Machine Learning

Juan Pablo Lewinger

7/13/2021

What is Machine Learning?

- Example of problem/task *not* well suited for machine learning: multiply two 5-digit numbers
  - A human can write a computer program (teach the computer/machine) exact steps/algorithm to follow
  - Computers excel at this kind of mechanical task – humans no so much

What is Machine Learning?

- Example of problem well suited for ML: Spam email recognition
    - Very easy for a human to do
    - But very hard for a human to translate into a well-defined sequence of steps/algorithm
- Idea behind ML: ‘feed’ the computer many examples (data) of spam and non-spam emails
    - Let the machine/computer come up with its own set of ‘rules’
- ‘Feeding’ **features** extracted from the emails and **labels** indicating spam/no-spam

Features and Labels

- Features: Characteristics of the emails human believes relevant to determining spam/no-spam
  - Total number of words (numeric feature)
  - Appearance of keywords/phrases (e.g. ‘act now’, ‘for free’, ‘save $’) (yes/no) (binary feature)
  - Single word frequencies (numeric features)
  - Pairs of words frequencies (numeric features)
  - Email sender in your contact list (yes/no)?
  - Time of day email was sent 0-24 (numeric)
  - Day of week email wass sent (categorical)
- Label: spam/no-spam (binary)

Algorithms

- How does a machine learn by example?
- Need a algorithm to train a model/label generator (e.g. classify a new email as spam/no-spam)
  - Input: features and labels of the training data/examples
  - Output: ‘label generator’ for new instances (without label)
- Example algorithms/method for the spam problem: Naive Bayes Classifier, Support Vector Machines, Discriminant Analysis, logistic regression
- Need a training method to ‘fit’ the model (e.g. gradient descent)

Machine Learning vs. Statistics

- Lots of overlap - they share many techniques/models (e.g. linear and logistic regression, principal components)
- But goals are different
- Statistics focuses on inference:
  - estimation and hypothesis testing of parameters of interest (e.g. odds ratio, regression slope)
  - Establishing association/causation (e.g. is smoking a risk factor for lung cancer?)
- Machine learning focuses on prediction and detection (e.g. does this patient have lung cancer? Is this patient likely to have a relapse?)
- We'll refer to both as 'prediction'

Machine Learning vs. Statistics

- ML Theory makes very few distributional assumptions: $i.i.d$ independent, identically distributed instances/samples
- Statistics (typically but not always) makes lots of distributional assumptions (e.g. data is normally distributed)
- ML assumes less but has the ‘simpler’ goal of prediction
- Statistics assumes more but has the more complex goal of establishing association/causation

Prediction

- Machine Learning emphasizes prediction
- But, why do we want to ‘predict/detect’?
  - to anticipate/predict future (e.g. patient likely to die within 5 years)
  - to avoid costly/unfeasible measure (e.g. estimate body-fat from BMI, detect whether a woman has ovarian cancer based on methylation measurements)
  - to automate/eliminate need for human intervention (e.g. spam detector)
- Two really different meanings of ‘prediction’:
  - prediction of future outcome
  - estimation of unmeasured variable/detection

Artificial Intelligence vs. Machine Learning

- Artificial Intelligence: “The effort to automate intellectual tasks normally performed by humans”*
- Machine Learning is a sub-field AI
- Route finding algorithms (e.g. Waze app) are AI but not ML

* “Deep Learning with R” by Allaire and Chollet

Example of Biomedical applications

- Predict whether a prostate cancer patient will have a recurrence based on gene expression profiles of the tumor
- Identify population structure/ancestry based on millions of single nucleotide polymorphism (SNP) genotypes
- Diagnose cancer subtype based on gene expression or methylation profiles
- Predict mesothelioma (aggressive cancer of the lining of lungs) based on exposure to asbestos, smoking, etc.
- Predict patient length of stay in hospital based on health records
- Identify groups of cities that have similar patterns of air pollution
- Early prediction (years before any symptoms) of likelihood of developing Alzheimer’s disease based on MRI
- Predicting body fat percent based on BMI, gender, and age
- Predict age based on genomewide methylation patterns
- Predict whether an enhancer (a DNA regulatory element) targets a particular gene

Supervised vs. Unsupervised Learning

- In supervised learning the instances have labels/variables we want to predict based on their features
    - Spam detection is a typical example of a supervised learning problem
    - Analogy with learning under the supervision of a teacher that knows the correct answer (labels)
- In unsupervised learning there are features but no labels
    - Goal is to find structure (groups, clusters) in the instances and/or the features
    - Learning is unsupervised because there is ‘no teacher’
    - Cluster analysis and Principal component analysis are typical unsupervised learning techniques

Classification vs. regression

- Supervised learning problems often are of one of two types depending on the type of label
- Classification: when label is binary (spam vs. non-spam) or categorical (3 breast cancer subtypes)
    - e.g. discriminant analysis, logistic regression, support vector machines
- Regression: when label is quantitative/numeric (% body fat, age, survival time)
    - Linear regression, Cox proportional hazards model

Glossary—Machine Learning vs. Statistics

- **feature** = predictor = independent variable = covariate = regressor = exogenous variable (economics)
- **label** = outcome = dependent variable
- **instance** = observation (for us usually subject, patient)
- **algorithm** = model (e.g. linear regression, logistic regression)
- **train** = fit model

Linear Regression

- Linear regression is a simple supervised regression (quantitative response/label) method
- Very useful despite simplicity
- More flexible than appears (can include higher order terms, interactions, splines, etc.)
- Starting point for other methods/algorithms: Discriminant analysis, logistic regression, ridge regression, LASSO, etc.

Example: predicting brain weight from head size

Data from brain weight (grams) and head size (cubic cm) for 237 adults aged 20 or above and classified by gender and age group from Middlesex Hospital, London

One of the main objects of the investigation which forms the subject of this paper, has been to obtain a series of reconstruction formulae, by which it will be possible, when in possession of certain chief measurements of the head, to predict within the limits of normal variation, the approximate weight of the brain

A Study of the Relations of the Brain to the Size of the Head Biometrika (1905)

The subject of brain-weight in man has for a long time been given considerable attention by anatomists and anthropologists. The reason for this is obvious. Since the brain is the organ of the mind it appeared to earlier workers that size of brain ought to be an index of intellectual capacity.

Biometrical Studies on Man: I. Variation and Correlation in Brain-Weight Biometrika (1905)

Brain weight example

```
setwd("/Users/jp/Google Drive/Teaching/Machine Learning/2021/Lectures/Lecture 2 - Linear regres
brain = read.table("brain.txt", header=T)
head(brain)
```

```
##   Sex Age Head.size Brain.weight
## 1   1   1      4512         1530
## 2   1   1      3738         1297
## 3   1   1      4261         1335
## 4   1   1      3777         1282
## 5   1   1      4177         1590
## 6   1   1      3585         1300
```

Brain weight example

Split data into training (70%) and test (30%) data

```
set.seed(301)
n = nrow(brain)
n
```

```
## [1] 237
```

```
trainset = sample(1:n, floor(0.7*n))
head(trainset, 10)
```

```
##  [1] 220 218 141   5  57 234  47  26  52   9
```

Brain weight example

Split data into training (70%) and test (30%) data

```
set.seed(301)
brain_train = brain[trainset,]; n_train = nrow(brain_train)
brain_test = brain[-trainset,]; n_test = nrow(brain_test)
dim(brain_train)
```

```
## [1] 165   4
```

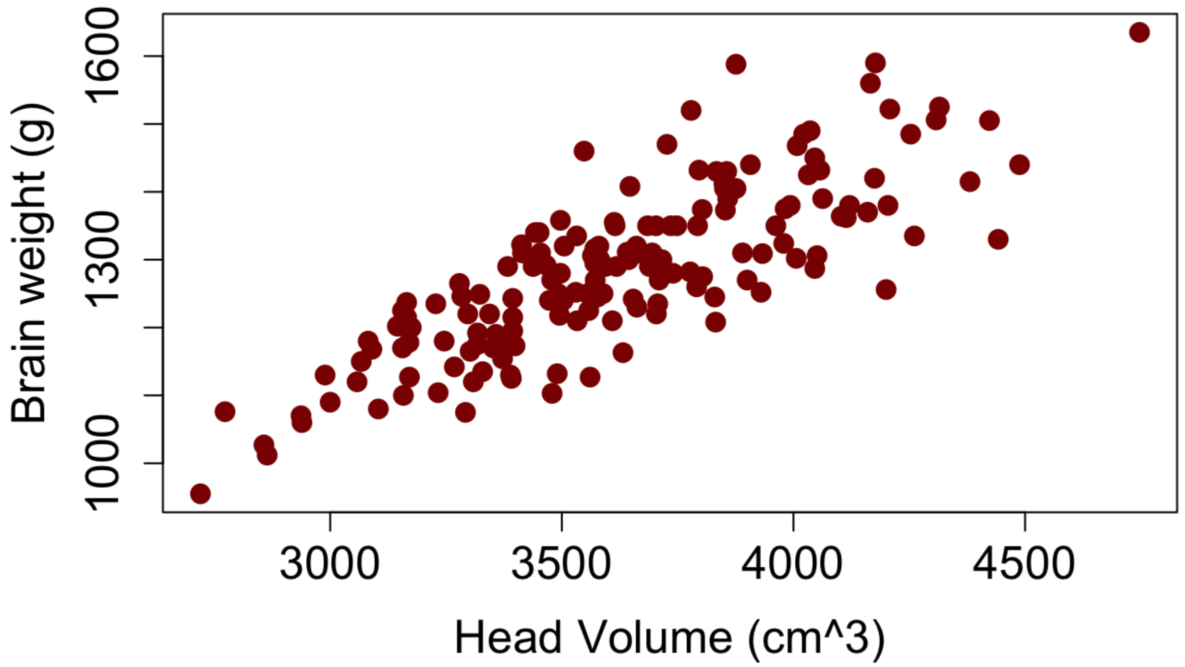
```
dim(brain_test)
```

```
## [1] 72  4
```

