Machine Learning II. Overfitting. Bias-variance tradeoff.

Juan Pablo Lewinger
07/03/2024

Yesterday

- **Linear regression - least squares**
- **A linear model fit in R:**
  - `lm(Brain.weight ~ Head.size + Sex + Age, data = brain_train)`
- **Training and evaluating performance on same data overestimates performance**
  - Randomly split into training and test sets (e.g. 70% training, 30% test):
    - Train model on training set
    - *Estimate* prediction performance on test set
  - Performance metrics: MSE, RMSE (absolute error) and R^2 (relative error)
- **For predicting on new data, use best possible model: i.e. model trained on all available data**
- **For prediction with LR, linearity and equal variance assumptions are not required (but if assumptions hold model may predict better)**

Interactions and higher order terms

A model with an interaction using the interaction `:` operator:

```
brain_lm1 = lm(Brain.weight ~  Sex + Head.size + Head.size:Sex , data = brain_train)
summary(brain_lm1)$coefficients
```

```
##                        Estimate Std. Error t value Pr(>|t|)
## (Intercept)            409.4131    97.4476    4.20 4.38e-05
## SexFemale             -163.4863   139.1747   -1.17 2.42e-01
## Head.size                0.2428     0.0256    9.49 3.10e-17
## SexFemale:Head.size      0.0435     0.0387    1.12 2.63e-01
```

Interactions and higher order terms

Same model can be fitted using the $*$ operator:

```
brain_lm1 = lm(Brain.weight ~  Sex*Head.size, data = brain_train)
summary(brain_lm1)$coefficients
```

```
##                      Estimate Std. Error t value Pr(>|t|)
## (Intercept)          409.4131    97.4476    4.20 4.38e-05
## SexFemale           -163.4863   139.1747   -1.17 2.42e-01
## Head.size              0.2428     0.0256    9.49 3.10e-17
## SexFemale:Head.size    0.0435     0.0387    1.12 2.63e-01
```

Interactions and higher order terms

We can also add non-linear terms:

```
brain_lm2 = lm(Brain.weight ~ Head.size + I(Head.size^2), data = brain_train)
summary(brain_lm2)$coefficients
```

```
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -2.53e+02   4.21e+02  -0.602   0.5481
## Head.size       5.80e-01   2.31e-01   2.510   0.0131
## I(Head.size^2) -4.28e-05   3.16e-05  -1.353   0.1779
```

Model selection

- So far we've dealt with fitting a single model
  - Train on training data
  - Assess prediction performance on test data
- In practice we want to choose from many possible models
  - Include/exclude particular features?
  - Include interactions or higher order terms?
  - Competing algorithms (e.g. linear regression vs. KNN)
- **How do we choose?**

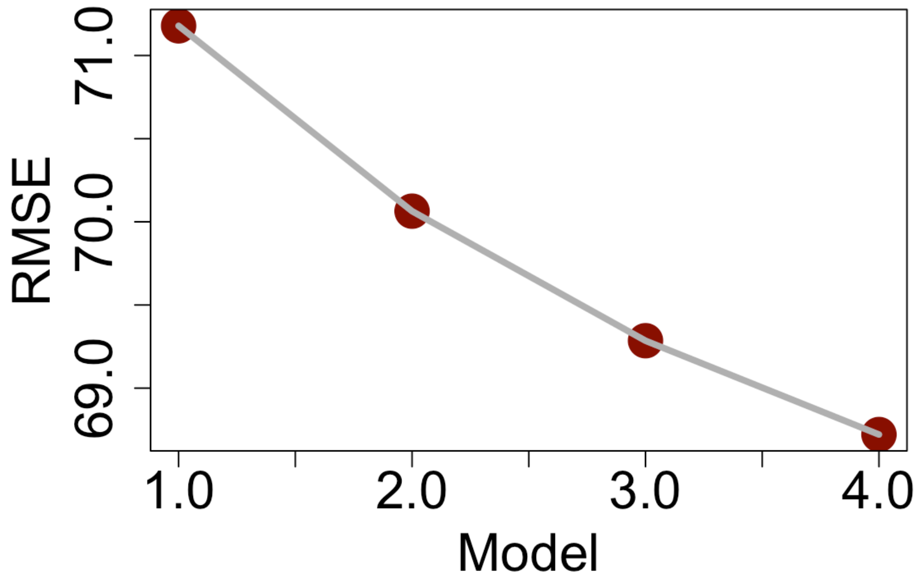
Model selection – example

Let's fit models of increasing complexity on the brain data:

```
brain_fit1 = lm(Brain.weight ~ Head.size, data=brain_train)

brain_fit2 = lm(Brain.weight ~ Head.size + Sex + Age, data=brain_train)

brain_fit3 = lm(Brain.weight ~ (Head.size + Sex + Age)^2, data=brain_train)

brain_fit4 = lm(Brain.weight ~ (Head.size + I(Head.size^2) + Sex + Age)^2, data=brain_train)
```

Model selection – example

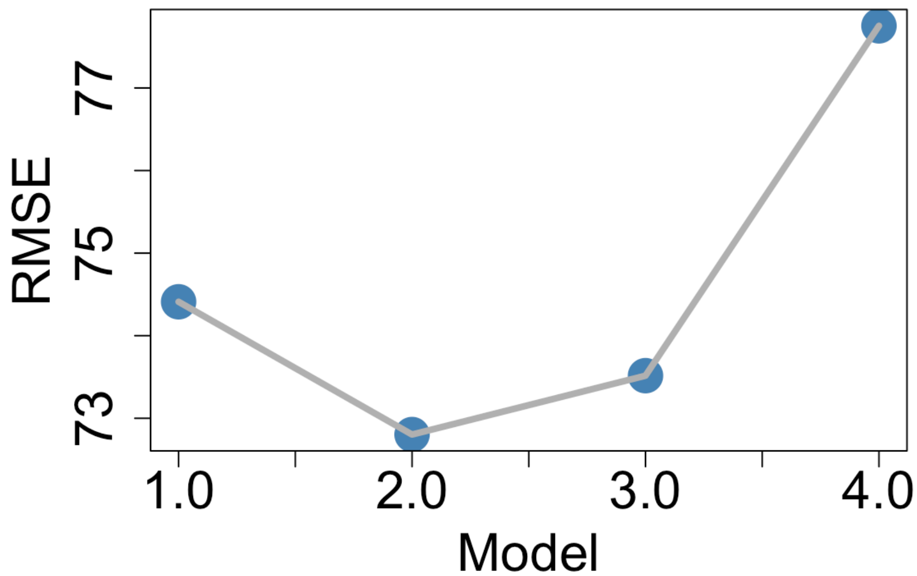
Let's look at their **training** RSMEs:



Which model should we choose?

Model selection – example

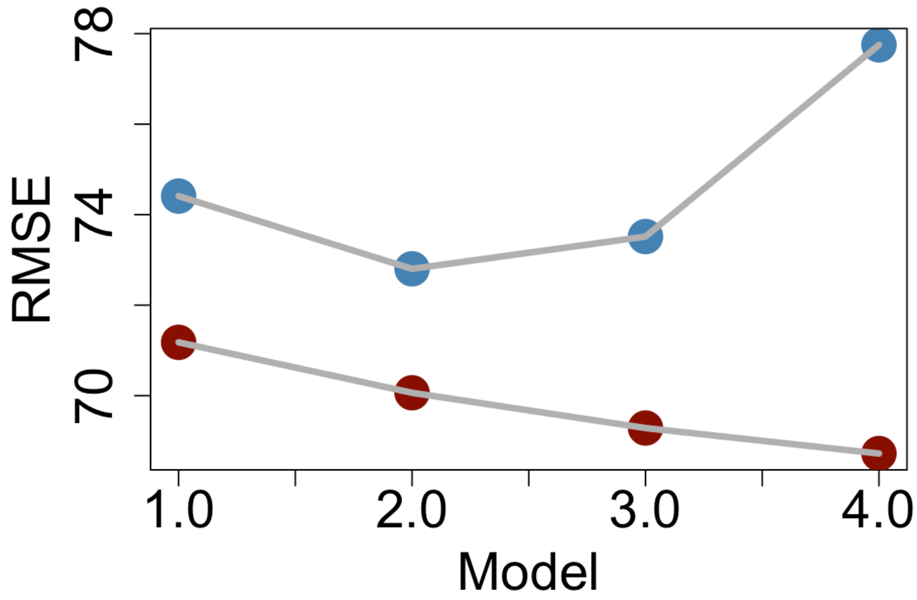
Let's look at their **test** RSMEs:



Which model should we choose?

