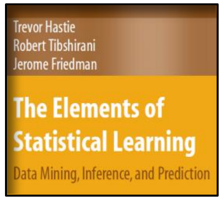
Linear regression, cross-validation, and the bias-variance trade-off

Machine Learning for Process Engineers Workshop

Stellenbosch University

March 2022



What is machine learning?

"...the [machine learning] model... is learnt based on the available training data. This is accomplished by using a learning algorithm which is capable of automatically adjusting the settings, or parameters, of the model to agree with the data. In summary, the three cornerstones of machine learning are: (1) the data, (2) the mathematical model, and (3) the learning algorithm"

-Machine Learning: A First Course for Engineers and Scientists (in press), Lindholm, Wahlström, Lindsten, Schön

"The approach taken [to machine learning] in applied mathematics and statistics has been from the perspective of function approximation and estimation."

"As statisticians, our exposition will naturally reflect our backgrounds and areas of expertise. However in the past eight years we have been attending conferences in neural networks, data mining and machine learning, and our thinking has been heavily influenced by these exciting fields. This influence is evident in our current research, and in this book."

-Elements of Statistical Learning (2nd ed. 2009), Hastie, Tibshirani, Friedman

Machine learning as function approximation

p 37

- Typical assumption:

$$y_i = f(\mathbf{x}_i) + \varepsilon_i$$

A measurement / response y_i
Depends on a set of predictors $\mathbf{x}_i$
Through some unknown function $f(\cdot)$,
But measurement is corrupted by random noise ε

Observation: Predictor-response pair $(y_i, \mathbf{x}_i)$, with $i = 1,2,3 \dots N$

Feature: Predictors may have multiple components, e.g. $x_{i,1}, x_{i,2}, \dots x_{i,p}$
these individual components are often called "features"

- Find an estimate $\hat{y}(\mathbf{x})$ that matches the data y
- What does "match" mean?

Machine learning as function approximation

p 40

- Typical assumption:

$$y_i = f(\mathbf{x}_i) + \varepsilon_i$$

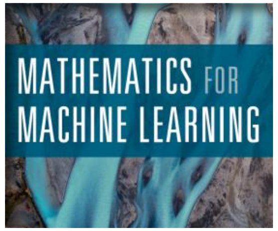
- Find an estimate $\hat{y}(\mathbf{x})$ that matches the data y
- What does “match” mean? **Measured by a loss function** $L\big(y_i, \hat{y}(\mathbf{x}_i)\big)$
- Common loss function is the squared error:

$$L\big(y_i, \hat{y}(\mathbf{x}_i)\big) = \big(y_i - \hat{y}(\mathbf{x}_i)\big)^2$$

- Why?

Machine learning as function approximation

p 42



p 292

- Typical assumption (in this example, x_i is a scalar, i.e. $p = 1$):

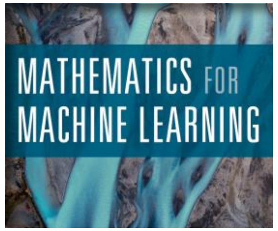
$$y_i = f(x_i) + \varepsilon_i$$

Assume $\varepsilon \sim \mathcal{N}(0, \sigma_n)$ → error is normally distributed with mean 0 and variance σ^2 and that the errors ε_i are independent

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p(y_i|x_i) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{y_i - f(x_i)}{\sigma}\right)^2\right)$$

Machine learning as function approximation

p 42



p 292

- Typical assumption (in this example, x_i is a scalar, i.e. $p = 1$):

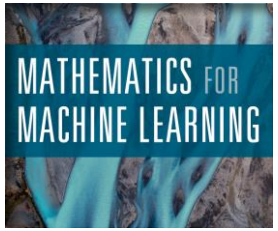
$$y_i = f(x_i) + \varepsilon_i$$

Assume $\varepsilon \sim \mathcal{N}(0, \sigma_n)$ → error is normally distributed with mean 0 and variance σ^2 and that <u>the errors ε_i are independent</u>

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p(y_i|x_i) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{y_i - f(x_i)}{\sigma}\right)^2\right)$$

Machine learning as function approximation

p 42



p 292

- Typical assumption (in this example, x_i is a scalar, i.e. $p = 1$):

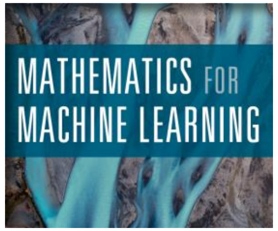
$$y_i = f(x_i) + \varepsilon_i$$

Assume $\varepsilon \sim \mathcal{N}(0, \sigma_n) \rightarrow$ <u>error is normally distributed with mean 0 and variance σ^2</u> and that the errors $\underline{\varepsilon_i}$ are independent

$$p(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^{N} p(y_i|x_i) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\left(\frac{y_i - f(x_i)}{\sigma}\right)^2\right)$$

Machine learning as function approximation

p 42



p 292

- Typical assumption (in this example, x_i is a scalar, i.e. $p = 1$):

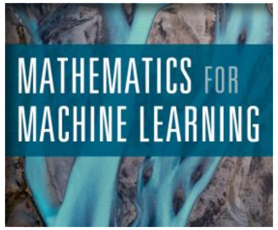
$$y_i = f(x_i) + \varepsilon_i$$

Assume $\varepsilon \sim \mathcal{N}(0, \sigma_n)$ → error is normally distributed with mean 0 and variance σ^2 and that the errors ε_i are independent

$$\ln p(\mathbf{y}|\mathbf{x}) = \ln \left[\prod_{i=1}^{N} p(y_i|x_i) \right] = \sum_{i=1}^{N} \ln p(y_i|x_i)$$