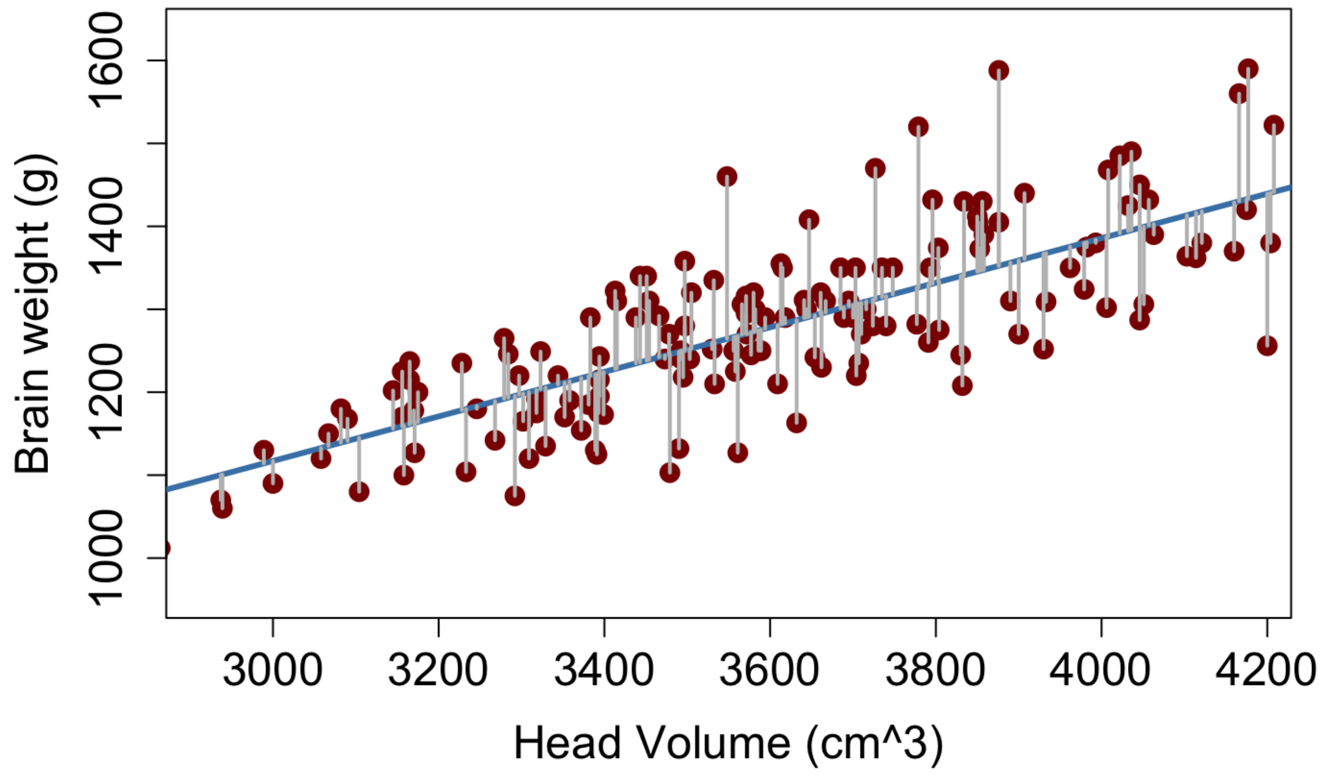
Brain weight example

Clear positive relationship between head volume and brain weight

```
par(mar=c(8,9,1,1))
plot(brain_train$Brain.weight ~ brain_train$Head.size,
     pch=16, cex=1.5, col='red4', xlab='Head Volume (cm^3)',
     ylab='Brain weight (g)', cex.lab=1.5, cex.axis=1.5, cex.main=1.5)
```



Linear regression fit



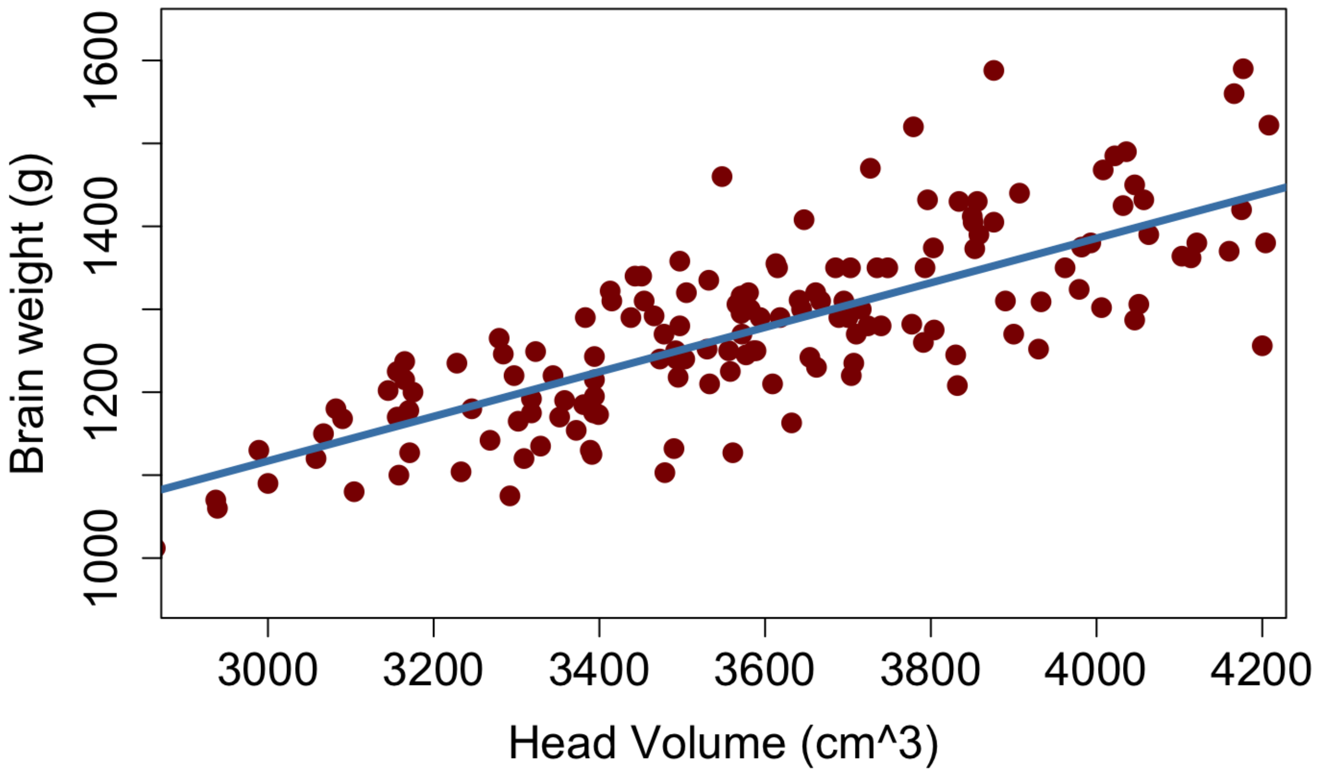
Simple Linear Regression on Brain data

Fit linear model to the training portion of the brain weight data

```
brain_lm0 = lm(Brain.weight ~ Head.size, data = brain_train)
summary(brain_lm0)
```

```
##
## Call:
## lm(formula = Brain.weight ~ Head.size, data = brain_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -183.437  -50.277   -3.188   47.098  235.591
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 311.29654   54.67694   5.693 5.67e-08 ***
## Head.size     0.26860    0.01501  17.891  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 71.58 on 163 degrees of freedom
## Multiple R-squared:  0.6626, Adjusted R-squared:  0.6605
## F-statistic: 320.1 on 1 and 163 DF,  p-value: < 2.2e-16
```

Simple Linear Regression on Brain data



Prediction

- Using LR we constructed a model that approximates brain weight based on head size in our (training) data
- But our goal is to predict brain weight based on head size in the population
- What population? **A population the training data can be considered a random sample of**
  1. Patients aged 20+ in London's Middlesex Hospital in 1905
  2. Population of London inhabitants aged 20 or older in 1905
  3. Population of London inhabitants aged 20 or older today
- If the model predicts in the population the training data was sampled from we say that the model *generalizes*

How do we know if the model generalizes?

- Ideally we'd need to compare the brain weight predicted by the model with the true brain weight in the entire target population to *compute the average error* incurred by the model.
- Obviously we can't do that. Next best thing is to take a 'test' sample from the target population and use it to estimate the average error by the *sample average error*
- How do I get such a sample? We set aside part of the original data for this purpose before building our model.
- We split the data into training and testing
    - Use training data for fitting the model
    - Use test data to evaluate/estimate how well the model performs

Prediction performance metrics

y_i is the true observed outcome (e.g. brain weight) and

$\widehat{y_i} = \widehat{\beta_0} + \widehat{\beta_1} x_i$ is the outcome predicted by the linear model for observation i

$y_i - \widehat{y_i}$ is the error in prediction for observation i

- Mean Square error:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{y_i})^2$$

- Root Mean Square error:

$$\text{RMSE} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (y_i - \widehat{y_i})^2}$$

- Root Mean Square error is perhaps easier to interpret as it's on the same scale as the outcome (e.g. grams instead of square grams)

Prediction performance metrics in R

```r
sqrt(sum((residuals(brain_lm0))^2)/(nrow(brain_train)-2))
```

```
## [1] 71.58431
```

```r
# Gives Residual standard error above
RMSE_train0 = sqrt(sum((residuals(brain_lm0))^2)/nrow(brain_train))

# RMSE
RMSE_train0
```

```
## [1] 71.14914
```

Prediction performance metrics

R^2, proportion of variance explained

$$\text{RSS} = \sum_{i=1}^{n} (y_i - \widehat{y_i})^2$$

$$\text{TSS} = \sum_{i=1}^{n} (y_i - \bar{y})^2$$

$$R^2 = \frac{\text{TSS}-\text{RSS}}{\text{TSS}} = 1 - \frac{\text{RSS}}{\text{TSS}}$$

```
RSS <- sum((residuals(brain_lm0))^2)
TSS <- sum((brain_train$Brain.weight - mean(brain_train$Brain.weight))^2)
R2_train0 = 1 - RSS/TSS
R2_train0
```

```
## [1] 0.662597
```

Which coincides with R^2 value in `lm` results above

Discussion Question

Can we evaluate prediction performance on the same data set used for building the model, i.e. the training data?

Evaluating (estimating) prediction performance

- Evaluating performance using training data **overestimates performance**
- We'd get an underestimate of true population prediction performance
- Why? Model was chosen so as to make MSE as small as possible in the training data.
- Analogous to using a practice exam to assess students
- Need to assess performance in yet **unseen/unused data**
- We **reserved test data** only for assessing performance
- Trade-off between:
    - better model (more training data) and
    - more accurate performance evaluation (more test data)
- Common split is **train=70-80%** and **test=20-30%** of data
- We will see next class that we can do better by using multiple training/testing splits

Prediction performance in weight data

```
pred0 = predict(brain_lm0, newdata=brain_test)
head(pred0)
```

```
##        1        2        6        7        8       13
## 1523.241 1315.341 1274.245 1327.966 1267.261 1289.018
```

```
head(cbind(brain_test, predicted=pred0))
```

```
##    Sex Age Head.size Brain.weight predicted
## 1    1   1      4512         1530  1523.241
## 2    1   1      3738         1297  1315.341
## 6    1   1      3585         1300  1274.245
## 7    1   1      3785         1400  1327.966
## 8    1   1      3559         1255  1267.261
## 13   1   1      3640         1355  1289.018
```

Prediction performance in weight data

```
n_test = nrow(brain_test)

RMSE_test0 =
  sqrt(sum((brain_test$Brain.weight - pred0)^2)/n_test)

RMSE_test0
```

```
## [1] 74.83068
```

```
RMSE_train0
```

```
## [1] 71.14914
```

As expected, $RMSE_{test} > RMSE_{train}$

Prediction performance in weight data

```r
R2_test0 = 1 -
  sum((brain_test$Brain.weight - pred0)^2)/
  sum((brain_test$Brain.weight - mean(brain_test$Brain.weight))^2)
```

```r
R2_test0
```

```
## [1] 0.571456
```

```r
R2_train0
```

```
## [1] 0.662597
```

As expected, $R^2_{test} < R^2_{train}$

Warning on R squared

- For training set $R^2 = corr(y, \hat{y})^2$
- **Not true** in test set: $R^2 \neq corr(y, \hat{y})^2$
- In **training set** $0 \leq R^2 \leq 1$
- In **test set** R^2 can be negative!
- Negative R^2 in test set indicates extremely poor prediction!

