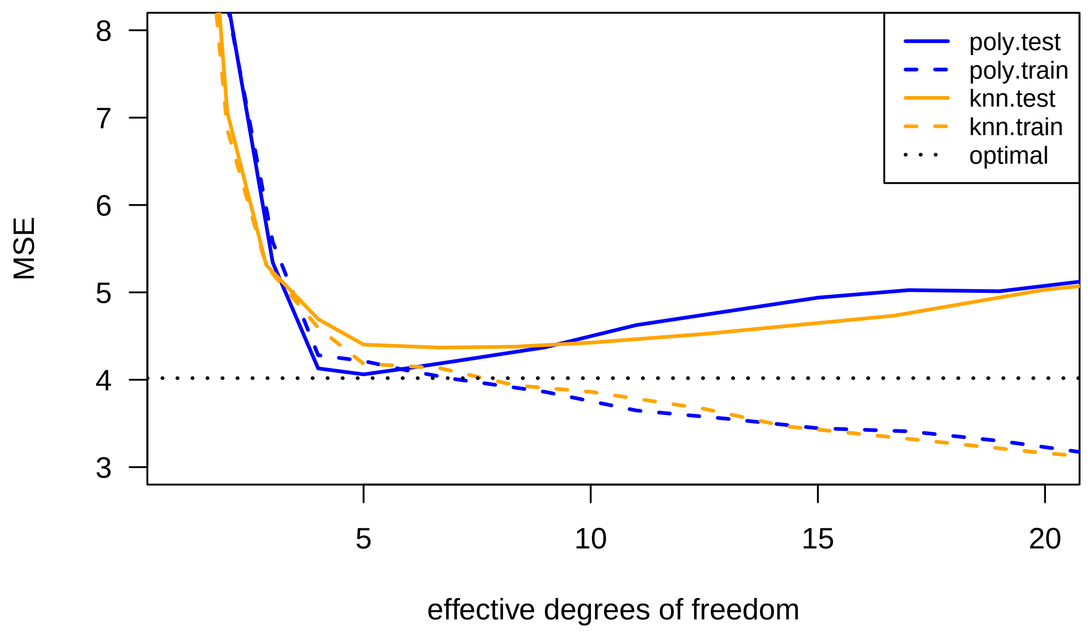
degree	RSS	npars
1	829.1	2
2	557.8	3
3	428.1	4
5	410.2	6
10	364.9	11
20	315.6	21

kNN

k	RSS	edf
100	1240.2	1
50	686.5	2
10	386.1	10
5	316.0	20
2	183.9	50

7 Evaluate Simulated Test Data (or which model is best)

I simulated 50,000 test observations, made predictions from each model, and recorded the estimated MSE/Risk.



Polynomial

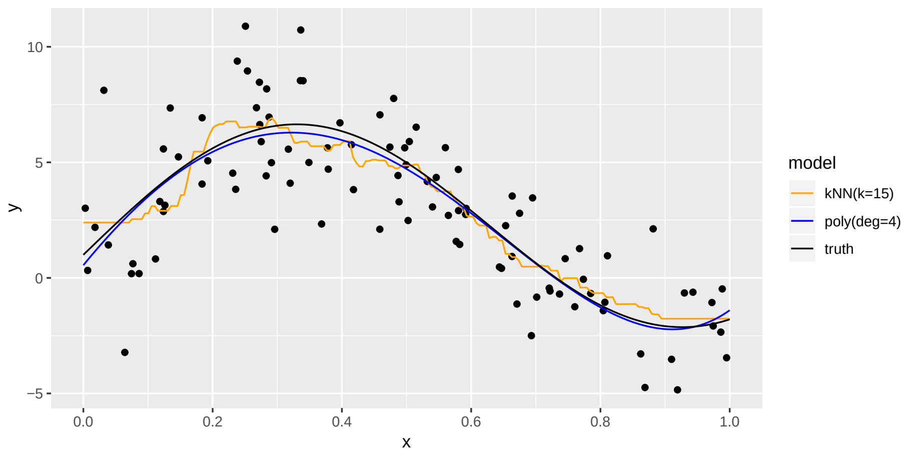
degree	edf	mse.train	mse.test
1	2	8.29	8.36
2	3	5.58	5.34
3	4	4.28	4.13
4	5	4.21	4.06
5	6	4.10	4.13
6	7	4.01	4.21
8	9	3.86	4.37
10	11	3.65	4.63
12	13	3.55	4.78
14	15	3.44	4.94
16	17	3.41	5.03
18	19	3.30	5.01
20	21	3.16	5.14