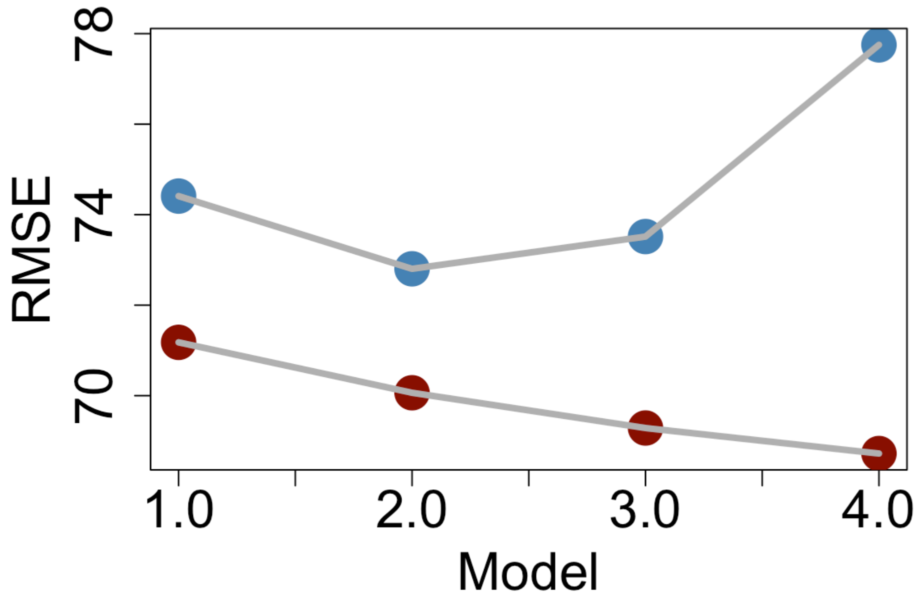
Model selection – example

Let's look at their **train and validation** RSMEs:



Which model should we choose?

Model selection – example



Pick Model 2, `Brain.weight ~ Head.size + Sex + Age`, which has the best test/validation error among the 4 models we considered

Model selection

- Picking model with smallest training MSE or highest training R^2 is a bad idea.
- Right thing to do is split data into training and testing as before and choose model with smallest **test** MSE or highest **test** R^2.
- This is a fair way to compare prediction performance across different models.
- BUT, how do we get and estimate of prediction performance for the best model?
- Already ‘used up’ test set for choosing best model

Model selection – Validation set

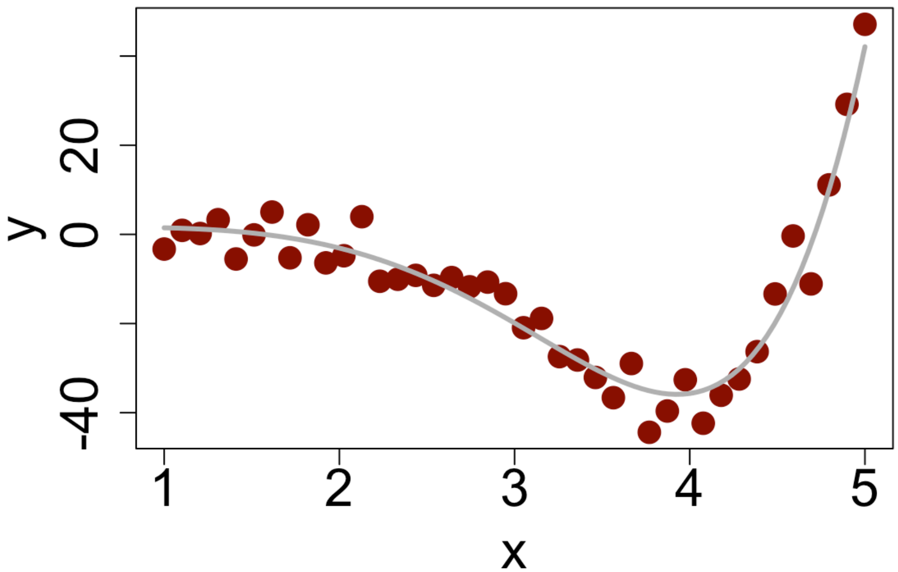
- Choosing smallest **test** MSE or highest **test** R^2 will give overly optimistic assessment of prediction error
- Reason is analogous to training error overestimating prediction performance
  - E.g. we fit linear regression by minimizing residual sum of squares RSS
  - In training data model fits better than in test data
- Minimizing test error over several models makes the best model ‘look too good’
- Need a separate, never-used test set for final estimate of prediction performance!

Model selection – Validation set

- For model selection we split the data in 3: **training, validation, and test sets**:
    - **Training set** we use to fit all models considered (e.g. several linear regression models)
    - **Validation set** we use to perform model selection – pick model that gives best prediction error on validation set
    - **Test set** we **only use once** to get an unbiased final estimate of the prediction error for the selected model
- Only works if we have relatively large n
- For smaller n we can repeat the split into training and validation sets multiple time (cross-validation)

Model selection – simulation

- Consider some simulated training data: $n = 40$
- Generating model/ true relationship is non-linear: $y = e^x \cos(x) + \epsilon$
- $\epsilon \sim N(0, \sigma^2)$, $\sigma = 5$



Model selection – simulation

We'll fit polynomial regression models of increasing degree/complexity:

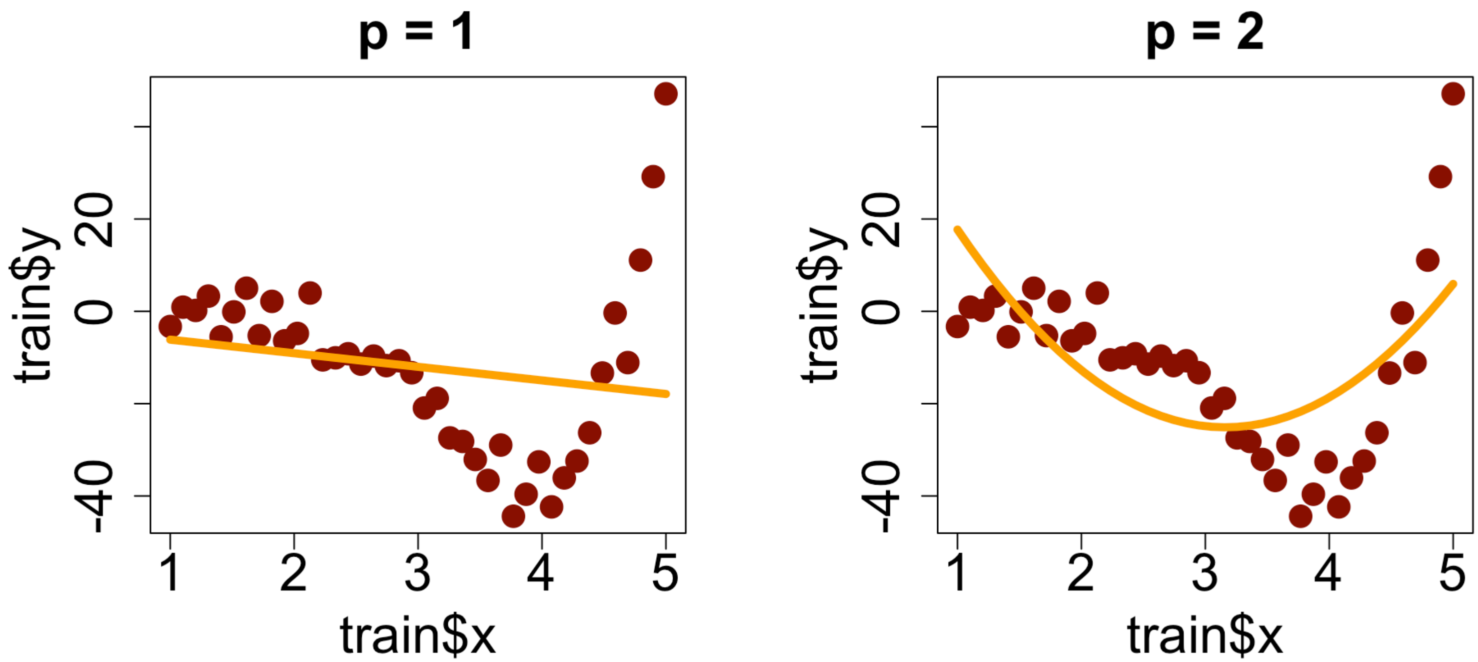
$$
\begin{aligned}
&Y = \beta_0 + \beta_1 X + \epsilon \\
&Y = \beta_0 + \beta_1 X + \beta_1 X^2 + \epsilon \\
&\vdots \\
&Y = \beta_0 + \beta_1 X + \beta_1 X^2 + \ldots + \beta_{25} X^{25} + \epsilon
\end{aligned}
$$