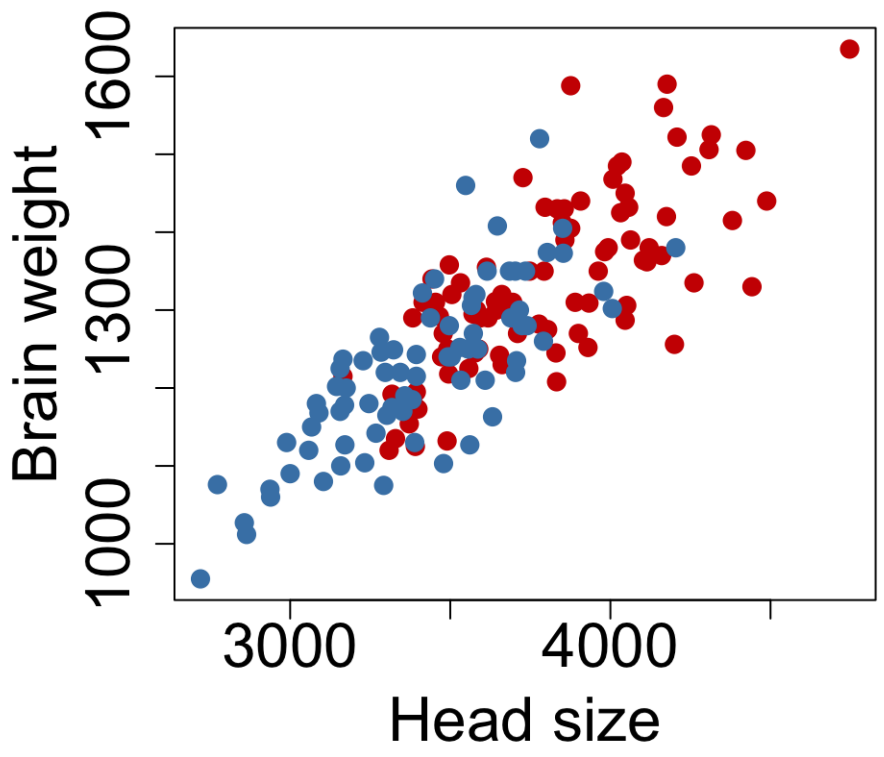
More complex models

- Include sex and age in addition to head size
- Need to convert `Sex` and `Age` to categorical variables before running `lm`

```r
brain$Sex = factor(brain$Sex, levels=1:2, labels=c("Male", "Female"))
brain$Age = factor(brain$Age, levels=1:2, labels=c("20-46", "46+"))
```

More complex models

- Plot by `Sex`, distinguishing Males (red) and Females (blue)



- looks like having separate intercepts may improve prediction

More complex models

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

```
##
## Call:
## lm(formula = Brain.weight ~ Head.size + Sex, data = brain_train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -183.796  -50.258   -3.397   48.411  233.505
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 334.03647   68.47214   4.878 2.54e-06 ***
## Head.size     0.26328    0.01786  14.742  < 2e-16 ***
## SexFemale    -7.35880   13.28798  -0.554     0.58
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 71.74 on 162 degrees of freedom
## Multiple R-squared:  0.6632, Adjusted R-squared:  0.6591
## F-statistic: 159.5 on 2 and 162 DF,  p-value: < 2.2e-16
```

More complex models

Computed train and test RMSE and R^2 for this modes (code not shown):

```
## RMSE_train0  RMSE_test0 RMSE_train1  RMSE_test1
##       71.15       74.83       71.08       74.17
```

- Adding `Sex` to the linear regression model yields marginal improvement in prediction performance

Interactions and higher order terms

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

```
##                          Estimate   Std. Error    t value     Pr(>|t|)
## (Intercept)         375.99706160  90.54334022  4.1526750 5.316234e-05
## Head.size             0.25226302   0.02367906 10.6534224 2.154458e-20
## SexFemale           -99.20658684 130.09215310 -0.7625870 4.468256e-01
## Head.size:SexFemale   0.02564627   0.03613451  0.7097445 4.788892e-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)     -3.592675e+02 3.927191e+02 -0.9148207 0.36164543
## Head.size        6.389791e-01 2.153420e-01  2.9672762 0.00346000
## I(Head.size^2) -5.061113e-05 2.935544e-05 -1.7240803 0.08660095
```

Model selection

- In practice we want to choose from many possible models
  - Include/exclude particular features?
  - Include interactions or higher order terms?
  - Competing models (e.g. linear regression vs. KNN)
- **How do we choose?**

Model selection

- Pick model with smallest training MSE or highest training R^2 is bad idea.
- Right thing to do is split data into training and testing as before and choose model with smallest **test** MSE or highest **training** R^2.
- This is a fair way to compare prediction performance across different models.
- BUT, how do we get the final estimate of prediction performance?
- 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 similar 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 (we'll assume so for now)
- A technique called cross-validation can be used for the more typical situation when n is not large