Machine learning as function approximation

p 42



p 292

- Typical assumption (in this example, x_i is a scalar, i.e. $p = 1$):

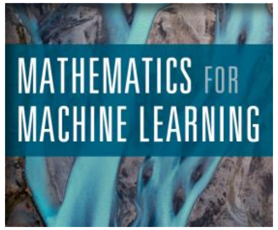
$$y_i = f(x_i) + \varepsilon_i$$

Assume $\varepsilon \sim \mathcal{N}(0, \sigma_n)$ → error is normally distributed with mean 0 and variance σ^2 and that the errors ε_i are independent

$$\ln p(\mathbf{y}|\mathbf{x}) = -N \ln\left[\sigma\sqrt{2\pi}\right] - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \left(y_i - f(x_i)\right)^2$$

Machine learning as function approximation

p 42



p 292

- Typical assumption (in this example, x_i is a scalar, i.e. $p = 1$):

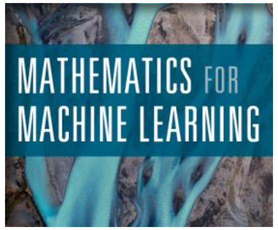
$$y_i = f(x_i) + \varepsilon_i$$

Assume $\varepsilon \sim \mathcal{N}(0, \sigma_n) \rightarrow$ error is normally distributed with mean 0 and variance σ^2 and that the errors ε_i are independent

$$\ln p(\mathbf{y}|\mathbf{x}) = -N \ln\left[\sigma\sqrt{2\pi}\right] - \frac{1}{2\sigma^2}\sum_{i=1}^{N}\left(y_i - f(x_i)\right)^2$$

Machine learning as function approximation

p 42



p 292

- Typical assumption (in this example, x_i is a scalar, i.e. $p = 1$):

$$y_i = f(x_i) + \varepsilon_i$$

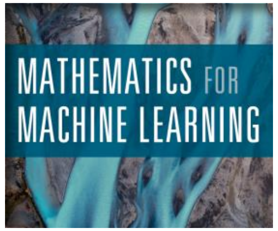
Assume $\varepsilon \sim \mathcal{N}(0, \sigma_n)$ → error is normally distributed with mean 0 and variance σ^2 and that the errors ε_i are independent

$$\ln p(\mathbf{y}|\mathbf{x}) = -N \ln\left[\sigma\sqrt{2\pi}\right] - \frac{1}{2\sigma^2} \sum_{i=1}^{N} \left(y_i - f(x_i)\right)^2$$

Minimizing the sum of squared errors is equivalent to maximizing the log-likelihood
(under certain assumptions)

Machine learning as function approximation

p 42



p 292

- Typical assumption:

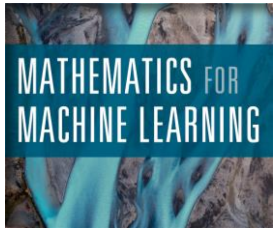
$$y_i = f(\mathbf{x}_i) + \varepsilon_i$$

- Start with squared error loss function $L\big(y, \hat{y}(\mathbf{x})\big) = \big(y - \hat{y}(\mathbf{x})\big)^2$
- Start with a linear function:

$$\hat{y}(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} = \sum_{j=0}^{p} x_{i,j}\beta_j = x_i^T \beta$$

Machine learning as function approximation

p 42



p 292

- Typical assumption:

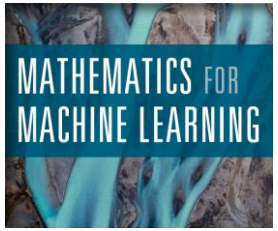
$$y_i = f(\mathbf{x}_i) + \varepsilon_i$$

- Start with squared error loss function $L\big(y, \hat{y}(\mathbf{x})\big) = \big(y - \hat{y}(\mathbf{x})\big)^2$
- Start with a linear function:

$$\hat{y}(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} = \sum_{j=0}^{p} x_{i,j}\beta_j = \mathbf{x}_i \cdot \boldsymbol{\beta}$$

Machine learning as function approximation

p 42



p 292

- Typical assumption:

$$y_i = f(\mathbf{x}_i) + \varepsilon_i$$

- Start with squared error loss function $L\big(y, \hat{y}(\mathbf{x})\big) = \big(y - \hat{y}(\mathbf{x})\big)^2$
- Start with a linear function:

$$\hat{y}(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} = \sum_{j=0}^{p} x_{i,j}\beta_j = \mathbf{x}_i \cdot \boldsymbol{\beta}$$

Machine learning as function approximation

- Typical assumption:

$$y_i = f(\mathbf{x}_i) + \varepsilon_i$$

- Start with squared error loss function $L\big(y, \hat{y}(\mathbf{x})\big) = \big(y - \hat{y}(\mathbf{x})\big)^2$
- Start with a linear function:

$$\hat{y}(\mathbf{x}_i) = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} = \sum_{j=0}^{p} x_{i,j}\beta_j = \mathbf{x}_i \cdot \boldsymbol{\beta}$$

We have included the "dummy variable" x_0 *to correspond to the intercept term* β_0

$$\boxed{\mathbf{x}_i = [1\ x_1\ x_2\ \ldots x_p]}$$

In MATLAB: First steps

- Open file
  `MLforProcEng_Workshop_1.m`”
- Run the cell
  %% Initialize and create an example of the data to be generated
  - You can run a single cell by moving your cursor to the cell (e.g. by clicking in the cell).
  - Press “CTRL+ENTER” to run only the highlighted cell

```
%% Initialize and create an example of the data to be generated
clc
clear
clf

f = @(t) 6*exp(-t.^2) .* sin(t);
sig_eps = 0.2;
```