Model selection – simulation

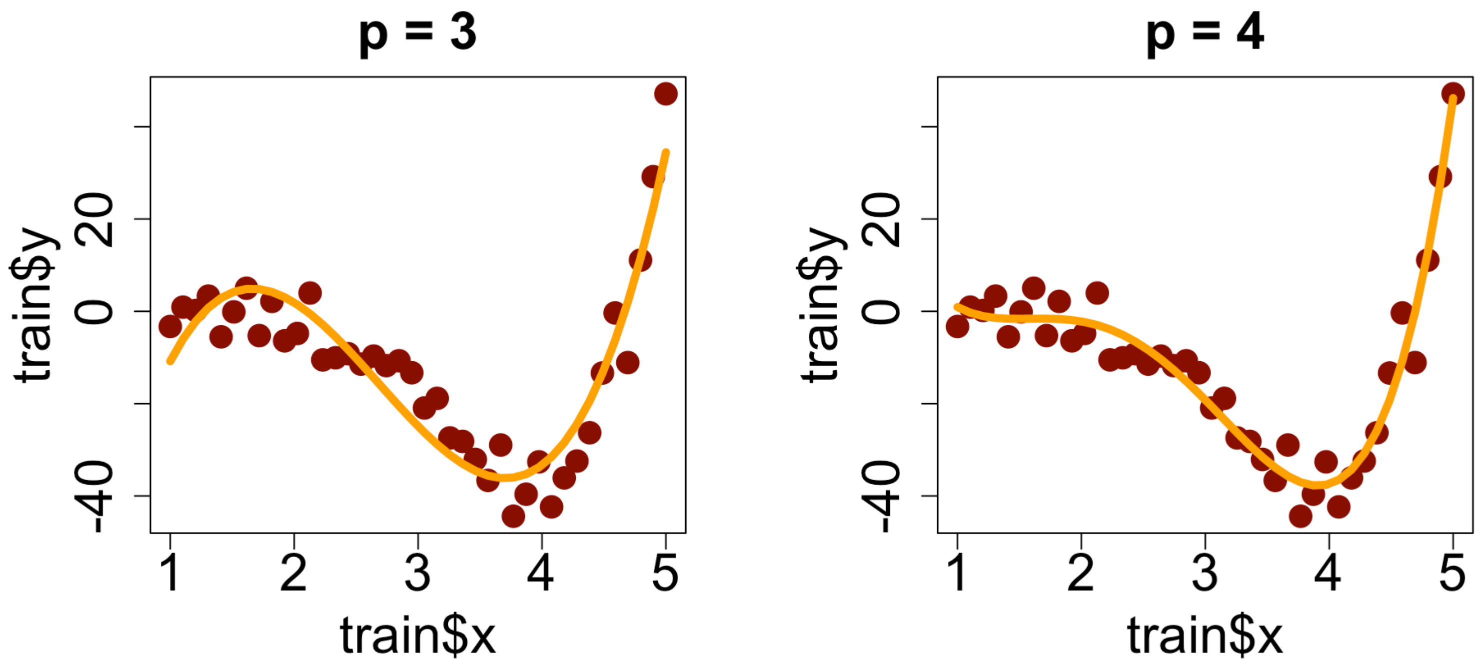
```
fit1 = lm(y ~ x, data = train)
fit2 = lm(y ~ x + I(x^2), data = train)
fit3 = lm(y ~ x + I(x^2) + I(x^3), data = train)
fit4 = lm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = train)
fit5 = lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5), data = train)
fit6 = lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6), data = train)
fit7 = lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + I(x^6) + I(x^7), data = train)
fit8 = lm(y ~ poly(x, 8), data = train)
fit9 = lm(y ~ poly(x, 9), data = train)
fit15 = lm(y ~ poly(x, 15), data = train)
fit20 = lm(y ~ poly(x, 20), data = train)
fit25 = lm(y ~ poly(x, 25), data = train)
```

Using poly() is more compact and numerically better!