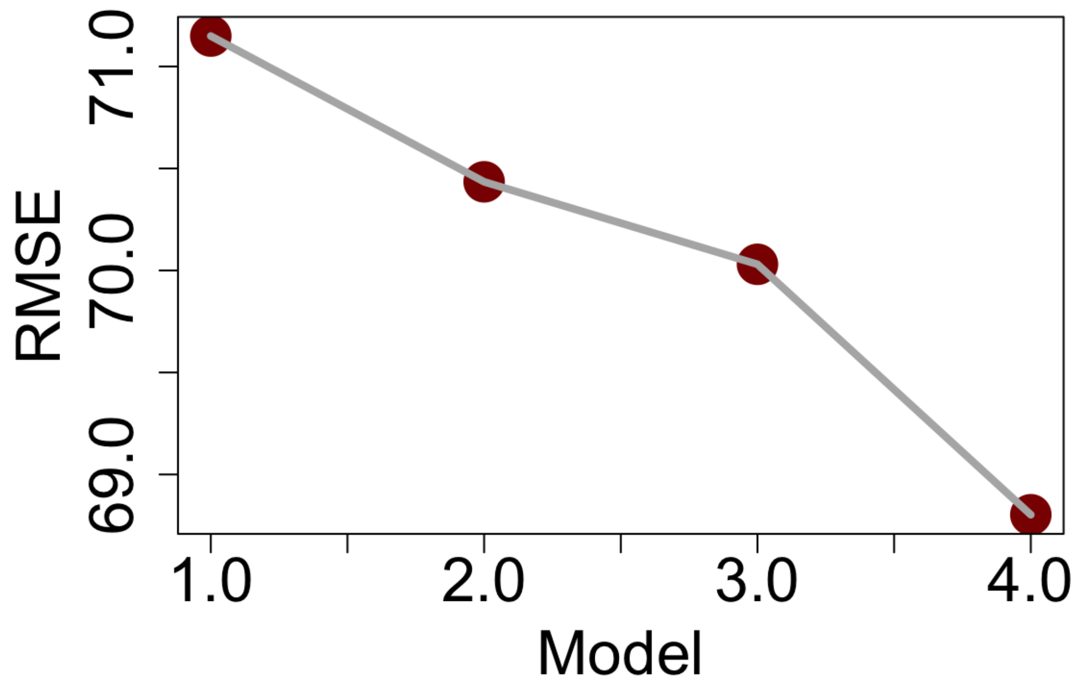
Model selection – example

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

```r
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

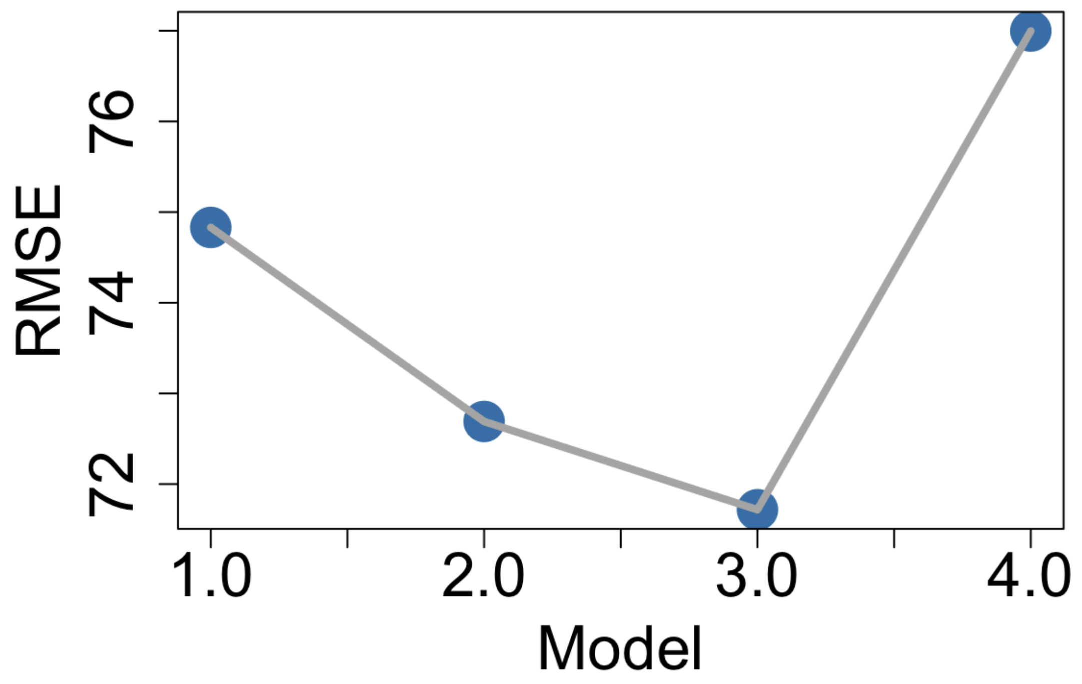
Let’s look at their **training** RSMEs:



Which model should we choose?

Model selection – example

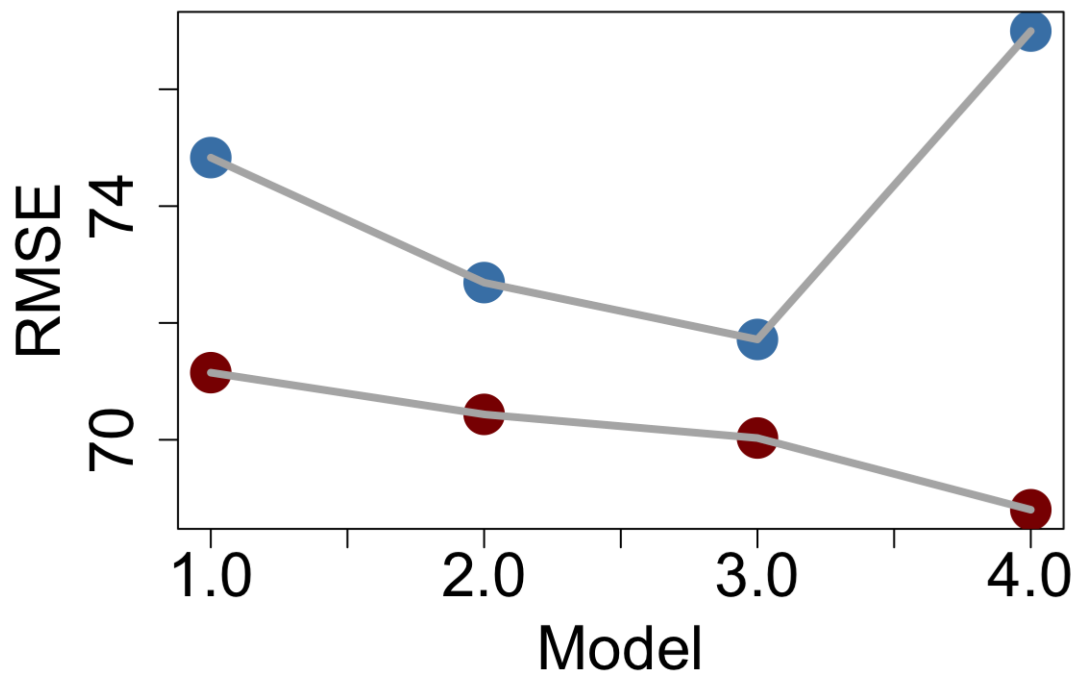
Let's look at their **validation** RSMEs:



Which model should we choose?

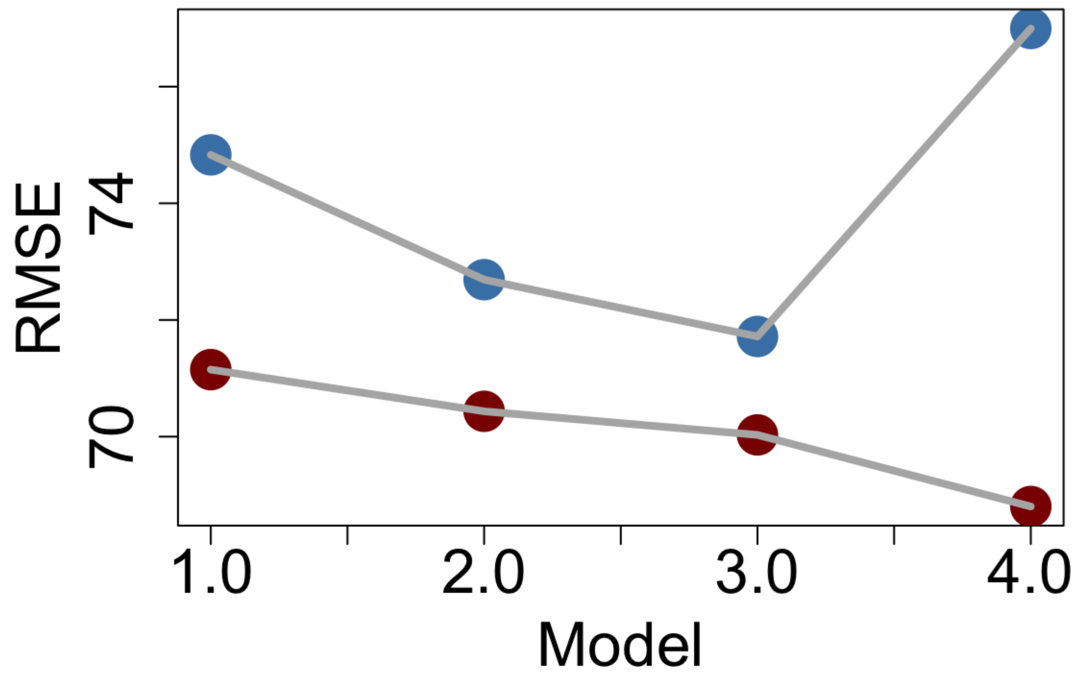
Model selection – example

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



Which model should we choose?

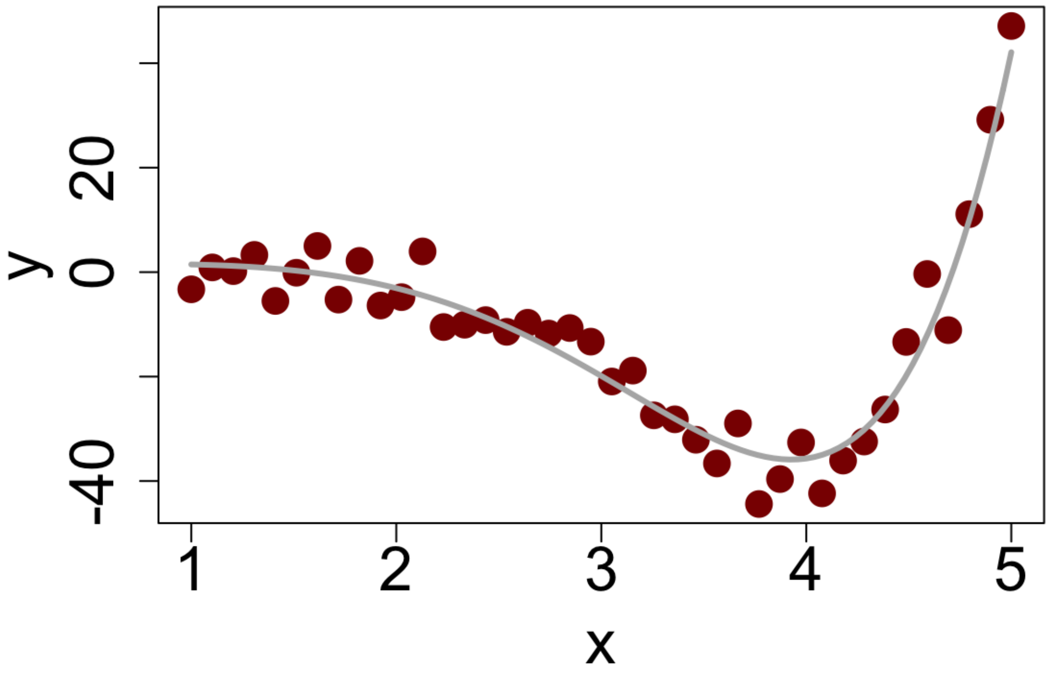
Model selection – example



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

Model selection – 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

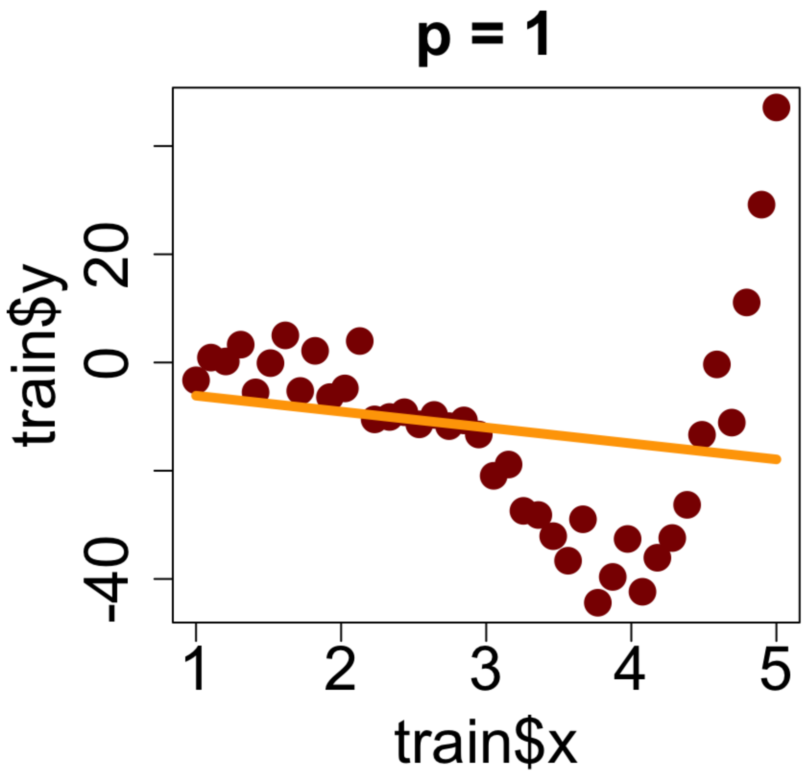
We'll fit polynomial regression models of increasing degree/complexity:

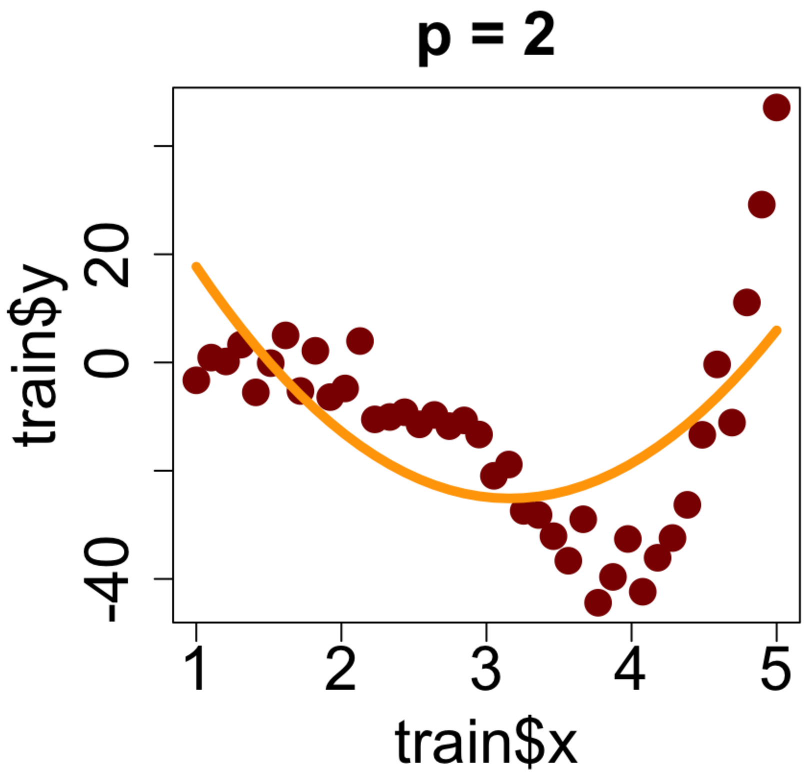
$$
\begin{aligned}
&Y = \beta_0 + \beta_1 X + \epsilon \\
&Y = \beta_0 + \beta_1 X + \beta_1 X^2 + \epsilon \\
&\vdots \\
&Y = \beta_0 + \beta_1 X + \beta_1 X^2 + \ldots + \beta_{25} X^{25} + \epsilon
\end{aligned}
$$

Model selection – simulation

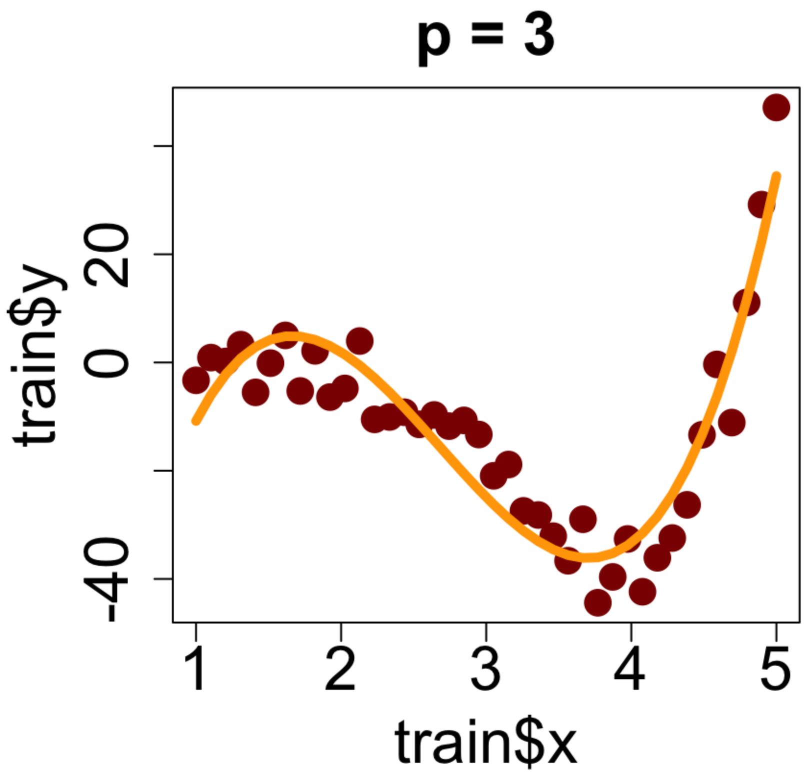
```
fit1 = lm(y ~ x, data = train)
fit2 = lm(y ~ x + I(x^2), data = train)
fit3 = lm(y ~ x + I(x^2) + I(x^3), data = train)
fit4 = lm(y ~ x + I(x^2) + I(x^3) + I(x^4), data = train)
fit5 = lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5), data = train)
fit6 = lm(y ~ x + I(x^2) + I(x^3) + I(x^4) + I(x^5) + 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)
```

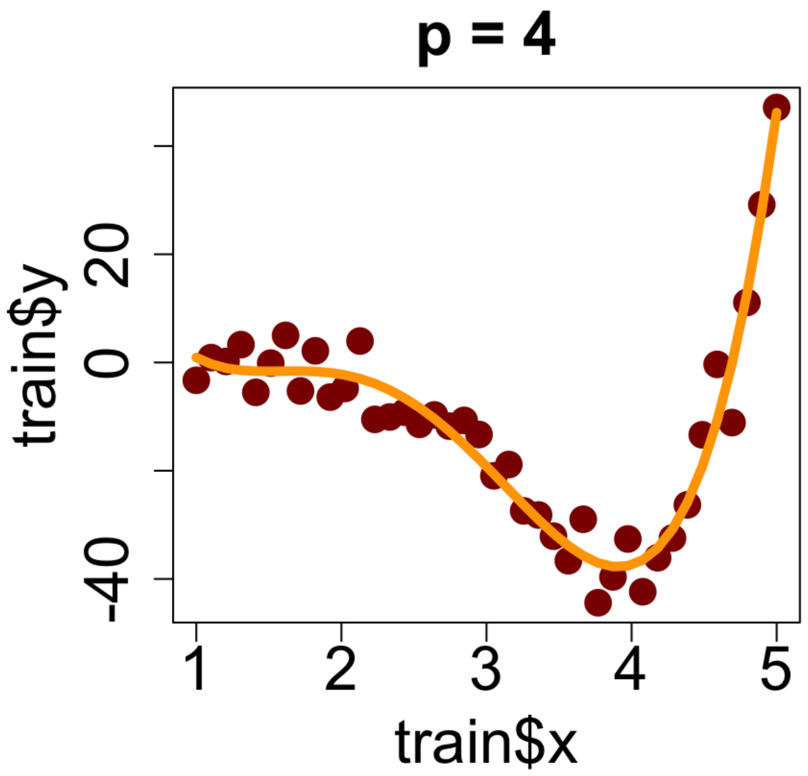
Model selection – simulation



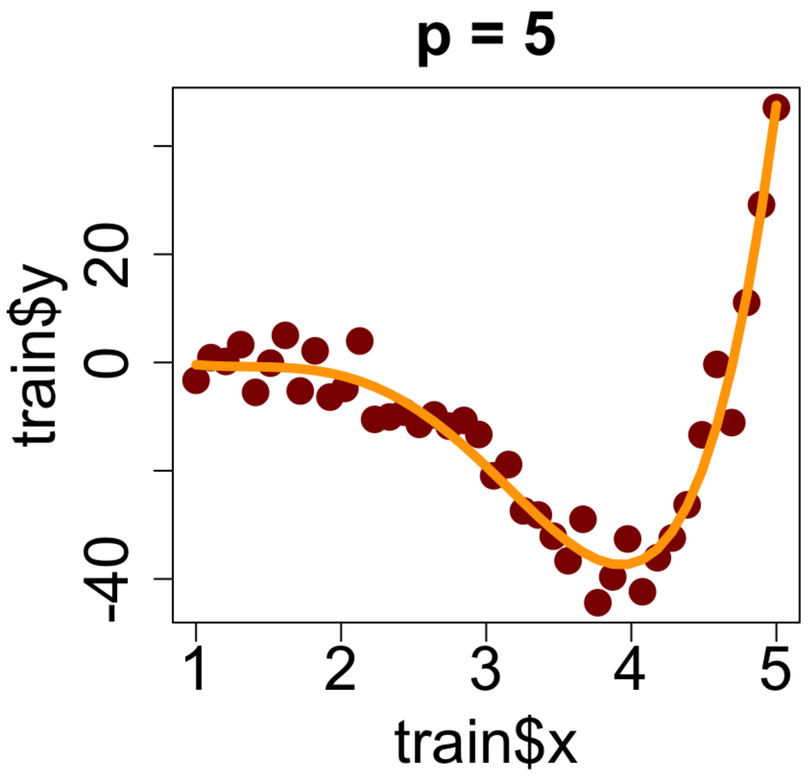


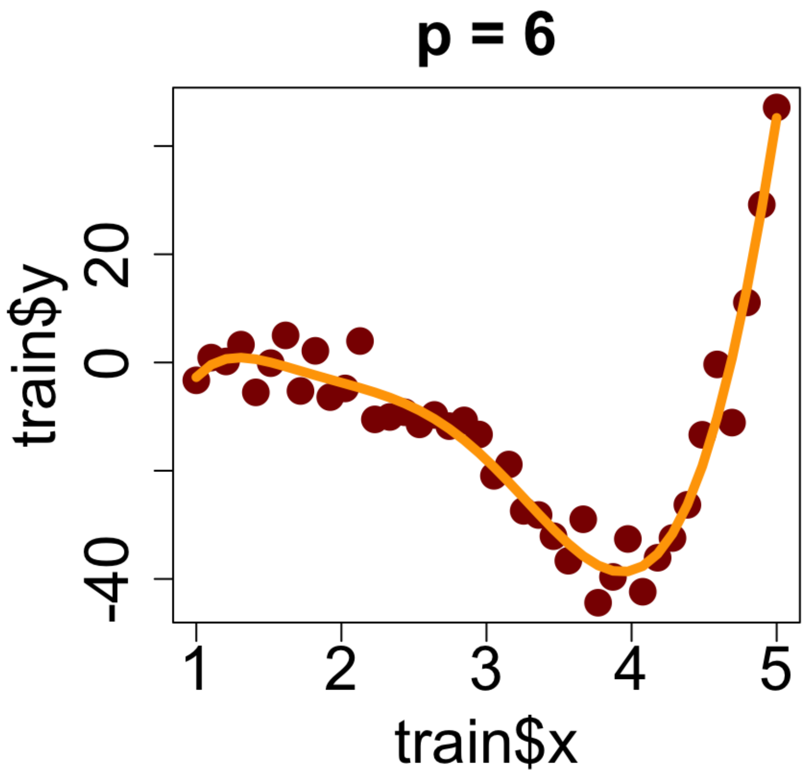
Model selection – simulation



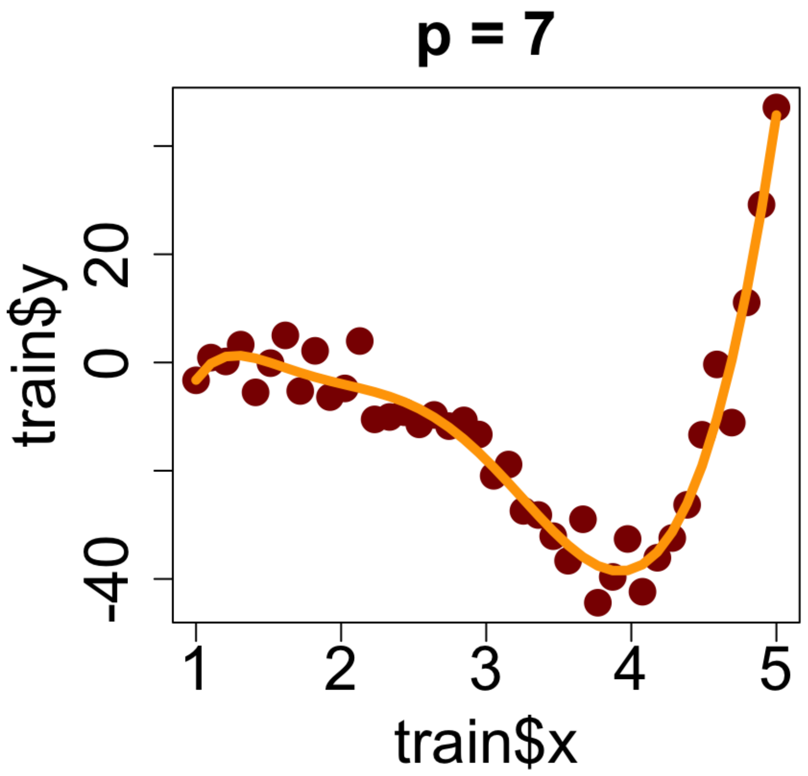