In MATLAB

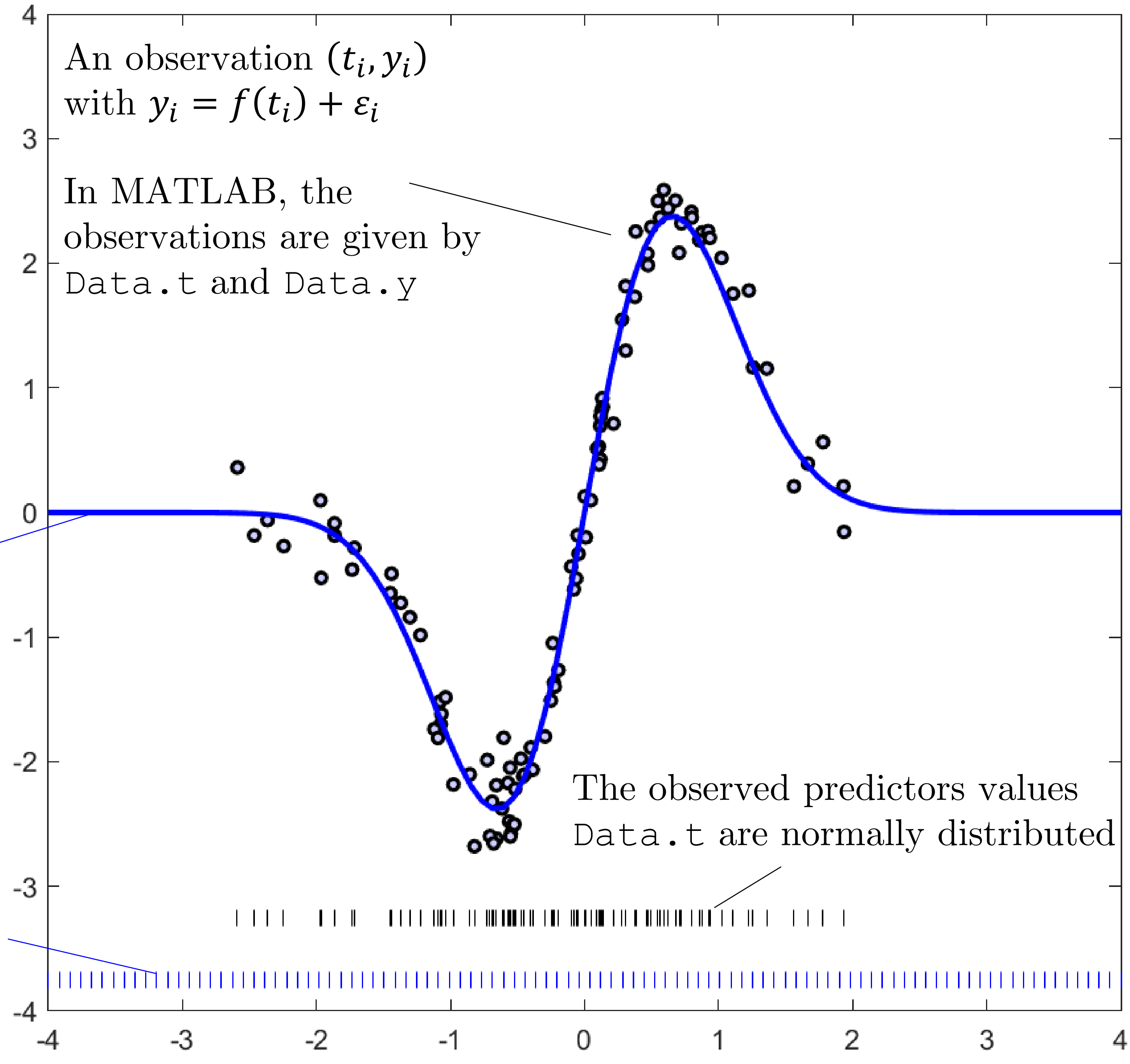
The variables `Data` and `Fit` are represented as “tables” in MATLAB. Columns in the table can be referenced using “dot indexing”, e.g. `Data.t`

Mean function

$$f(t) = 6\exp(-t^2)\sin(t)$$

In MATLAB, the mean function is evaluated at t-values given by `Fit.t` resulting in values `Fit.f`.

The t-values in `Fit.t` are equally spaced between -4 and 4, to allow us to plot a smooth curve `Fit.f`



In MATLAB: Example 1

- Go to the next cell

```
%% Example 1:  Fit a first order polynomial model
```

- Use "`fitlm`" to fit a linear model to the data
  - `mdl = fitlm(X, y)` returns a linear regression model of the responses y, fit to the data matrix X.