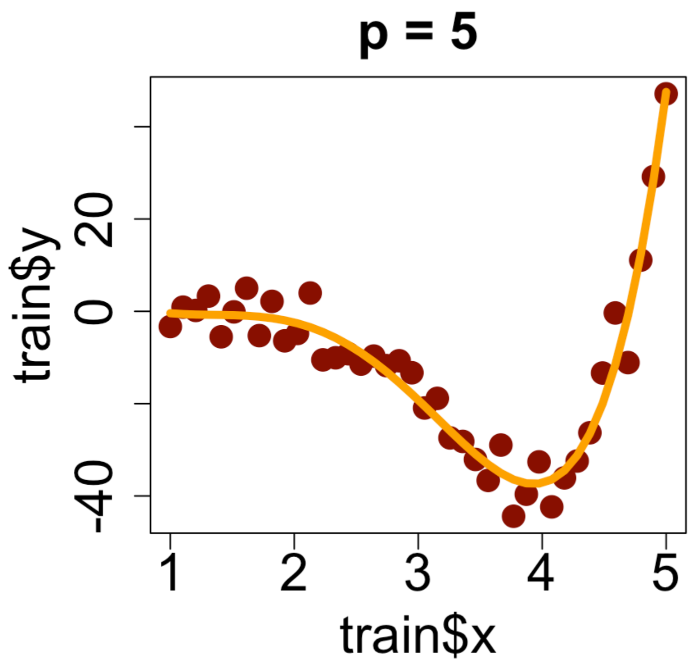
Model selection – simulation

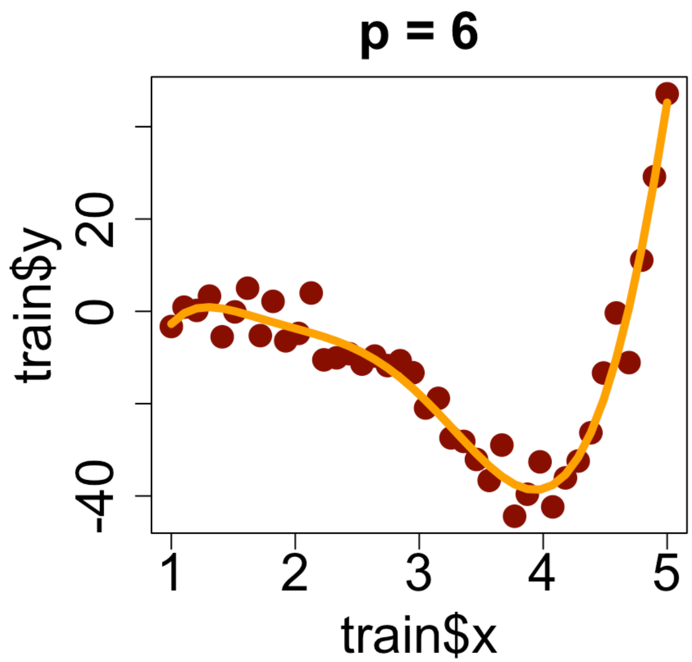


Model selection – simulation

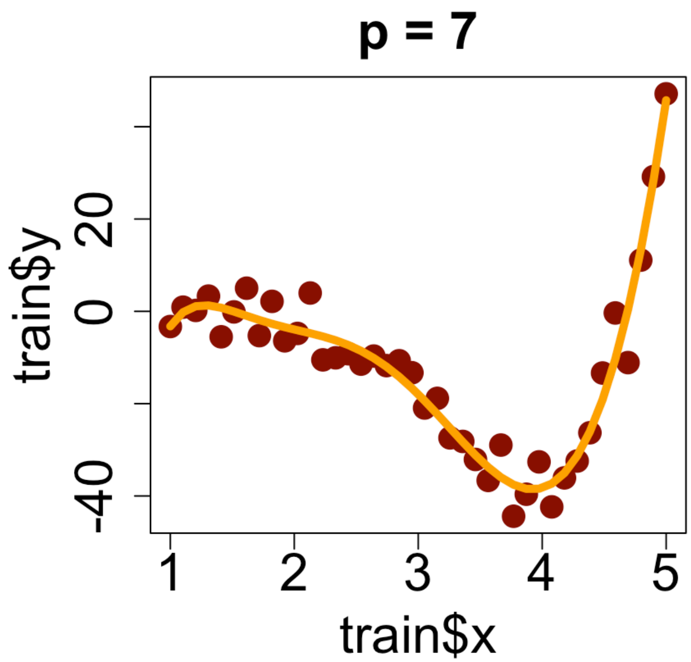


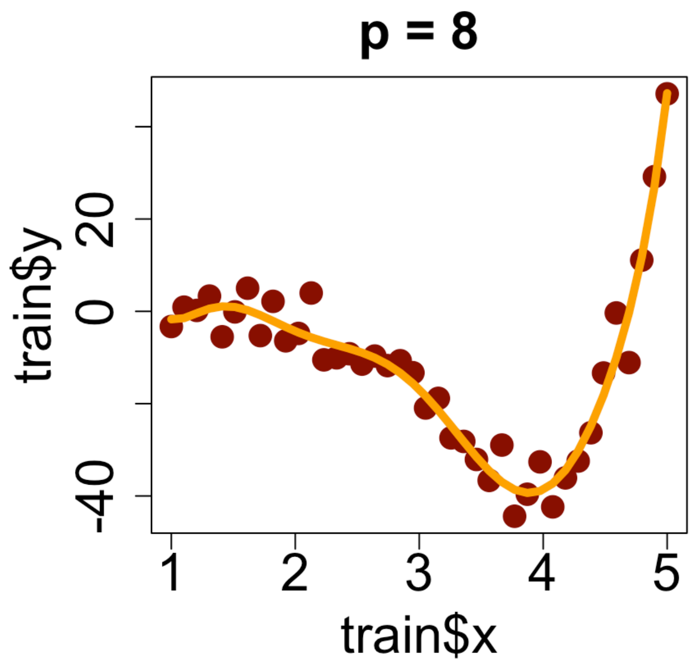
Model selection – simulation



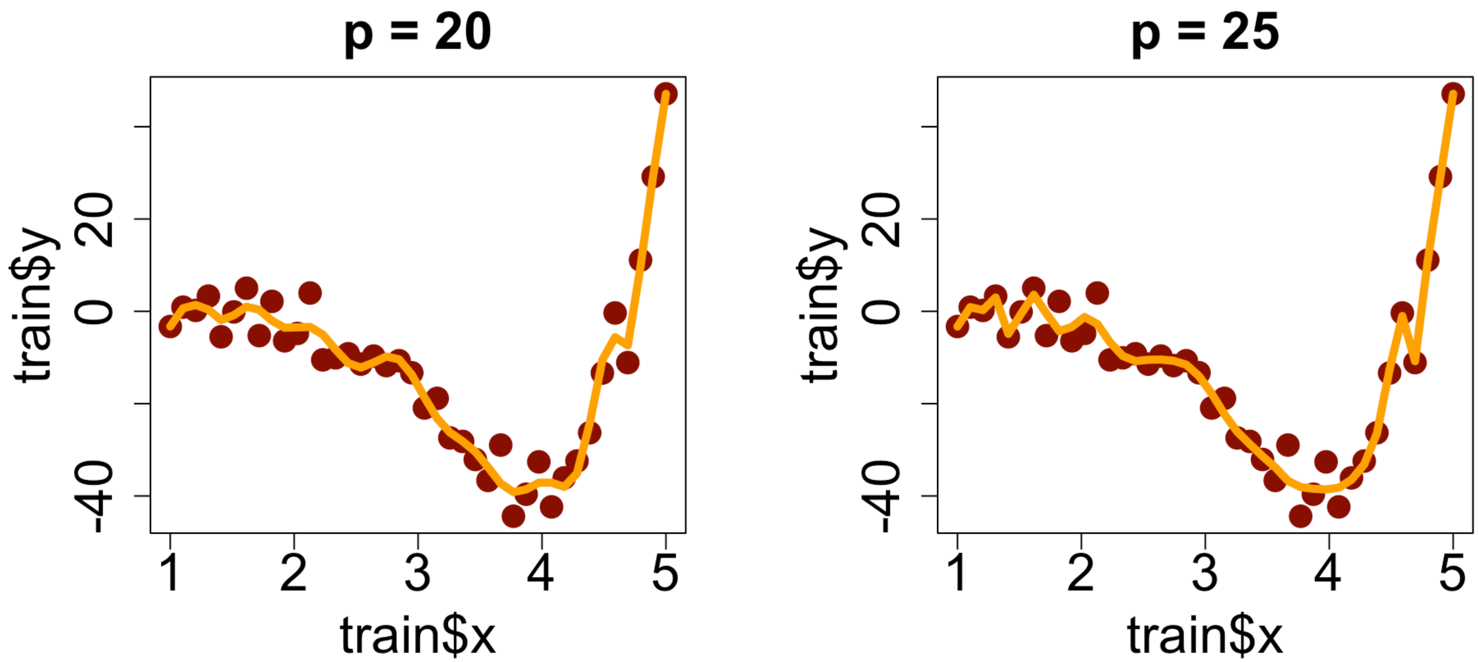


Model selection – simulation





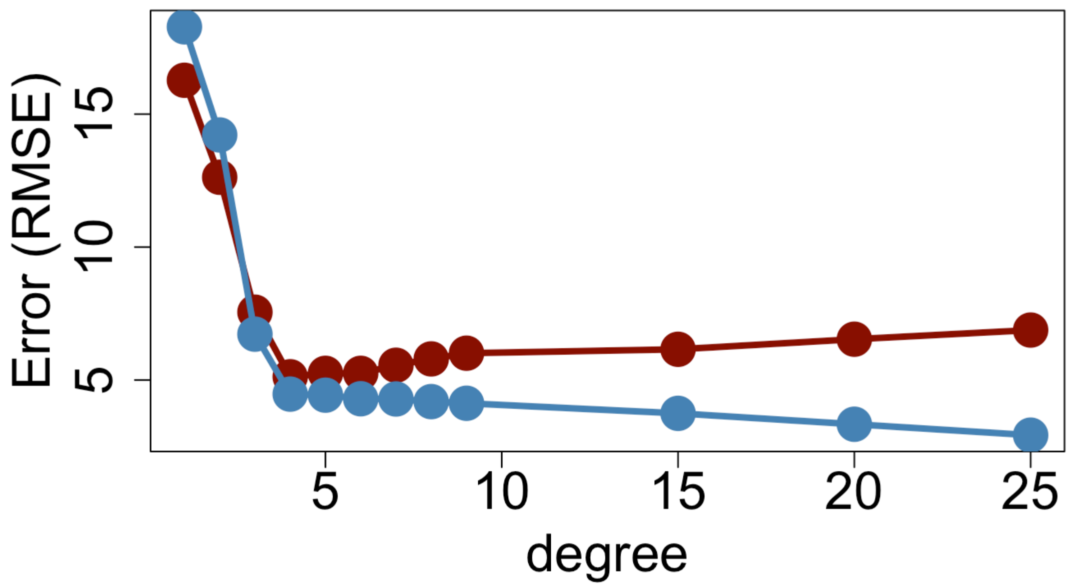
Model selection – simulation



Model selection – simulation

Training and validation RMSEs:

```
##             p=1  p=2  p=3  p=4  p=5  p=6  p=7  p=8  p=9 p=15 p=20 p=25
## Training   18.3 14.2 6.73 4.47 4.43 4.30 4.29 4.19 4.13 3.75 3.34 2.93
## Validation 16.3 12.6 7.55 5.12 5.25 5.25 5.56 5.79 6.01 6.16 6.54 6.88
```



Model selection – simulation

Training and validation RMSEs:

```
##             p=1  p=2  p=3  p=4  p=5  p=6  p=7  p=8  p=9 p=15 p=20 p=25
## Training   18.3 14.2 6.73 4.47 4.43 4.30 4.29 4.19 4.13 3.75 3.34 2.93
## Validation 16.3 12.6 7.55 5.12 5.25 5.25 5.56 5.79 6.01 6.16 6.54 6.88
```

- Model with $p = 4$ is 'just right': it **minimizes the validation error**
- (But the RMSE is estimated with error in the validation set due to finite sample size)
- (Could also perhaps choose $p = 5$ or $p = 6$ since RMSE is quite close to that of model with $p = 4$)