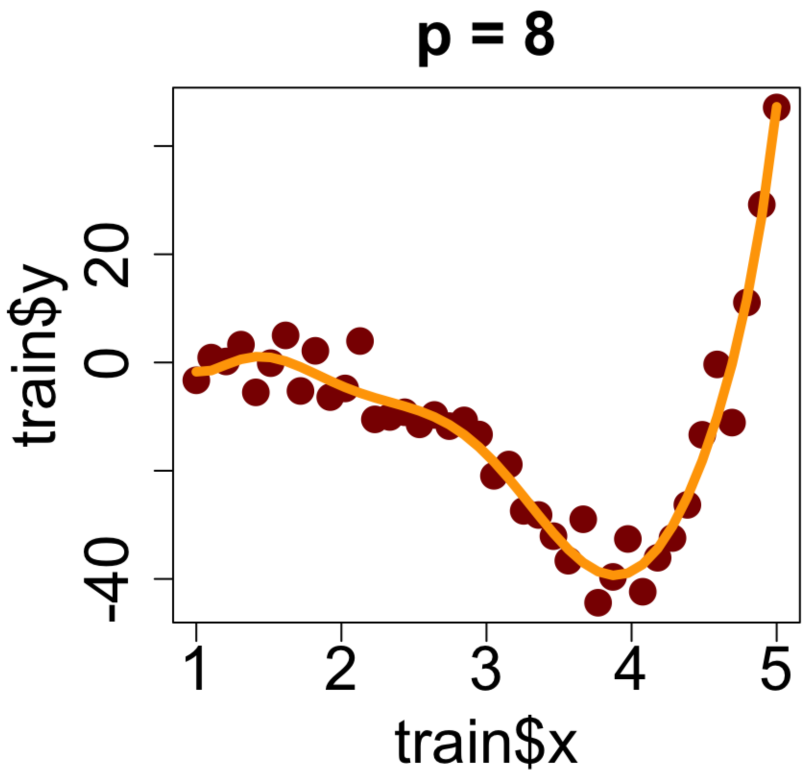
Model selection – simulation



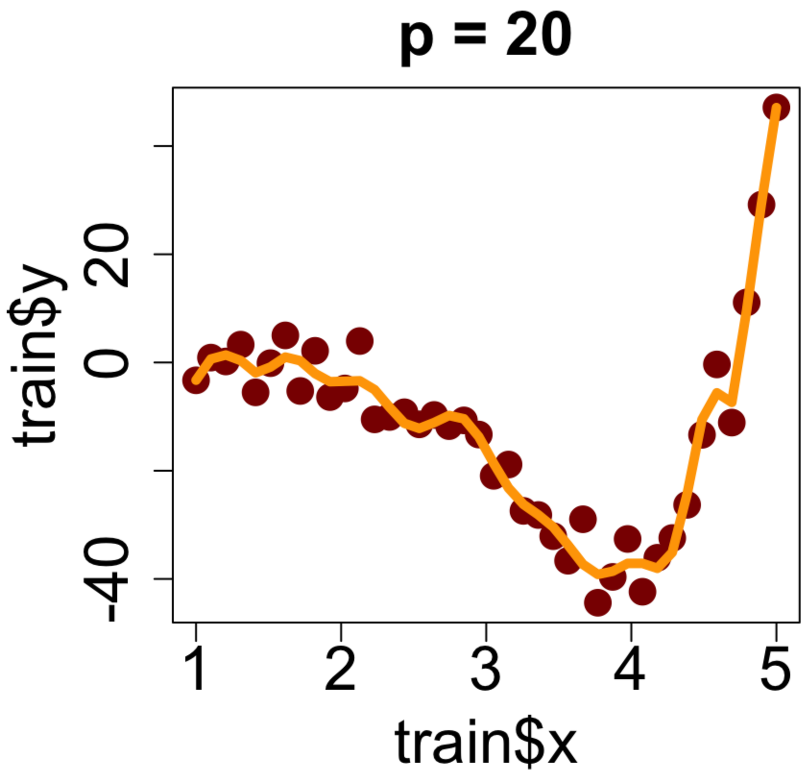


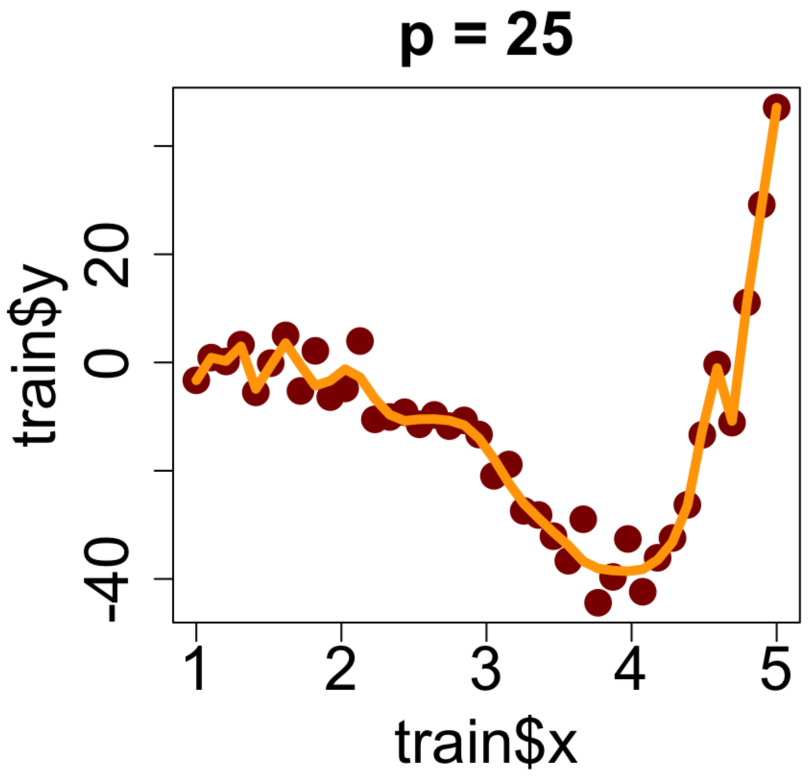
Model selection – simulation





Model selection – simulation

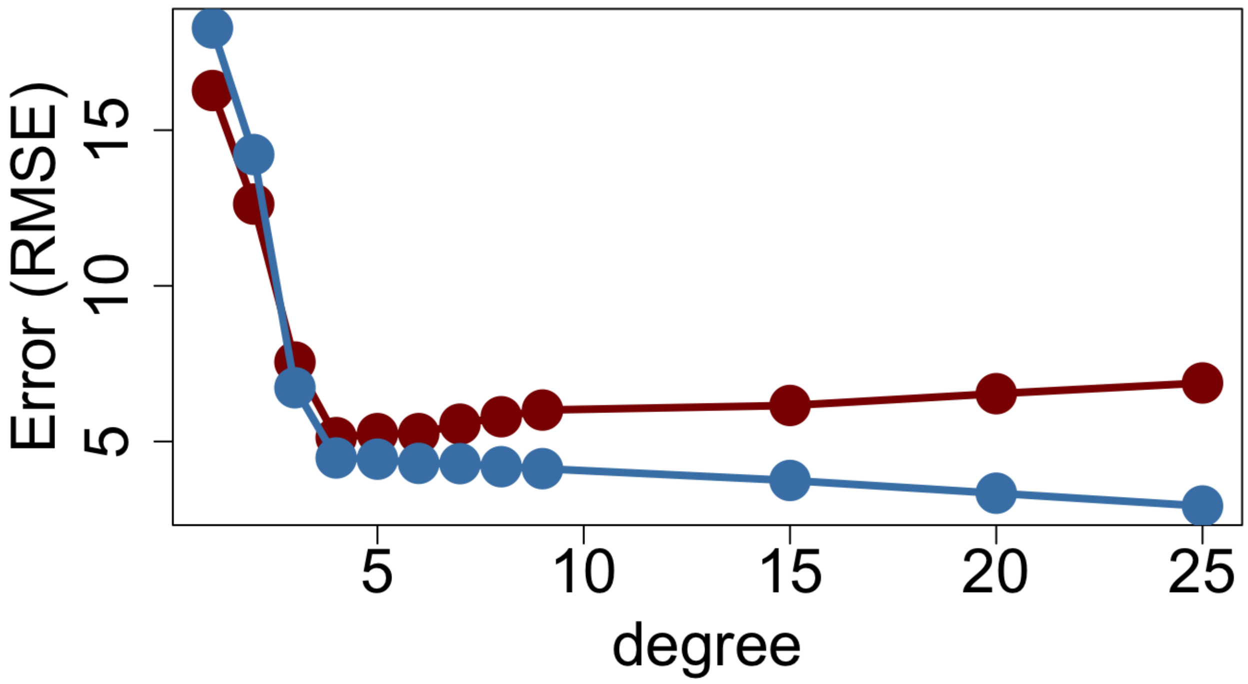




Model selection – simulation

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   18.28 14.22 6.73 4.47 4.43 4.30 4.29 4.19 4.13 3.75 3.34 2.93
## Validation 16.27 12.63 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   18.28 14.22 6.73 4.47 4.43 4.30 4.29 4.19 4.13 3.75 3.34 2.93
## Validation 16.27 12.63 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 $p = 4$)

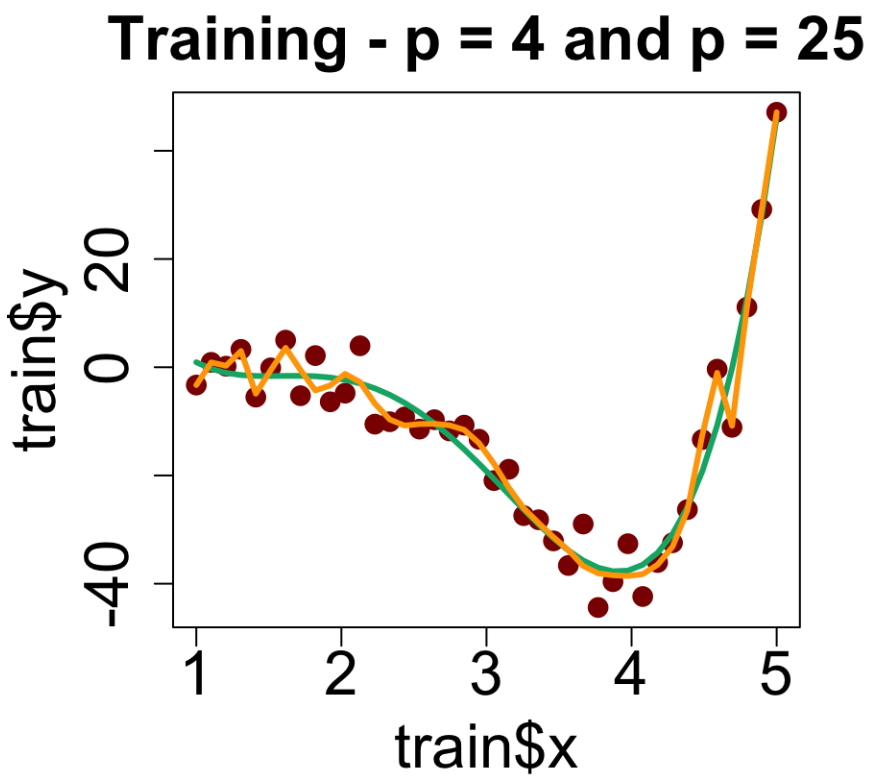
Summary

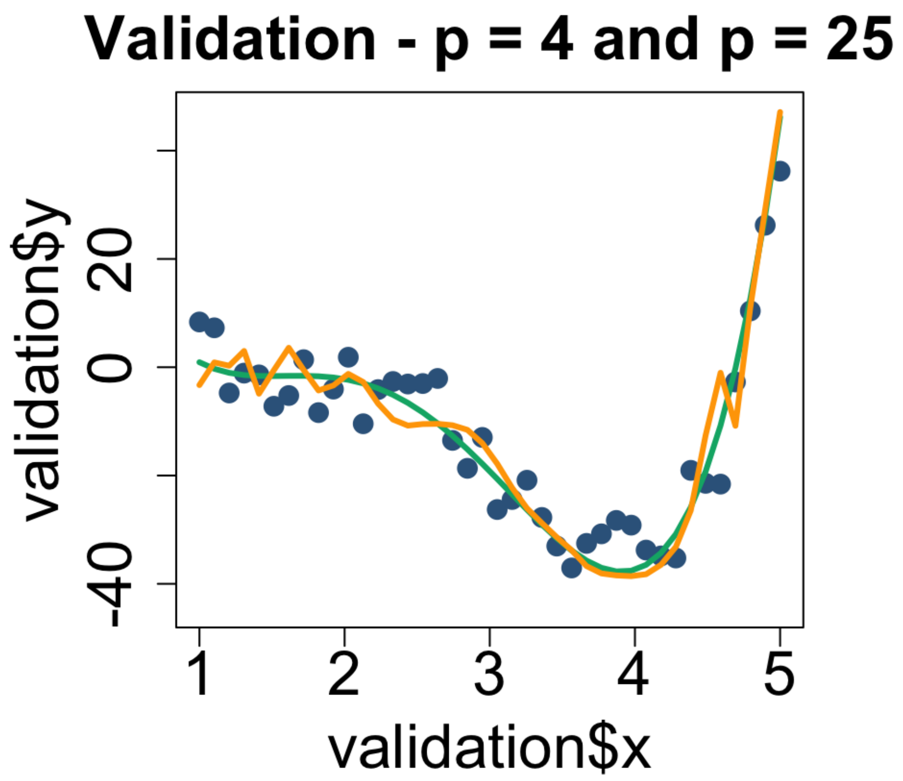
- We typically want try several models because we don’t know beforehand which one will predict best
- Cannot compare models based on the training error because more complex models will always have smaller training error