    Hint: in our system, "X" should be `Data.t` *and "y" should be* `Data.y`

- Use "`disp(mdl)`" to display the properties of the fitted model

- Use "`predict(mdl, Fit.t)`" to find the model predictions at the equally spaced data points "`Fit.t`"

In Python: Example 1

- Go to the next cell

  #%% Example 1: fit a first order polynomial model

- Use "`linear_model.LinearRegression()`" to fit a linear model
  - <u>`mdl`</u> `= linear_model.LinearRegression()` creates a model object,
  - `mdl.fit(X, y)` fits the data matrix X to the response y

  Hint: in our system, "X" should be `Data.t` *and "y" should be* `Data.y`

- Use "`mdl.predict(Fit.t)`" to find the model predictions at the equally spaced data points "`Fit.t`"

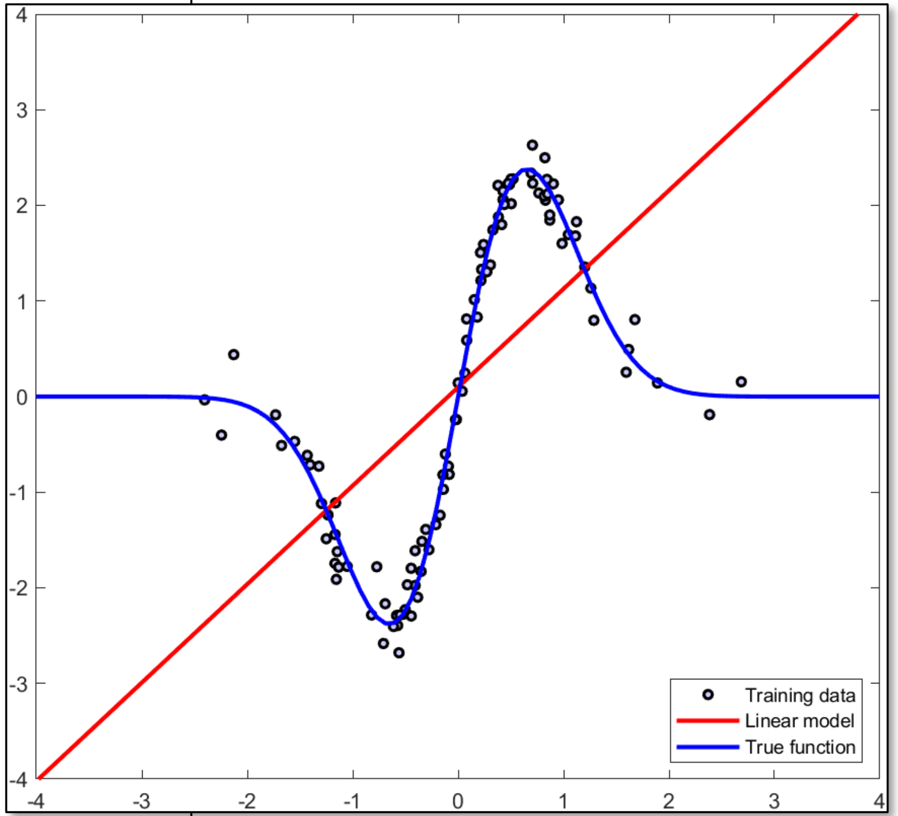
```matlab
%% Example 1:  Fit a first order polynomial model
% Generate 100 observations and plot data
Data = GenerateData(f, sig_eps, 100, …
                      true, [-4 4 -4 4], "on");


% Fit a first order polynomial model using "fitlm"
mdl = fitlm(Data.t, Data.y);


% Display your fitted model using "disp"
disp(mdl)


% Evaluated the fitted model at equally spaced points…
Fit.linear = predict(mdl, Fit.t);


% Plot the fitted linear model…
plot(Fit.t, Fit.linear, 'r', …
      Fit.t, Fit.f, 'b', 'LineWidth',2);
```



Fitting polynomials using linear regression

p 51

- In the example: one predictor variable t, one response variable y
- Fit a polynomial to data using linear regression:

$$\hat{y}(t) = \beta_0 + \beta_1 t + \beta_2 t^2 \ldots \beta_p t^p = \sum_{j=0}^{p} t^j \beta_j = [1 \;\; t \;\; t^2 \;\; \ldots t^p]\boldsymbol{\beta}$$

- Create a “design matrix” **X** with observations (in rows) and features (in columns)

$$\mathbf{X} = \begin{bmatrix} 1 & t_1 & t_1^2 & \ldots & t_1^p \\ 1 & t_2 & t_2^2 & \ldots & t_2^p \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & t_N & t_N^2 & \ldots & t_N^p \end{bmatrix}$$

In MATLAB: Example 2

- Go to the next cell, and set the value of "p" to your choice

```
%% Example 2:  Fit a p-order polynomial model
⋮
% Fit a p-th order polynomial model
p = 4;
```

- Use "`x2fx`" to create a design matrix with columns $[t \;\; t^2 \;\; t^3 \;\; \ldots t^p]$ (you can leave the dummy variable of ones)
  - `X = x2fx(Data.t, (1:p)').` converts a matrix of predictors `t` to a design matrix `X` for regression analysis. See "`doc x2fx`" for more info

- Use "`disp(mdl)`" to display the properties of the fitted model

- Use "`predict(mdl, Fit.t)`" to find the model predictions at the equally spaced data points "`Fit.t`"

In Python: Example 2

- Go to the next cell, and set the value of "p" to your choice

```
%% Example 2:  Fit a p-order polynomial model
⋮
% Fit a p-th order polynomial model
p = 4;
```

- Use "`poly = preprocessing.PolynomialFeatures(p)`" and "`X = poly.fit_transform(Data.t)`" to create a design matrix with columns $[t\ t^2\ t^3\ \ldots t^p]$

- Use the same approach to convert "`Fit.t`" to a design matrix "`X_fit`", then use "`mdl.predict(X_fit)`" to find the model predictions at the equally spaced data points "`Fit.t`"