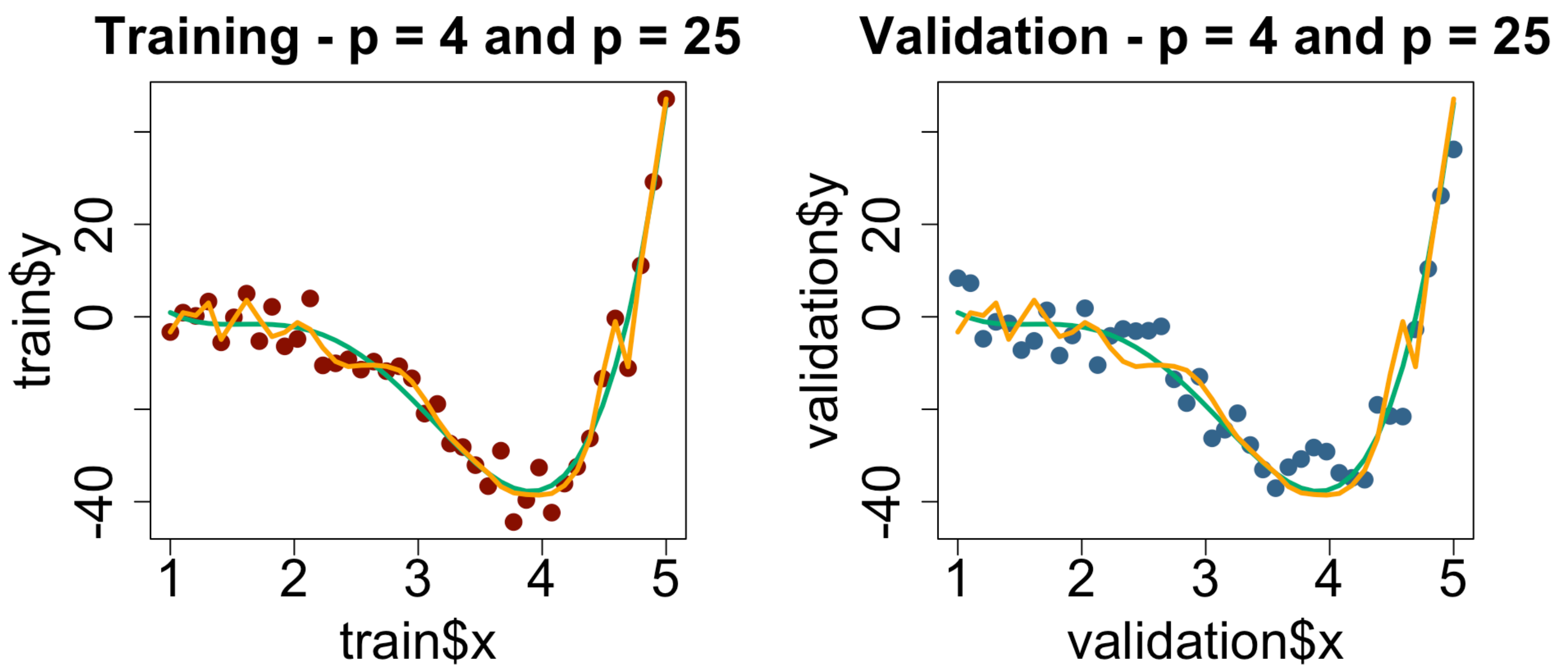
Model selection - summary

- We typically want try several models because we don't know beforehand which one will predict best
- Cannot compare models based on the training error because more complex models will always have smaller training error
- Should use a separate validation set to compare models
- But validation error does not give honest assessment of prediction error:
    - Need yet a separate test set to evaluate performance

Overfitting and Underfitting

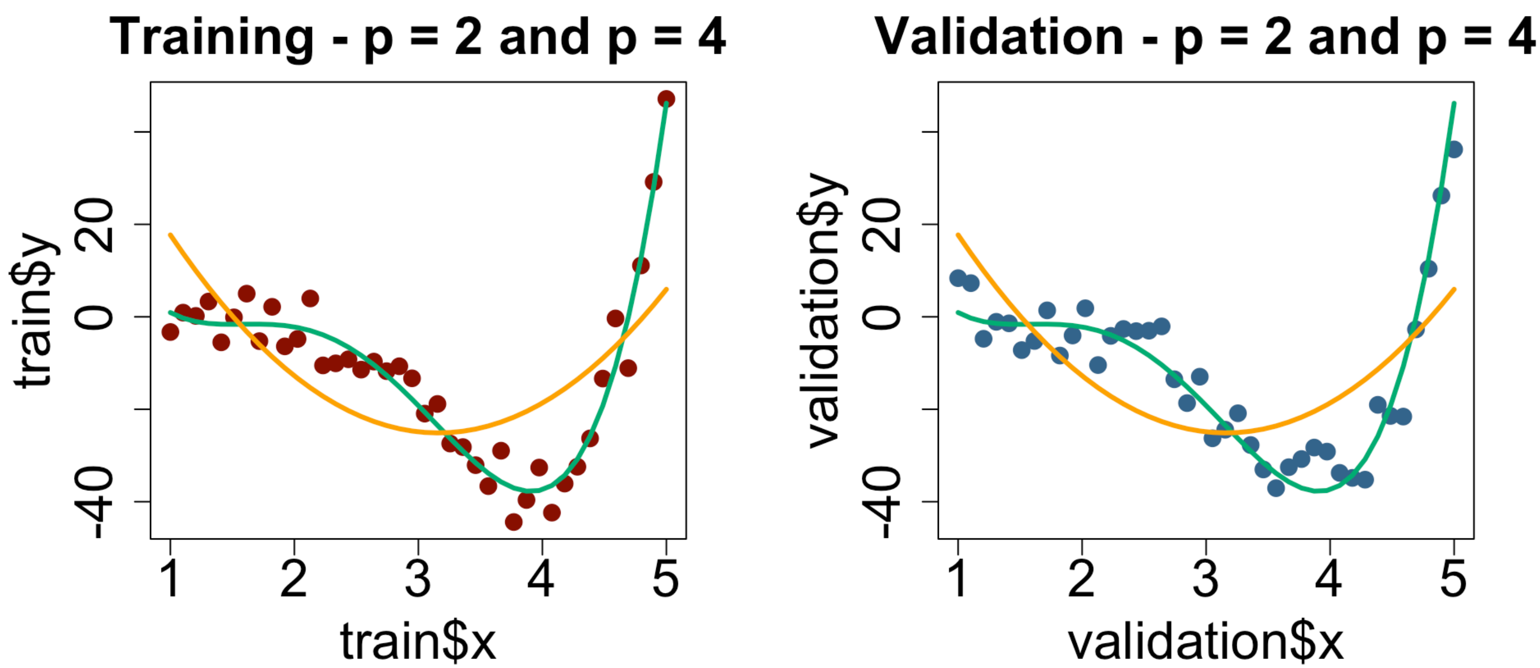
- Overfitting occurs when the model captures the noise rather than the trend in the training data:
  - The model fits the training data ‘too well’
  - Model closely follows the quirks of the training data but does not generalize/predict well
  - Fails to capture the true underlying relationship between features and outcome
- Underfitting occurs when the model can’t capture the true underlying relationship between features and outcome
- The more complex/flexible the model the higher the potential for overfitting
- Need to choose the right level of complexity to avoid overfitting
- One way to accomplish this is by performing model selection, i.e. comparing different models using a validation set