- Should use a separate validation set to compare models
- But validation error does 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 achieve this is by comparing different models using a validation set

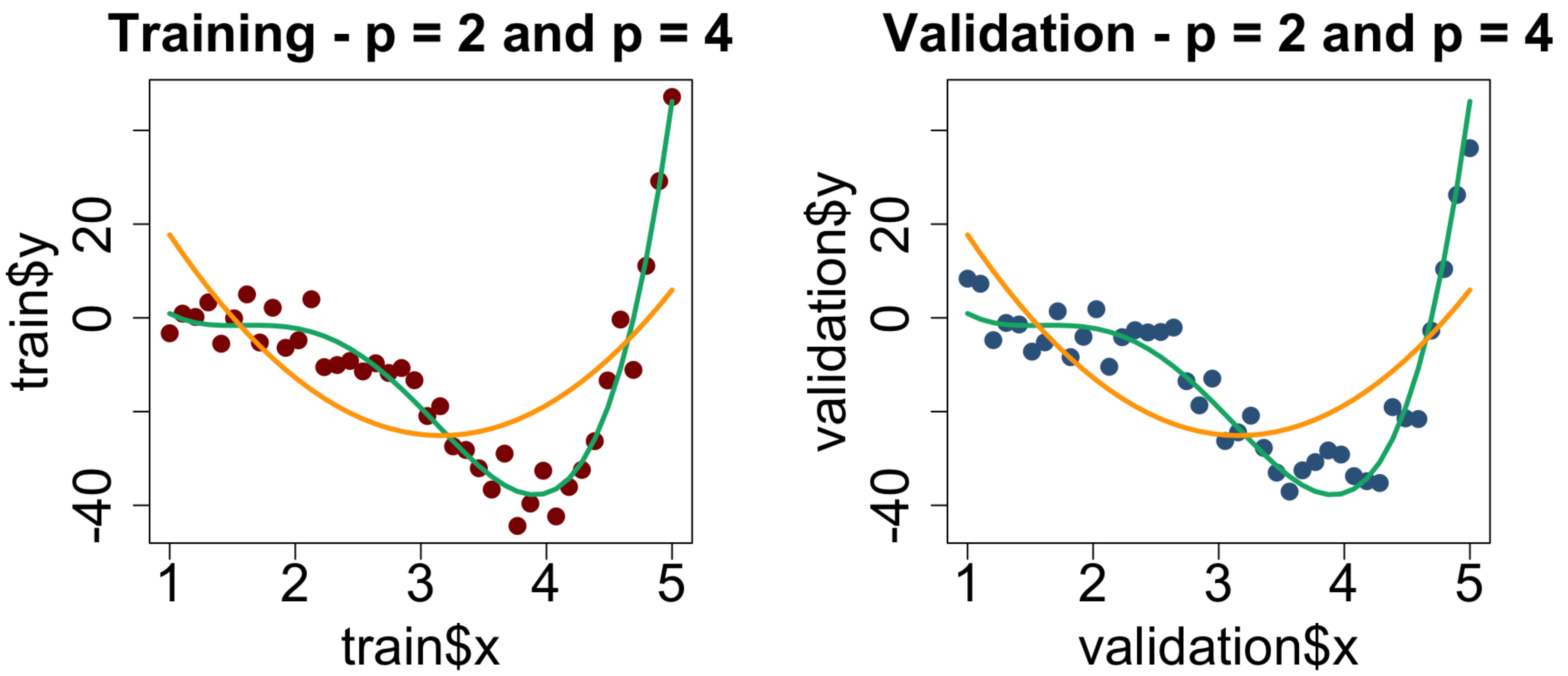
Overfitting





- **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)

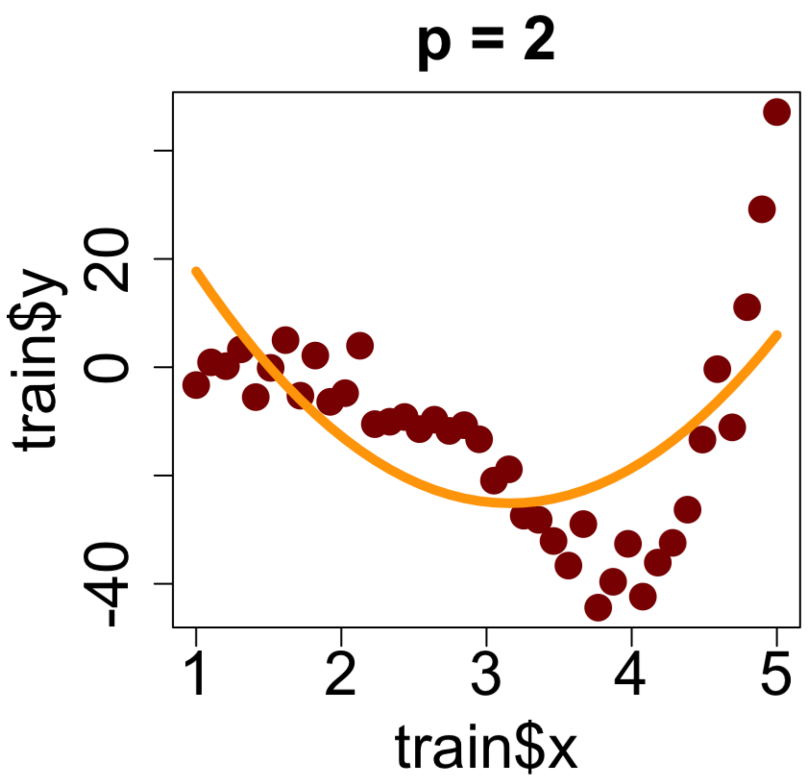
Bias–Variance Trade-off

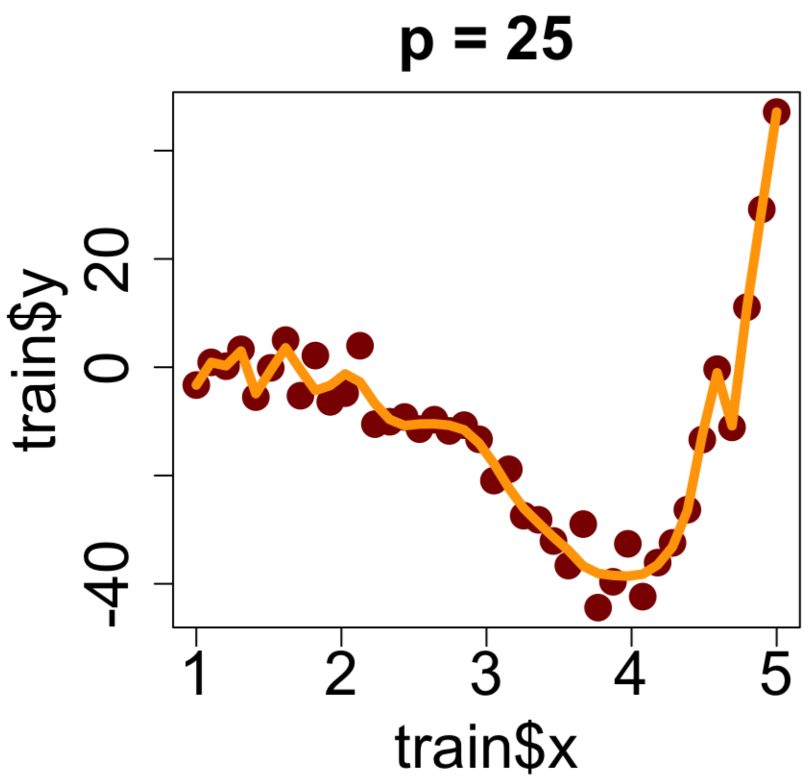
- Average prediction error = bias^2 + variance
- **Bias** term represents how far the average estimated regression function is from the true regression function across training sets
- **Variance** term represents how variable the estimated regression function is across training sets

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

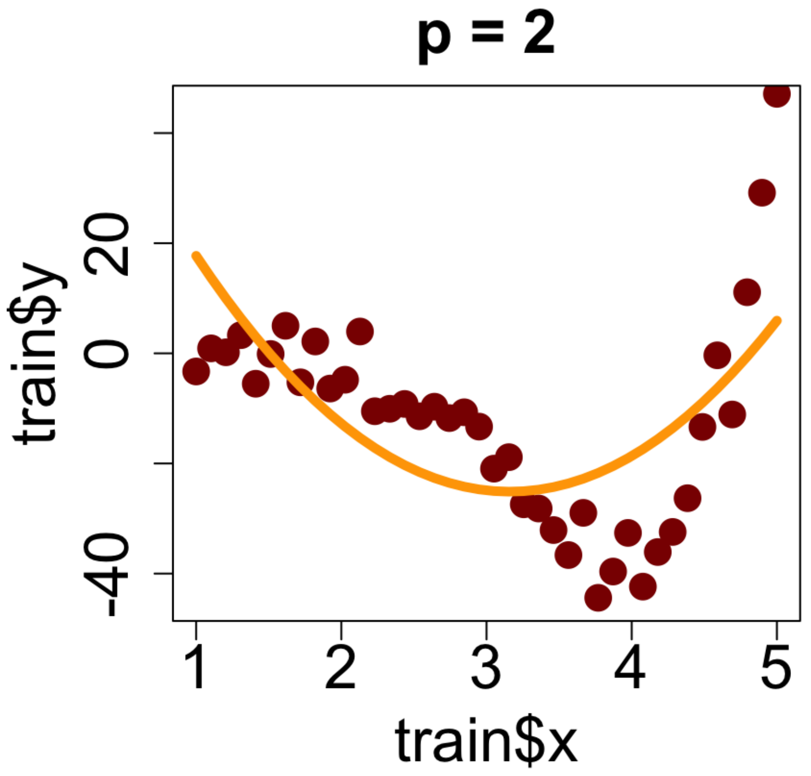


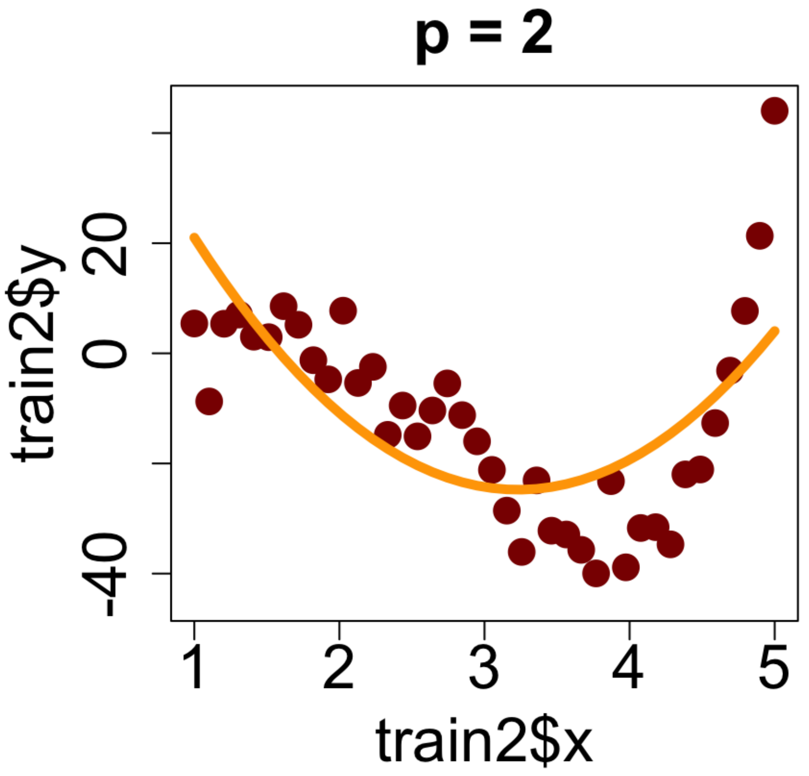