kNN

k	edf	mse.train	mse.test
100	1.00	12.40	13.71
50	2.00	6.87	7.06
35	2.86	5.30	5.31
25	4.00	4.59	4.69
20	5.00	4.18	4.40
15	6.67	4.13	4.37
12	8.33	3.94	4.38
10	10.00	3.86	4.42
8	12.50	3.67	4.52
7	14.29	3.47	4.61
6	16.67	3.34	4.73
5	20.00	3.16	5.03
2	50.00	1.84	6.57

Observations:

- as the complexity increases, both classes of models *overfit*.
  - overfit means model too complex
  - underfit means model is not complex enough
  - see discrepancy between training and test performance
- The polynomial with degree = 4 has the best test performance with an approximate $\text{MSE} = 4.06$.
- The optimal $\text{MSE} = 4$.
  - I only know this because I know how the data was generated



8 Ensemble Models

In class, you gave your votes for which model you thought was best:

model	number of votes
knn(k=10)	7
knn(k=5)	5
poly(deg=5)	4
poly(deg=3)	4
poly(deg=4)	2
poly(deg=10)	2
knn(k=20)	2
knn(k=7)	1
knn(k=8)	1

- Can we use the collective *wisdom of the crowds* to help make a better prediction?
- An *ensemble model* is one that combines several models together.

The approach is to create a new ensemble model that is a weighted sum of the individual models

$$f_w(x) = \sum_{j=1}^{p} w_j f_j(x)$$

In our specific example, we had

$$f_w(x) = \frac{7}{28} f_{\text{knn}}(x, k = 10) + \frac{5}{28} f_{\text{knn}}(x, k = 5) + \frac{4}{28} f_{\text{poly}}(x, deg = 5) + \ldots + \frac{1}{28} f_{\text{knn}}(x, k = 8)$$

and the corresponding test performance is given by

$$R = \frac{1}{M} \sum_{j=1}^{M} (y_j - \hat{f}_w(x_j))^2$$

where J is the number of test observations.

This gives an MSE of 4.24, which is better than most individual models

model	n	w	mse
poly(deg=4)	2	0.07	4.06
poly(deg=3)	4	0.14	4.13

model	n	w	mse
poly(deg=5)	4	0.14	4.13
ensemble	NA	NA	4.24
knn(k=20)	2	0.07	4.40
knn(k=10)	7	0.25	4.42
knn(k=8)	1	0.04	4.52
knn(k=7)	1	0.04	4.61
poly(deg=10)	2	0.07	4.63
knn(k=5)	5	0.18	5.03

9 Bias-Variance Trade-off

This section explores the bias-variance trade-off for the examples we covered last class. This involves examining the theoretical properties of an estimator.

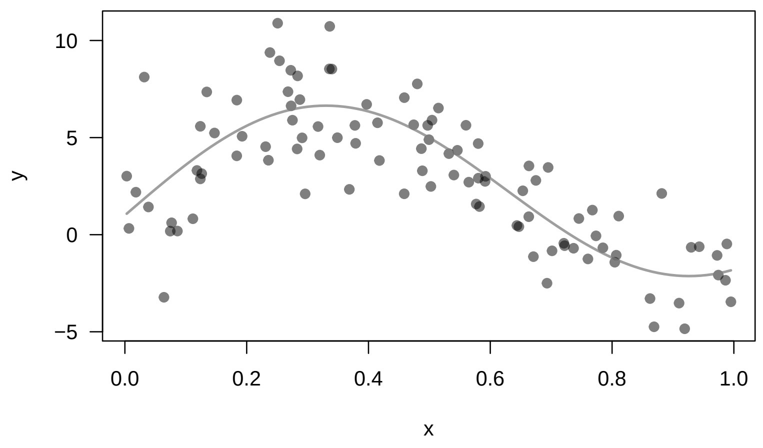
9.1 Data Generating Functions

Here we set the data generation functions. $X \sim U[0,1]$ and $f(x) = 1 + 2x + 5\sin(5x)$ and $y(x) = f(x) + \epsilon$, where $\epsilon \overset{\text{iid}}{\sim} N(0,2)$.