Overfitting



- **Overfitting** – Model with $p = 25$ fitted the noise in the training data
- In validation set polynomial of order $p = 25$ performs very poorly

Uderfitting



-

Underfitting – Model with $p = 2$ not flexible enough to capture underlying trend - In validation set polynomial of order $p = 2$ performs poorly

Overfitting and Underfitting

Preventing overfitting and underfitting is a central issue in Machine Learning

- Prevent underfitting by considering sufficiently flexible models
- Prevent overfitting by avoiding overly complex models
- In practice we don't know beforehand the right level of complexity
- We choose model complexity/perform model selection using validation set
- Model **complexity/flexibility/capacity** increases with:
    - Number of features
    - Degree of polynomial terms
    - Number of interaction terms
    - Type of model (e.g. nonparametric KNN is much more flexible than linear regression)

Formal ML setting

- Training set: $(\mathbf{X}_1, Y_1), (\mathbf{X}_2, Y_2), \ldots, (\mathbf{X}_n, Y_n)$
- Vector of features for i^{th} sample: $\mathbf{X}_i = (X_{i1}, \ldots, X_{ip})$
- **Only assumption is independence across samples (instances/observations/subjects)**
- $Y = f(X_1, \ldots, X_p) + \epsilon,$
- $f(X_1, \ldots, X_p) = E[Y|X_1, \ldots, X_p]$
- $\epsilon = Y - f(X_1, \ldots, X_p)$
- $E[\epsilon] = E[Y - E[Y|X_1, \ldots, X_p]] = E[Y] - E[E[Y|X_1, \ldots, X_p]] = 0$

Formal ML setting

- In supervised ML goal is to estimate regression function f to make predictions about Y
- In statistics goal is to make inference:
  - which predictor $X_1, \ldots, X_p$ is associated with Y? (e.g. test for association)
  - what is the relationship between each predictor and the response? (e.g. estimate effect size)
- In Biomedical problems interest is usually in both prediction and inference
- In ML we emphasize prediction. In statistics we emphasize inference.
- In general, need more assumptions to make inferences than prediction
  - data follows particular distribution e.g. normal, binomial, multinomial, poisson

Reducible and irreducible errors in prediction

- Assume we have and estimate of the regression function f, $\widehat{f}$ (e.g. from fitting a linear regression)
- The average or expected test mean square error is given by:

$$\begin{aligned} E[MSE] &= E[(Y - \widehat{Y})^2] && = \\ &= E[(f(\mathbf{X}) + \epsilon - \widehat{f}(\mathbf{X}))^2] && = \\ &= E[(f(\mathbf{X}) - \widehat{f}(\mathbf{X}))^2] + Var[\epsilon] \end{aligned}$$

- $Var[\epsilon]$ is the ‘irreducible error’:
  - unknown, intrinsic noise to the problem – can’t do anything about it.
- $E[(f(\mathbf{X}) - \widehat{f}(\mathbf{X}))^2]$ is the reducible error:
  - We can make it smaller by using a better estimate $\widehat{f}$ of f
  - E.g. add higher order terms or use more flexible/complex model (e.g. KNN)
  - Goal is to estimate f to make the reducible error as small as possible

Bias–Variance Trade-off

- The reducible error can in turn be partitioned as:

$$
\begin{aligned}
& E\left[\left(f(\mathbf{X})-\widehat{f}(\mathbf{X})\right)^2\right] = \\
= {} & E\left[\left(f(\mathbf{X})-E[\widehat{f}\,](\mathbf{X})\right)^2\right] + E\left[\left(\widehat{f}(\mathbf{X})-E[\widehat{f}\,](\mathbf{X})\right)^2\right] = \\
= {} & \operatorname{Bias}\left(\widehat{f}(\mathbf{X})\right)^2 + \operatorname{Var}\left(\widehat{f}(\mathbf{X})\right)
\end{aligned}
$$

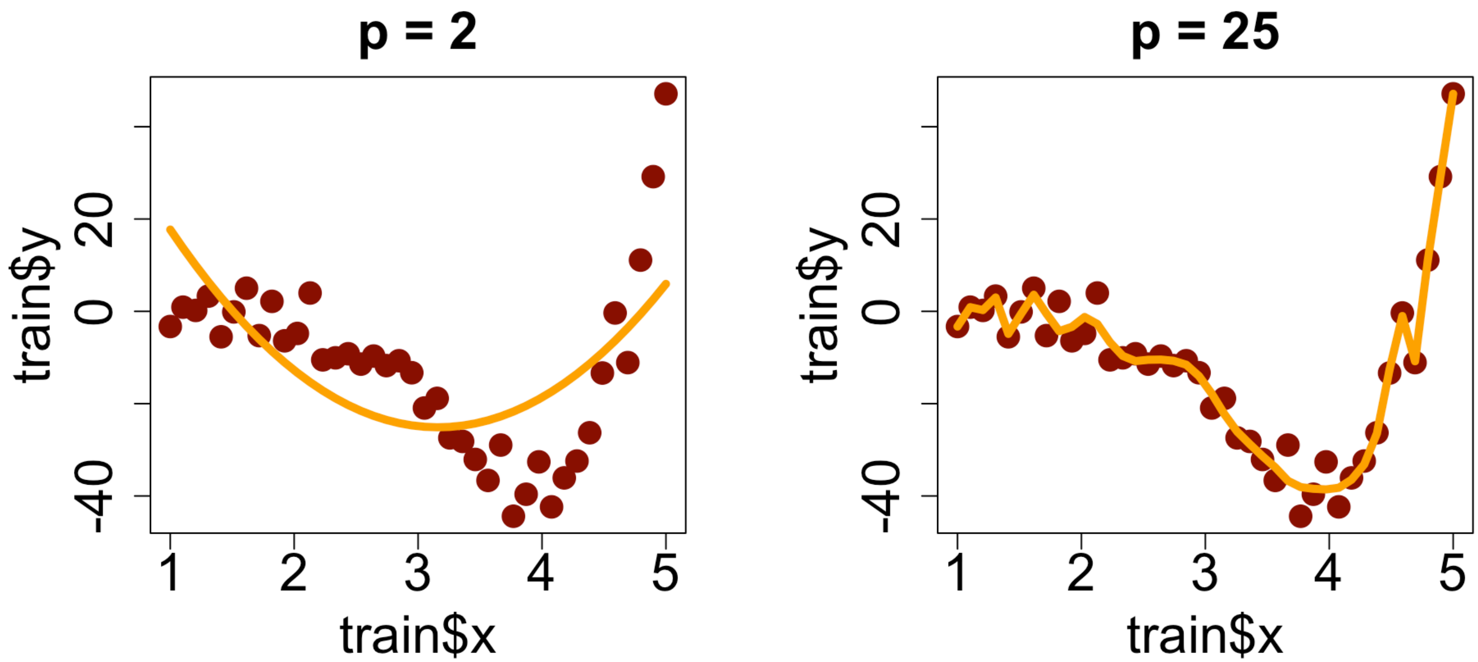
Bias–Variance Trade-off

- Average prediction error = bias^2 + variance
- **Bias** term represents how far the average estimated regression function is from the true regression function across training sets
- **Variance** term represents how variable the estimated regression function is across training sets