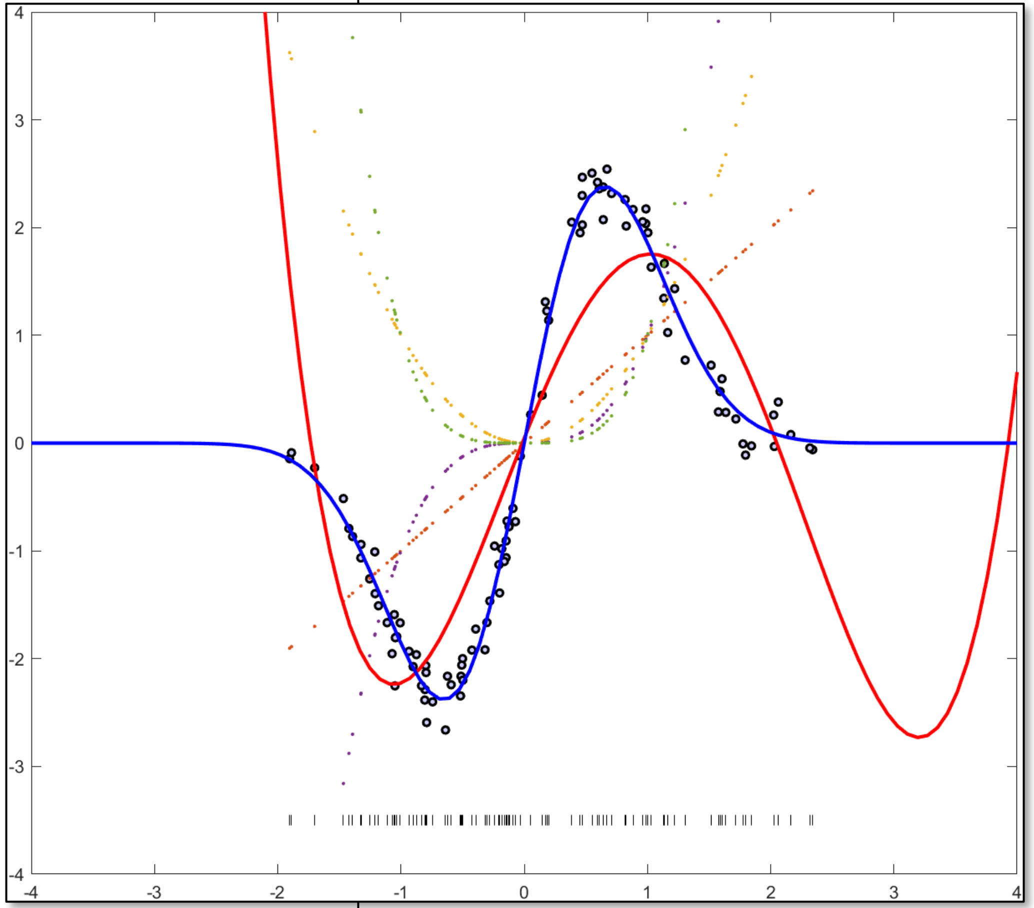
```
%% Example 2:  Fit a p-order polynomial model
clf;
Data = GenerateData(f, sig_eps, 100, true, [-4 4 -4 4], "on");

p = 4; % Fit a p-th order polynomial model

% Create a "design matrix" X…
X = x2fx(Data.t, (1:p)');
plot(Data.t, X, '.', Data.t, 0*Data.t-3.5, 'k|');

% Fit the linear model using "fitlm"…
mdl = fitlm(X, Data.y);
disp(mdl)

% Create a design matrix "X" using "x2fx" and
% the equally spaced vector "Fit.t"…
X = x2fx(Fit.t, (1:p)');
Fit.poly = predict(mdl, X);
plot(Fit.t, Fit.poly,'r', Fit.t, Fit.f, 'b', 'LineWidth',2);
```



The goal of machine learning

p 57

- So far: train models to minimize error on <u>training</u> data

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} E\big(\mathbf{y}_{train}; \hat{y}(\mathbf{X}_{train}; \boldsymbol{\beta})\big)$$

"Finding the value of [$\boldsymbol{\beta}$] which is such that the model fits the training data as well as possible is a natural idea. However... the ultimate goal of machine learning is not to fit the training data as well as possible, but rather to find a model that can generalize to new data, not used for training the model. Put differently, the problem that we are actually interested in solving is not [the equation above] but rather..."

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} E\big(\mathbf{y}_{new}; \hat{y}(\mathbf{X}_{new}; \boldsymbol{\beta})\big)$$

-Machine Learning: A First Course for Engineers and Scientists (in press), Lindholm, Wahlström, Lindsten, Schön

The goal of machine learning

p 57

- Empirical modelling: “extrapolation” is a common concern
- **Machine learning explicitly searches for models that perform well on new data**

How do we measure performance on new data?

In MATLAB: Example 3

- Go to the next cell and generate two data sets

```
%% Example 3:  Estimate the "test error" ...
⋮
Train = GenerateData(f, sig_eps, 100); % TRAINING dataset...
Test  = GenerateData(f, sig_eps, 100); % TEST dataset...
```

- Use the method from the previous example to *train* the model on the TRAINING data

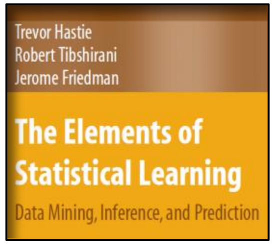
```
p = 4;
X_train = x2fx(Train.t, (1:p)');
mdl = fitlm(X_train, Train.y);
Train.y_pred = predict(mdl, X_train);
MSE_Train = mean( (Train.y - Train.y_pred).^2 )
```