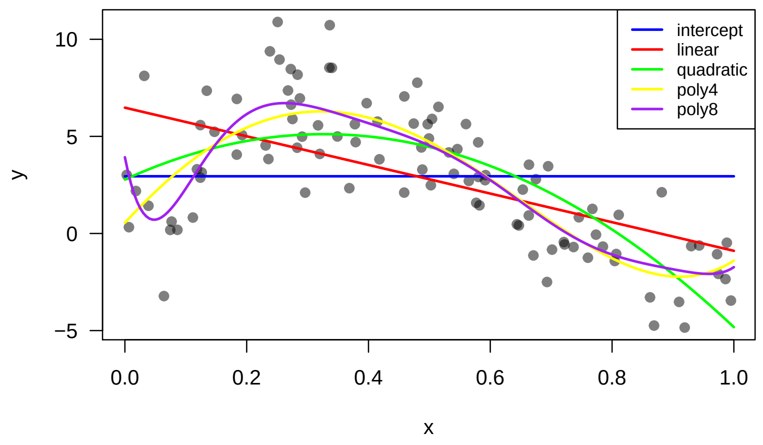
```
sim_x <- function(n) runif(n)             # generate n obs from U[0,1]
f <- function(x) 1 + 2*x +5*sin(5*x)      # true mean function
sim_y <- function(x){                     # generate Y|X from N{f(x),sd}
  n = length(x)
  f(x) + rnorm(n, sd=2)
}
```

9.2 One Realization

Last class, we explored one realization from this system.

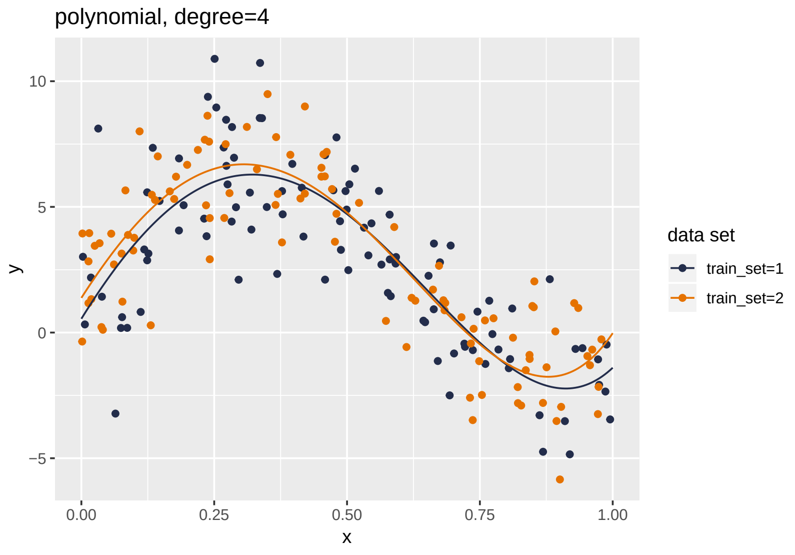


And then fit several polynomial regression models. Recall by polynomial regression I mean using a predictor function $\hat{y}(x) = f(x,d) = \sum_{j=0}^{d} x^j \beta_j$, where $d\{0,1,\ldots\}$ is the degree.