Bias–Variance Trade-off

- Consider our simulated example:
  - True relationship is: $Y = e^X \cos(X) + \epsilon, \; \epsilon \sim N(0, \sigma^2)$
- We'll compare a simple vs. complex linear regression fitted on training data generated under model above:
  - Simple: $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$
  - Complex: $Y = \beta_0 + \beta_1 X + \ldots + \beta_{25} X^{25} + \epsilon$

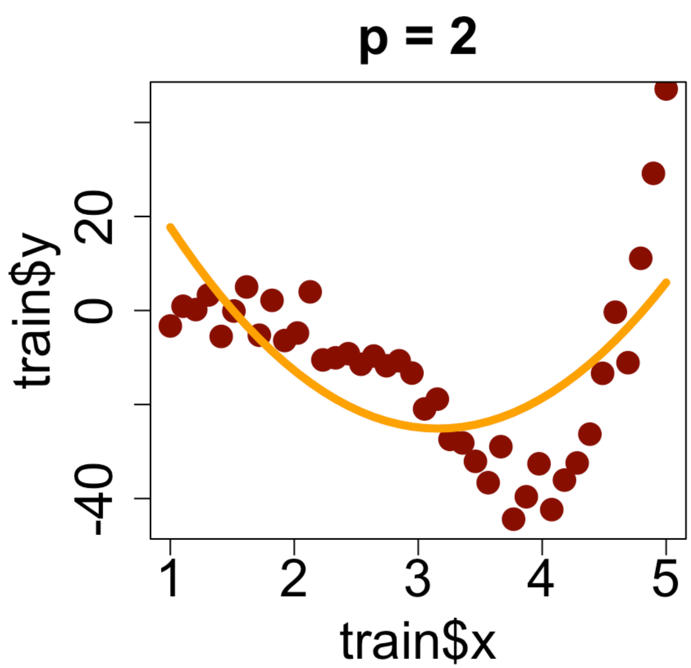
Bias–Variance Trade-off

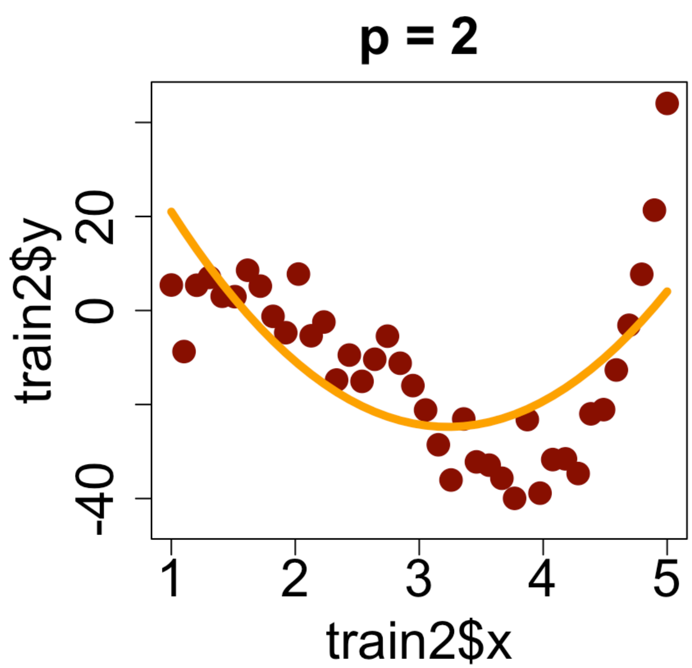


Which fitted curve will change more if the training set changes?