Omit the semicolon to print the result on the console

- Evaluate the trained model (don't refit) on the TEST data, and compare the MSE

```
X_test = x2fx(Test.t, (1:p)');
Test.y_pred = predict(mdl, X_test);
MSE_Test = mean( (Test.y - Test.y_pred).^2 )
```

The goal of machine learning



p 219

- Datasets $(\mathbf{y}_{train}, \mathbf{X}_{train})$ and $(\mathbf{y}_{test}, \mathbf{X}_{test})$ are random variables (due to noise ε)
- Training- and testing error estimates E_{train} and E_{test} are similarly random variables
- Generate *many* training and test datasets to estimate *the expected test error*

$$\mathbb{E}\big(E_{test}(\boldsymbol{\beta})\big) = \int L\big(\mathbf{y}, \hat{y}(\mathbf{X}; \boldsymbol{\beta})\big)\, p(\mathbf{y}, \mathbf{X}) d\mathbf{y} d\mathbf{X}$$

$$\approx \frac{1}{M} \sum_{m=1}^{M} L\left(\mathbf{y}_{m,test}, \hat{y}\big(\mathbf{X}_{m,test}; \boldsymbol{\beta}_m\big)\right)$$

Where each $\big(\mathbf{y}_{m,test}, \mathbf{X}_{m,test}\big)$ represents a training data set, and $\boldsymbol{\beta}_m$ is trained using the m^{th} training data set.

In MATLAB: Example 4

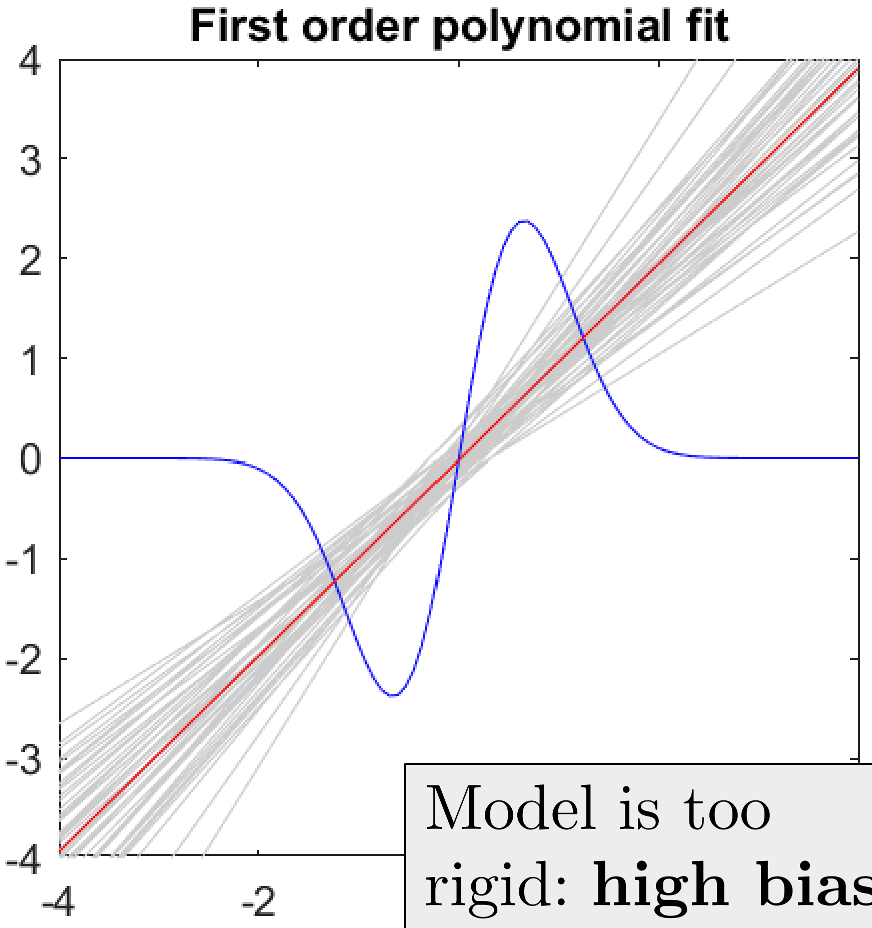
- Go to the next cell and generate two data sets

```
%% Example 4:  Estimate the "test error" …
```