9.3 A second realization

Suppose we drew another training set (using same distributions and sample size n):



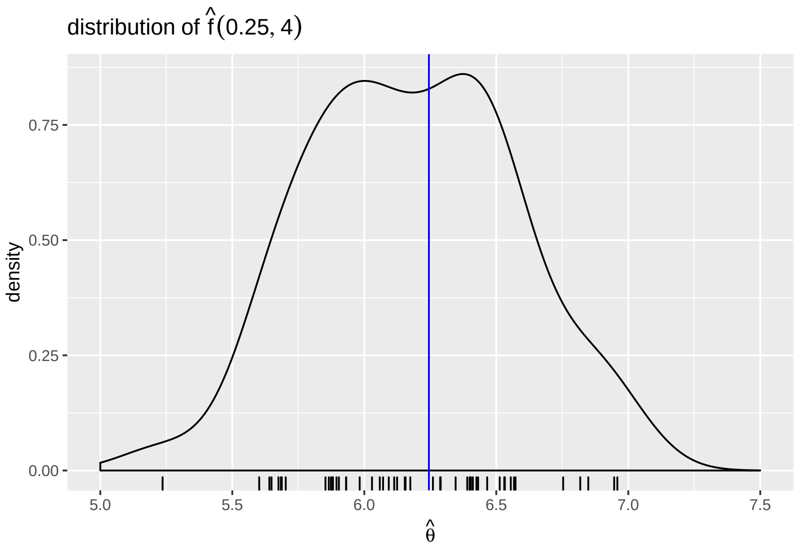
- We get another fitted curve using the new training data.
- While the two curves are visually similar, they are not identical.
- If we took more training samples, we would more fitted curves
- What we want to study in this section is the likelihood that we will happen to get a *good* fit given a single training data set.

9.4 Bias, Variance, and Mean Squared Error (MSE)

- The statistical properties of an estimator can help us understand its potential performance
- Let $D = [(X_1, Y_1), (X_2, Y_2), \ldots, (X_n, Y_n)]$ be *training data*
- Let $\hat{\theta} = \hat{\theta}(D)$ be the estimated parameter *calculated from the training data D*
  - E.g. $\theta = f(x)$, $\hat{\theta} = \hat{f}(x)$
  - θ is a random variable; it has a distribution.

9.4.1 Distribution of θ

- Consider the distribution of $\theta = \hat{f}_{\text{poly}}(0.25, d = 4)$.
- I generated 50 different training data sets (each with $n = 100$), fit a polynomial (deg=4) model to each data set, and recorded the estimate at $x = 0.25$.



9.4.2 Some properties of an estimator

- **Bias** of an estimator is defined as $\mathrm{E}_D[\hat{\theta}] - \theta$
- **Variance** of an estimator is defined as $\mathrm{V}_D[\hat{\theta}] = \mathrm{E}_D[\hat{\theta}^2] - \mathrm{E}_D[\hat{\theta}]^2$
- **MSE** of an estimator is defined as $\mathrm{MSE}(\hat{\theta}) = E_D[(\hat{\theta} - \theta)^2]$

$$\begin{aligned}\mathrm{MSE}(\hat{\theta}) &= E_D[(\hat{\theta} - \theta)^2] \\ &= V_D[\hat{\theta} - \theta] + E_D[\hat{\theta} - \theta]^2 \\ &= V_D[\hat{\theta}] + E_D[\hat{\theta} - \theta]^2\end{aligned}$$

- Estimators are often evaluated based on MSE, being unbiased, and/or having minimum variance (out of all unbiased estimators)
- These properties are based on the *distribution of an estimate*.
  - Once we observe the training data, the resulting estimate may be great or horrible.
  - However these theoretical properties provide insight into what we can expect and how much confidence we can have in the estimates.

9.5 Estimating the Bias, Variance, and Mean Squared Error (MSE)

- We have examined the Risk (e.g., MSE) *conditioning on the training data*
- Now we will relax this and bring in the uncertainty in the training data D

Under a squared error loss function $L(Y, f(X)) = (Y - f(X))^2$, the Risk, or expected loss, at a particular $X = x$ is

$$\begin{aligned}\mathrm{MSE}_x(f) &= \mathrm{E}_{DY|X}[(Y - \hat{f}_D(x))^2 \mid X = x] \\ &= \mathrm{V}[Y \mid X = x] + \mathrm{V}[\hat{f}_D(x) \mid X = x] + \Big(\mathrm{E}[\hat{f}_D(x) \mid X = x] - f(x)\Big)^2 \\ &= \text{irreducible error} + \text{model variance} + \text{model squared bias}\end{aligned}$$