Bias–Variance Trade-off

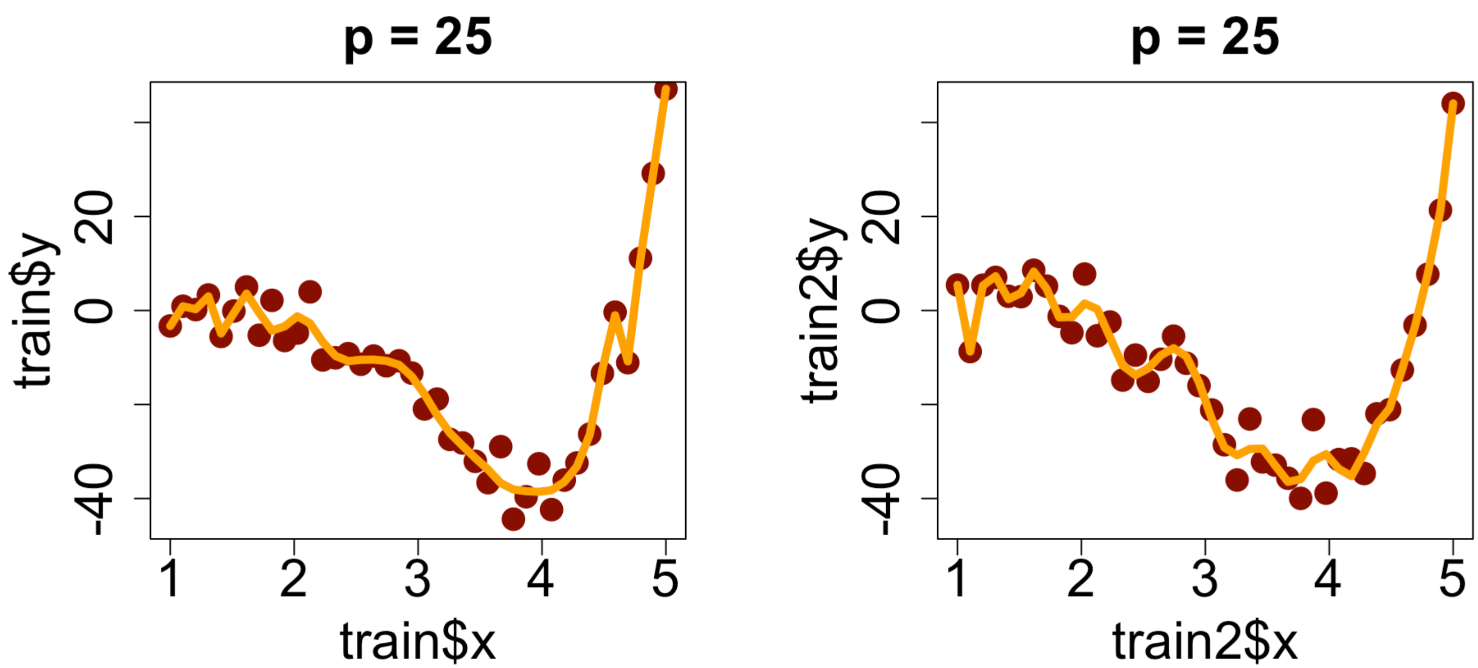
Two different training sets





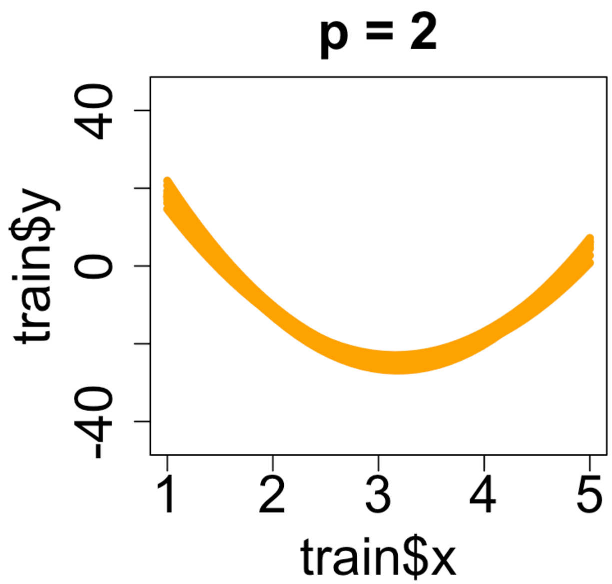
Bias–Variance Trade-off

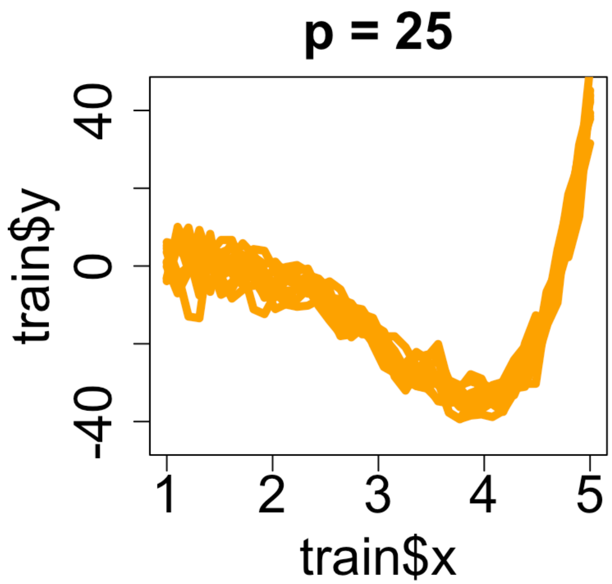
Two different training sets



Bias–Variance Trade-off

10 different training sets

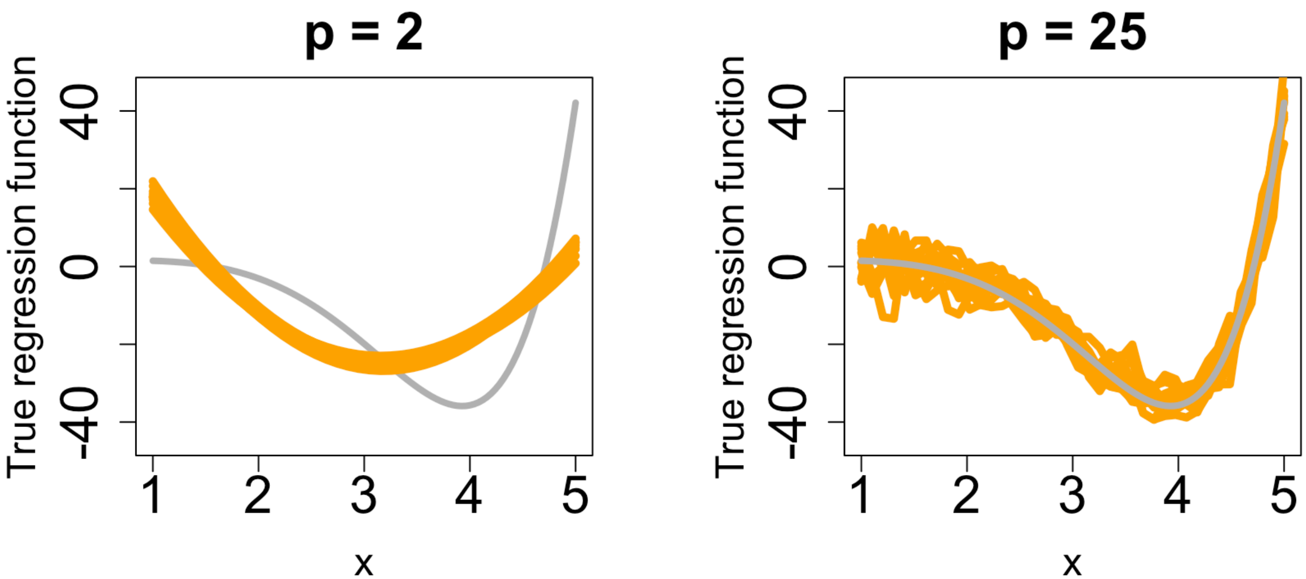




- Simple model $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$ exhibits **low variance**
- Complex model $Y = \beta_0 + \beta_1 X + \ldots + \beta_{25} X^{25} + \epsilon$ exhibits **high variance**

Bias–Variance Trade-off

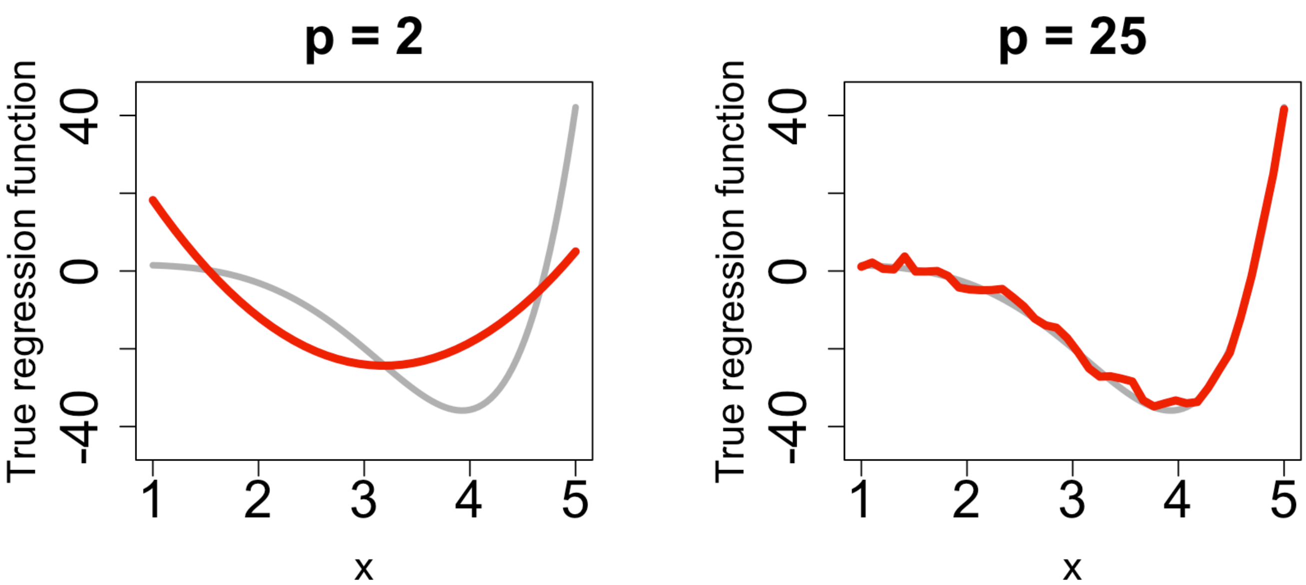
Let's add the true regression function: $f(x) = exp(x)cos(x)$



Which model approximates the true regression line more closely?

Bias–Variance Trade-off

Average regression line over 10 different training sets



- Simple model $Y = \beta_0 + \beta_1 X + \epsilon$ exhibits **high bias**
- Complex model $Y = \beta_0 + \beta_1 X + \ldots + \beta_{25} X^{25} + \epsilon$ exhibits **low bias**

Bias–Variance Trade-off

- Goal of model selection in regression is to minimize reducible error
- Reducible error = bias^2 + variance
- Complex/flexible models exhibit high variance but low bias
- Simple/rigid models exhibit low variance but high bias
- In model selection we find a model with 'just the right' amount of complexity:
  - model complexity with the best bias and variance trade-off
  - that minimizes reducible error