Which fitted curve will change more if the training set changes?

Bias–Variance Trade-off

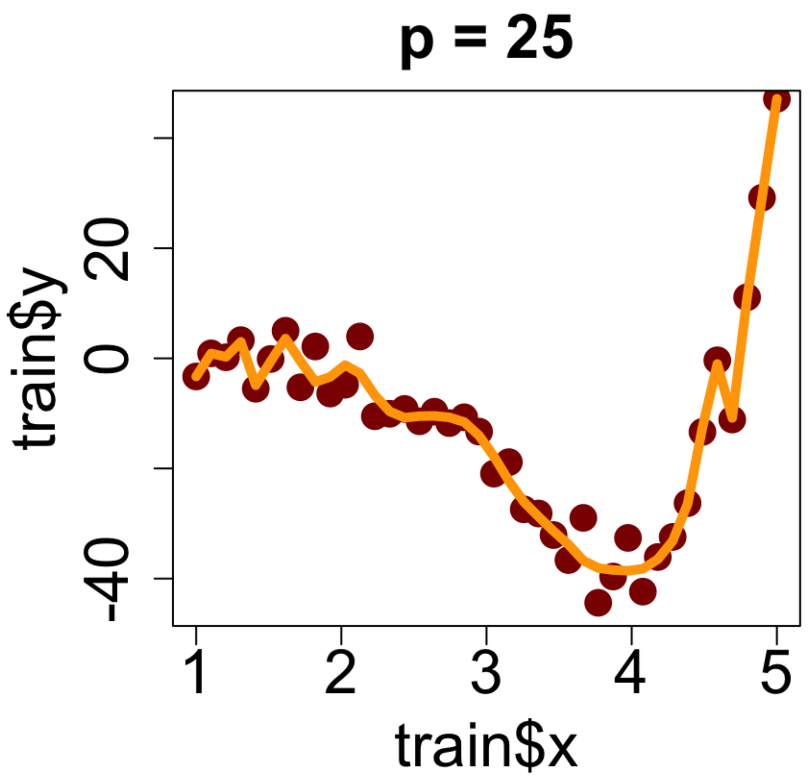
Two different training sets

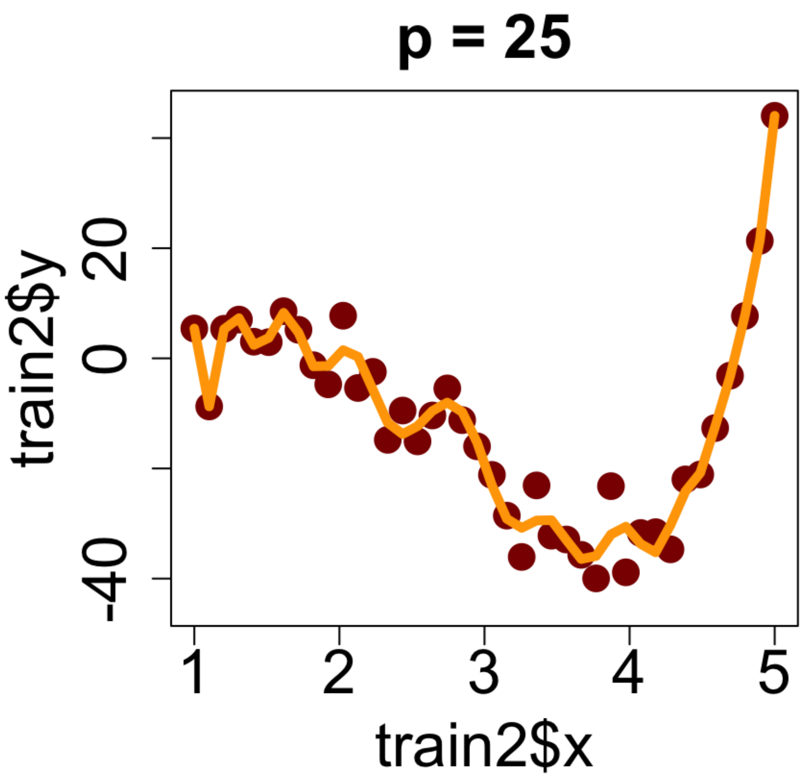




Bias–Variance Trade-off

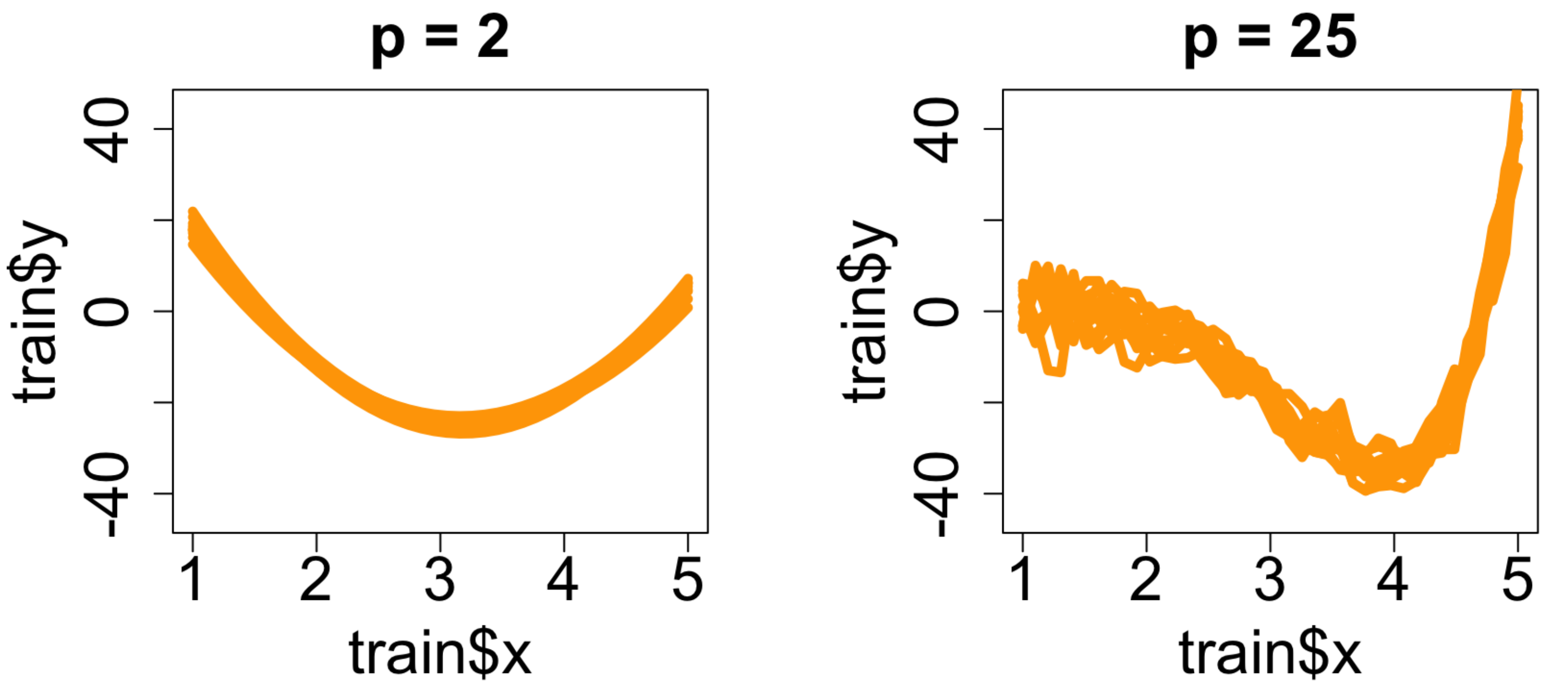
Two different training sets





Bias–Variance Trade-off

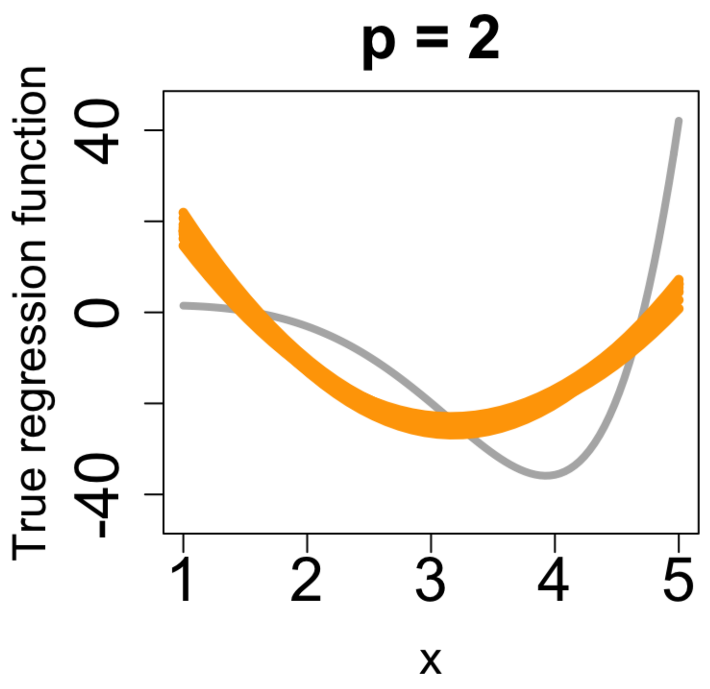
10 different training sets

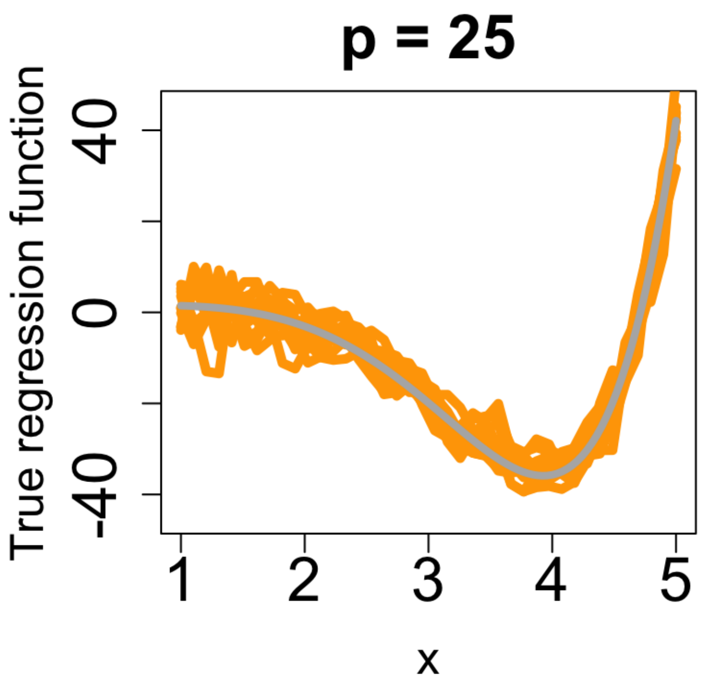


- Simple model $Y = \beta_0 + \beta_1 X + \beta_2 X^2 + \epsilon$ exhibits **low variance**
- Complex model $Y = \beta_0 + \beta_1 X + \ldots + \beta_{25} X^{25} + \epsilon$ exhibits **high variance**

Bias–Variance Trade-off

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

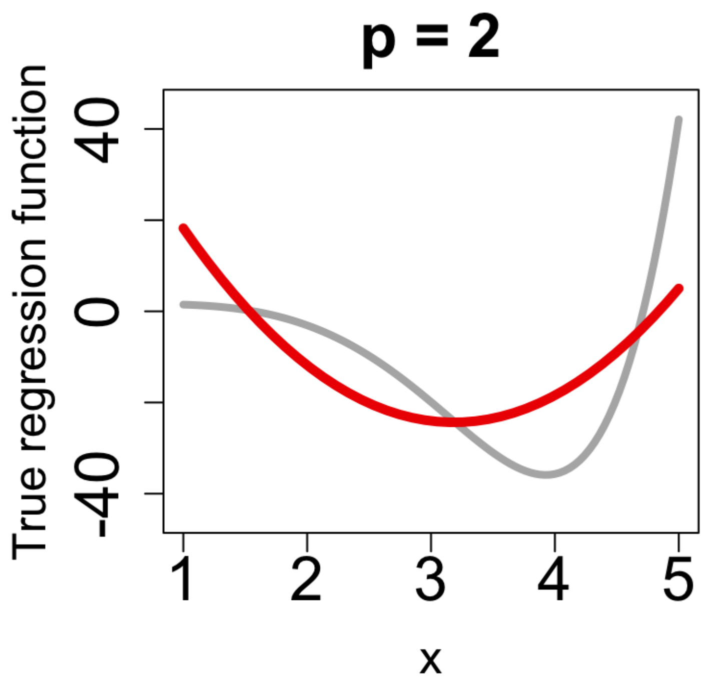


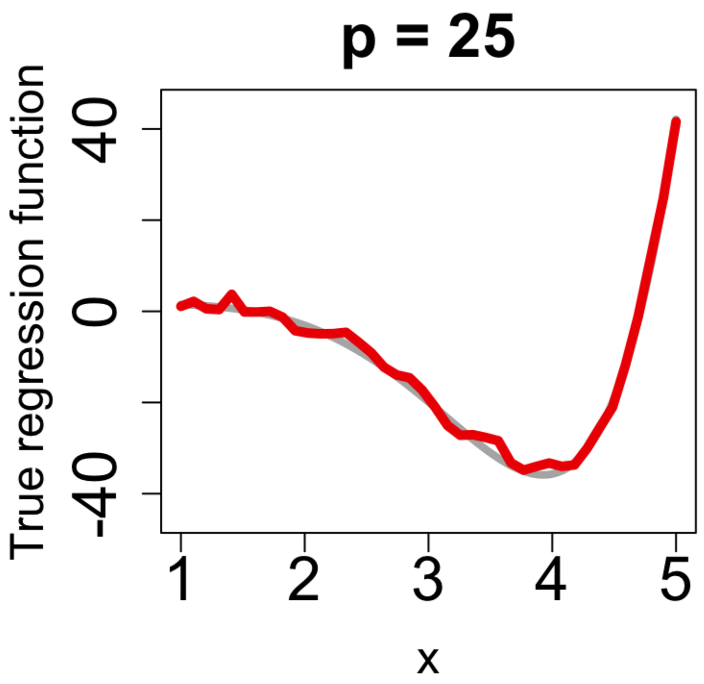


Which model approximates the true regression line more closely?

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