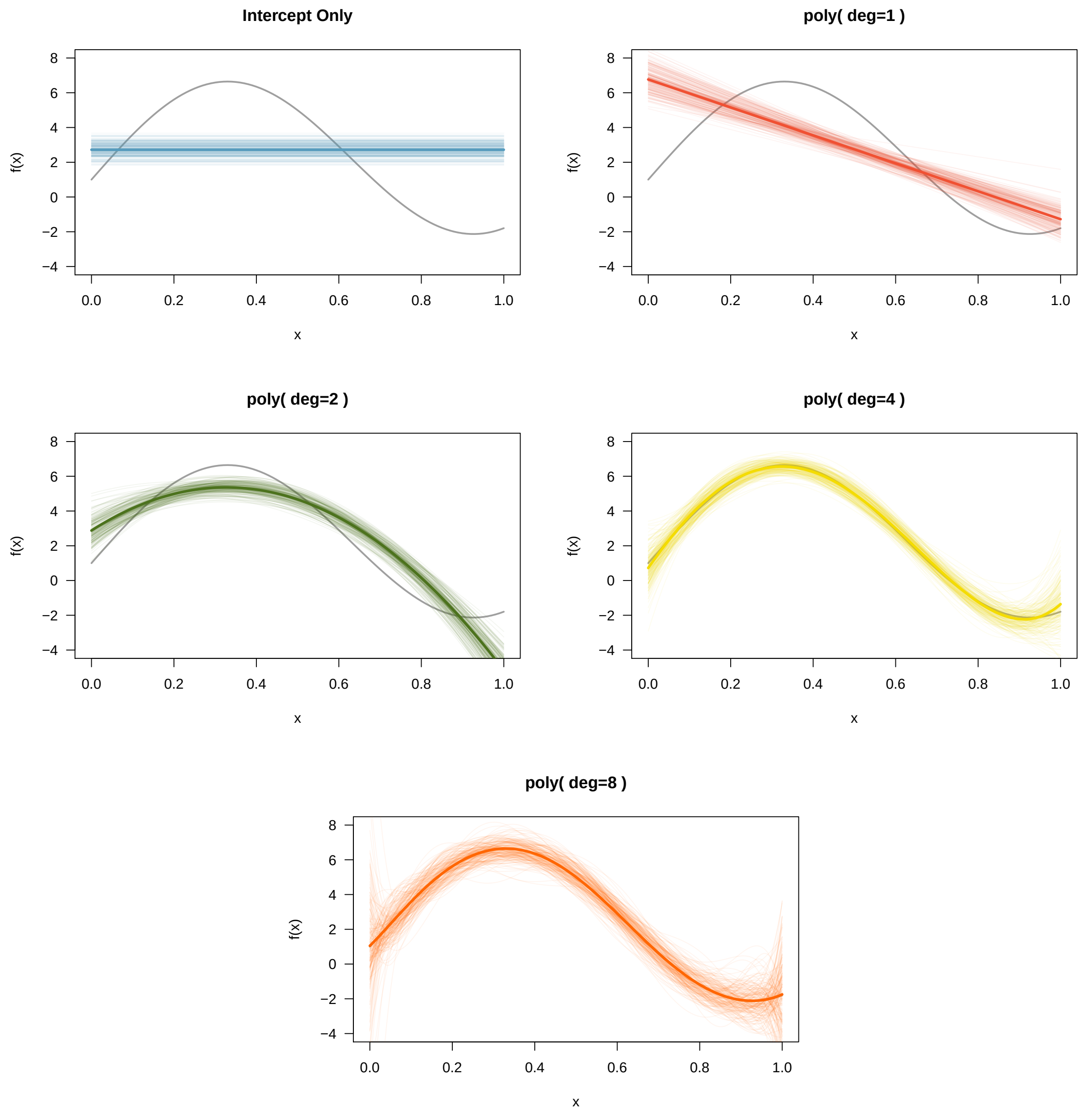
where D is the training data, f is the true model, and $\hat{f}_D(x)$ is the prediction at $X = x$ estimated from the training data D.