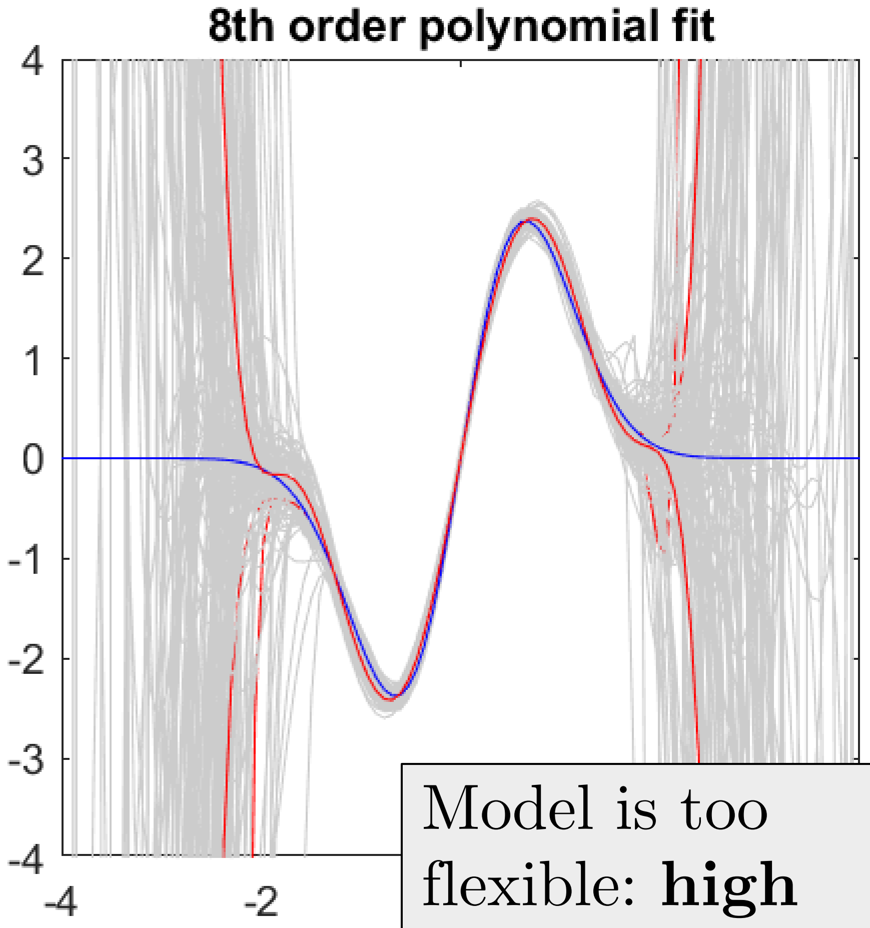
- This is a larger piece of code that I have completed beforehand – you do not need to add anything. Just press “CTRL+ENTER” to run.
  Note that this code can be a little slow…

- Please work through the code in your own time to understand the details, if you are interested (you are welcome to ask any questions)