- We have set $\mathrm{V}[Y|X] = 2^2$ in the set-up.

- We can estimate the model variance and bias with simulation
  - generate new data $D_m = \{(Y_i, X_i)\}$ for simulations $m = 1, 2, \ldots, M$ (use the same sample size n)
  - fit the models with data D_m getting $\hat{f}_{D_m}(\cdot)$
  - now we can estimate $\mathrm{E}[\hat{f}_D(x)] \approx \bar{f}(x) = \frac{1}{M}\sum_{m=1}^{M} \hat{f}_{D_m}(x)$ and $\mathrm{V}[\hat{f}_D(x)] \approx s_f^2(x) = \frac{1}{M-1}\sum_{m=1}^{M}(\hat{f}_{D_m}(x) - \bar{f}(x))^2$

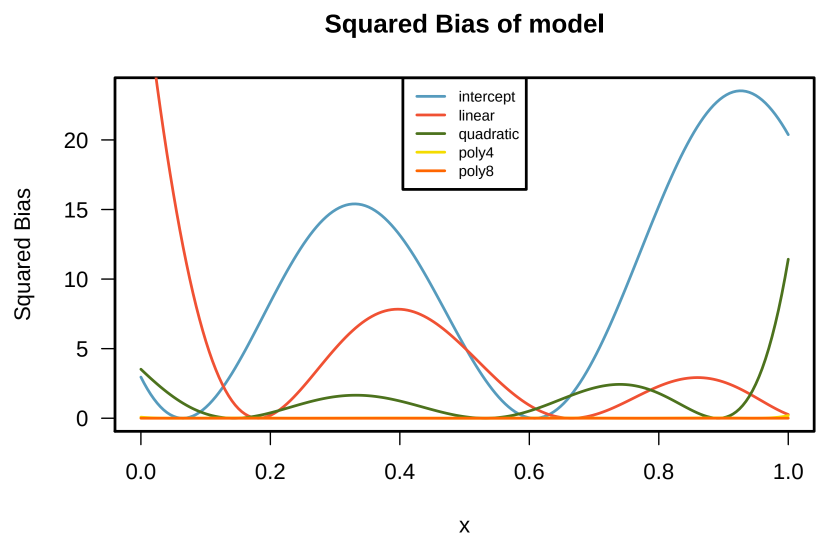
9.5.1 Simulation

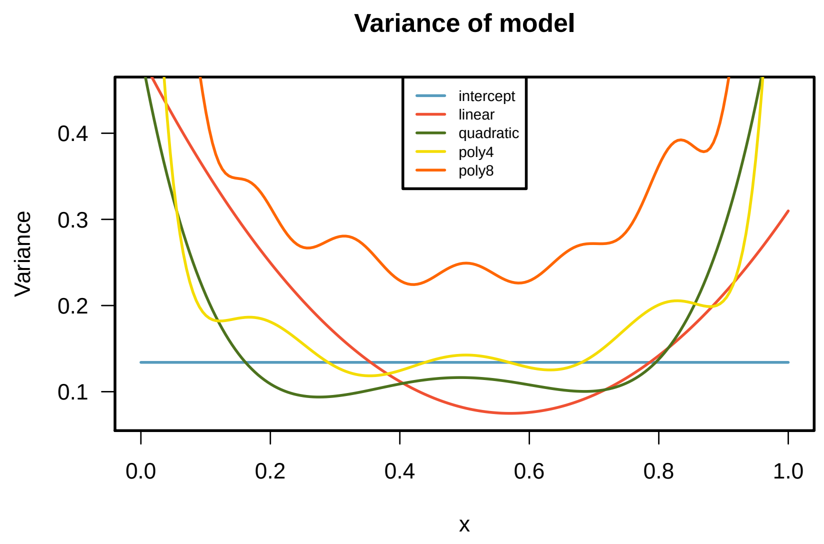
I ran 2000 simulations to generate $\{\hat{f}_m(x, deg = d) : d \in \{0, 1, 2, 4, 8\}, m \in \{1, 2, \ldots, 2000\}\}$.

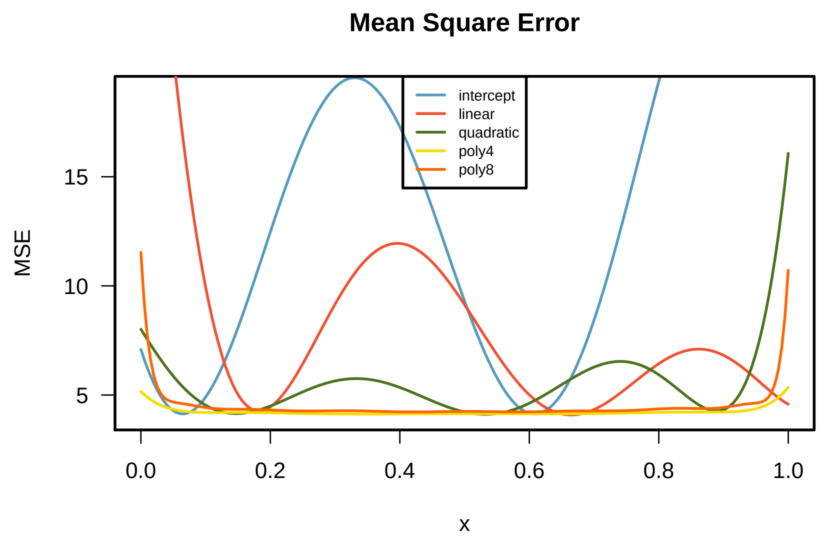


9.5.2 Observations

- This shows the bias and variance of each model.
- You can see that as the complexity (e.g. degree) of the model increases, the bias decreases but the variance (especially at the edges) increases.
- The bias is the difference between the true regression function (dark gray line) and the model mean (dark colored line).
- The variation is seen in the width of the transparent curves, one for each simulation.







- Notice that model variance and model bias vary over x.

9.5.3 Integrated MSE

The above analysis examines the $\text{MSE}_x(f)$ over a set of x's. However, in a real setting, the overall test error will based on *all* of the actual test X values. So we are usually more interested in the *integrated* MSE or

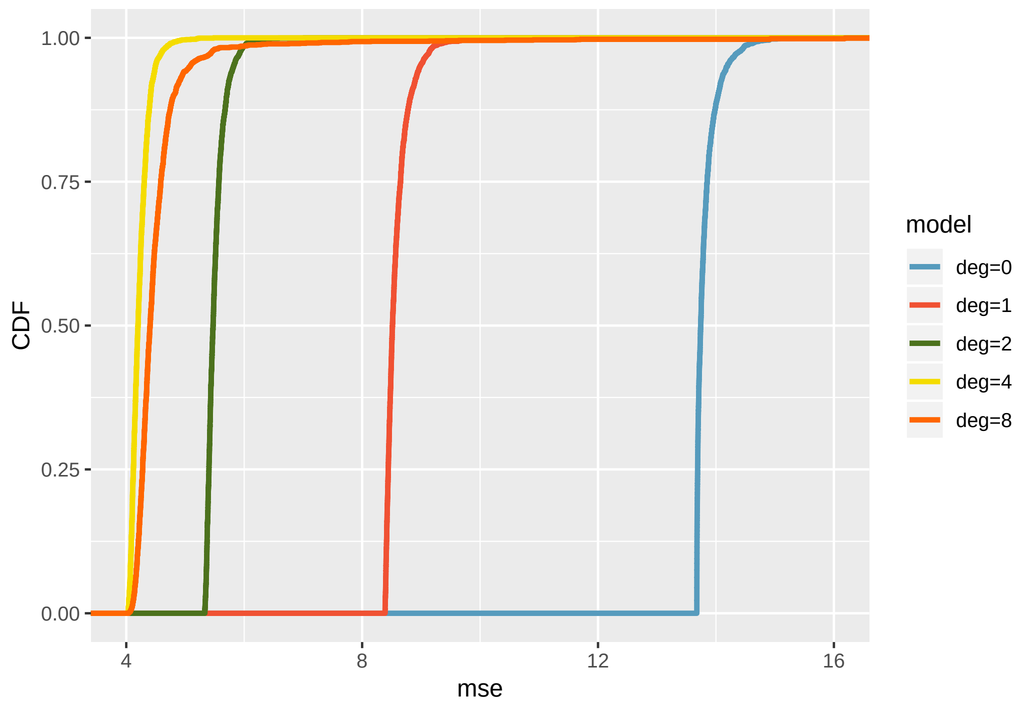
$$\begin{aligned}\text{MSE}(f) &= \text{E}_{DYX}[(Y - \hat{f}_D(X))^2] \\ &= E_X[\text{MSE}_X(f)] \\ &= \int \text{MSE}_x(f)\,\text{Pr}(dx)\end{aligned}$$

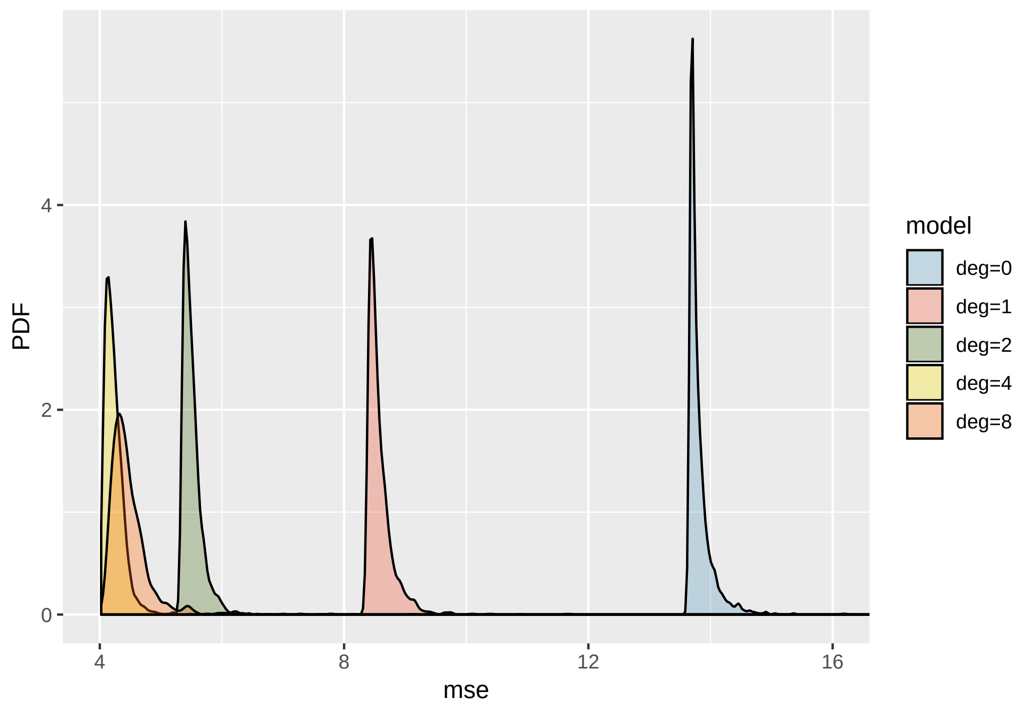
	deg=0	deg=1	deg=2	deg=4	deg=8
mse	13.81	8.58	5.51	4.23	4.53
bias.sq	9.68	4.39	1.33	0.01	0.00
var	0.13	0.19	0.18	0.22	0.53

9.6 What does it all mean

The main point here is we desire to find a predictive model that has just the right *complexity*. This will depend on both the true complexity of the data as well as the sample size. A model that is too complex will have low bias, but potentially high variance. A model that is not complex enough will have high bias, but lower variance.

In a real setting, we will only observe one training data (a single curve) and will have to decide the optimal complexity. So here will we examine the entire distribution of MSE values (one for every simulation).





While its possible that we could just happen to get a particular training data realization that favors a model other than the globally optimal model, this is unlikely for the bad models. However, it is not uncommon for "close" models.

degree	n
2	1
4	1941
8	58