In MATLAB: Example 4



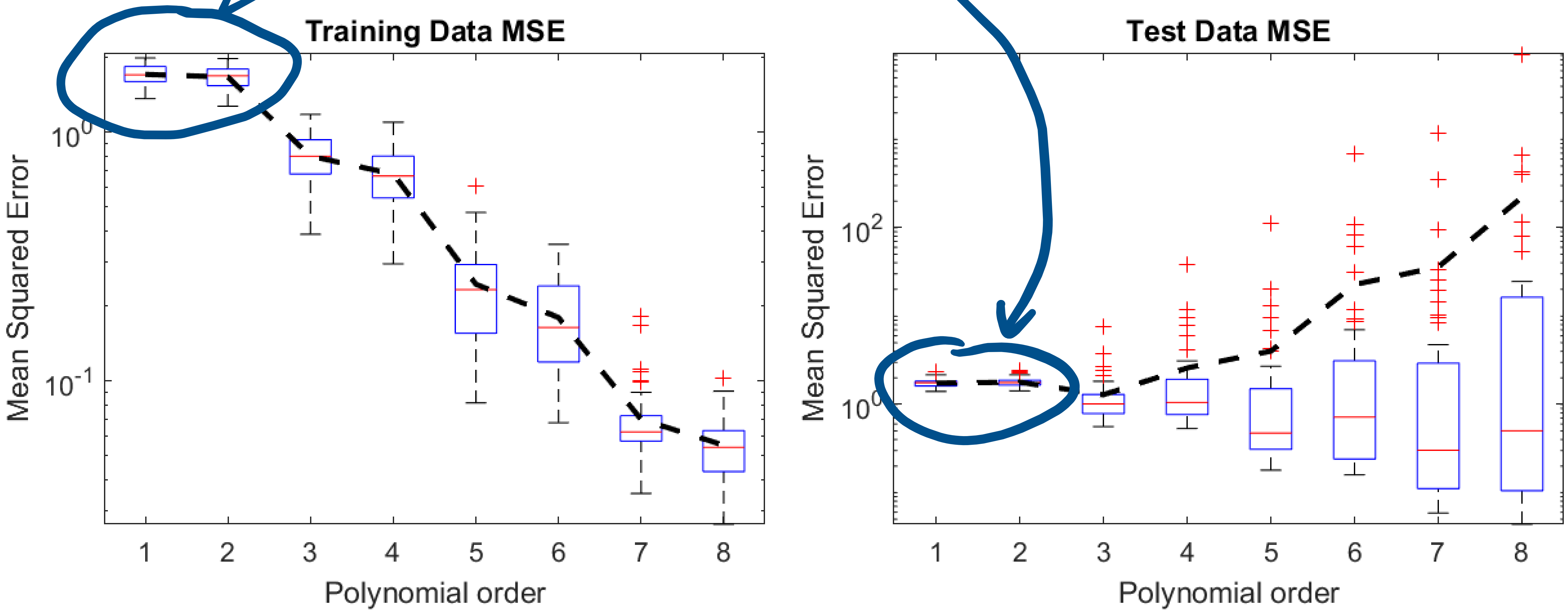
Model is too rigid: **high bias**



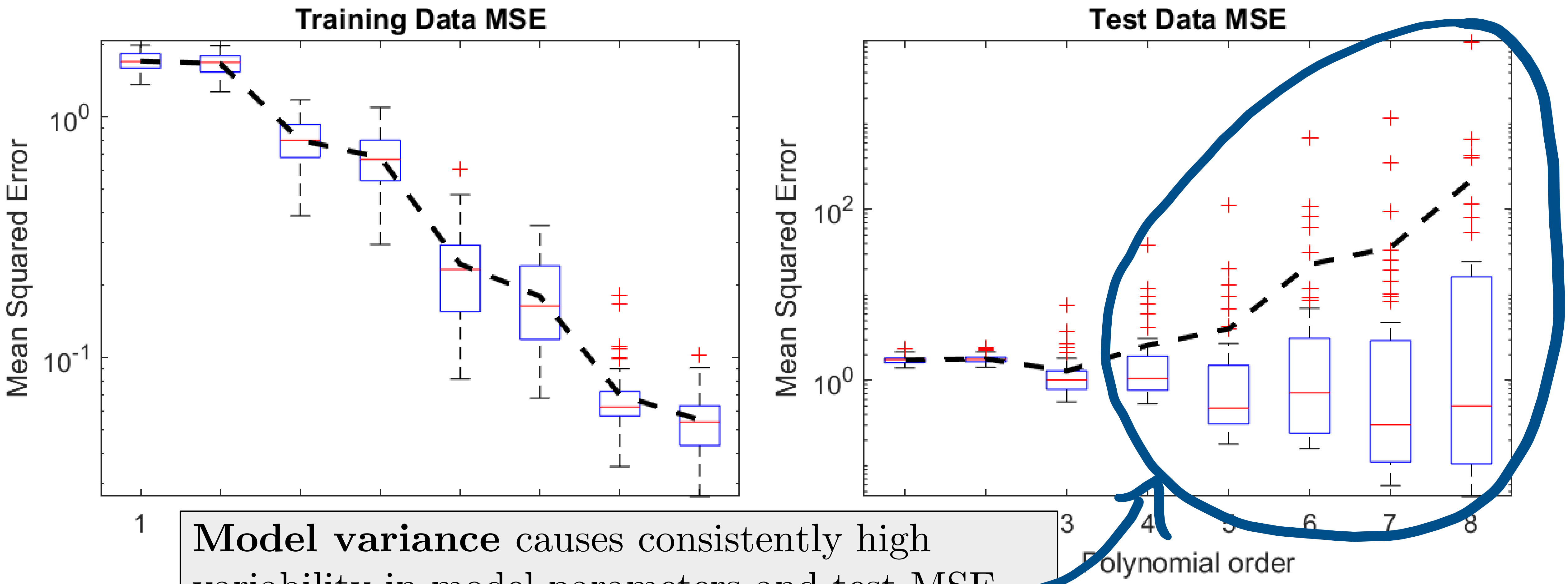
Model is too flexible: **high variance**

In MATLAB: Ex

Model bias causes consistently high MSE values (both training and test error)

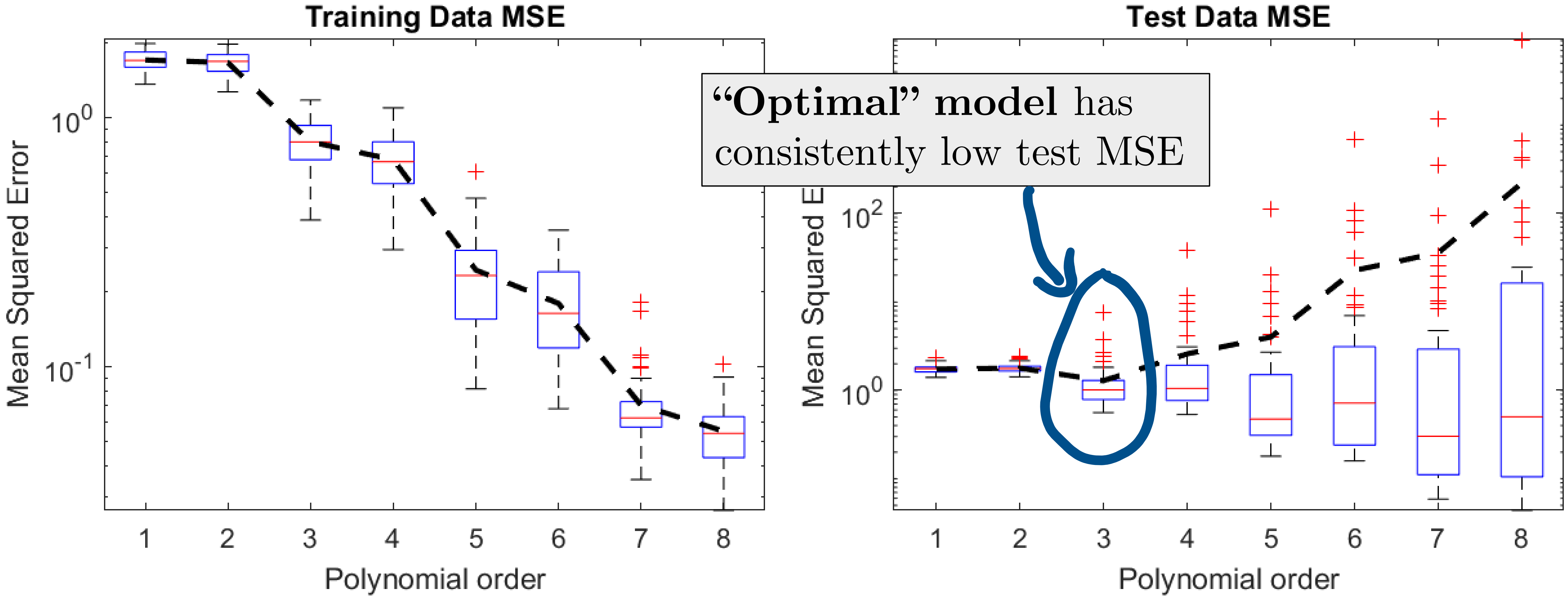


In MATLAB: Example 4



Model variance causes consistently high variability in model parameters and test MSE

In MATLAB: Example 4

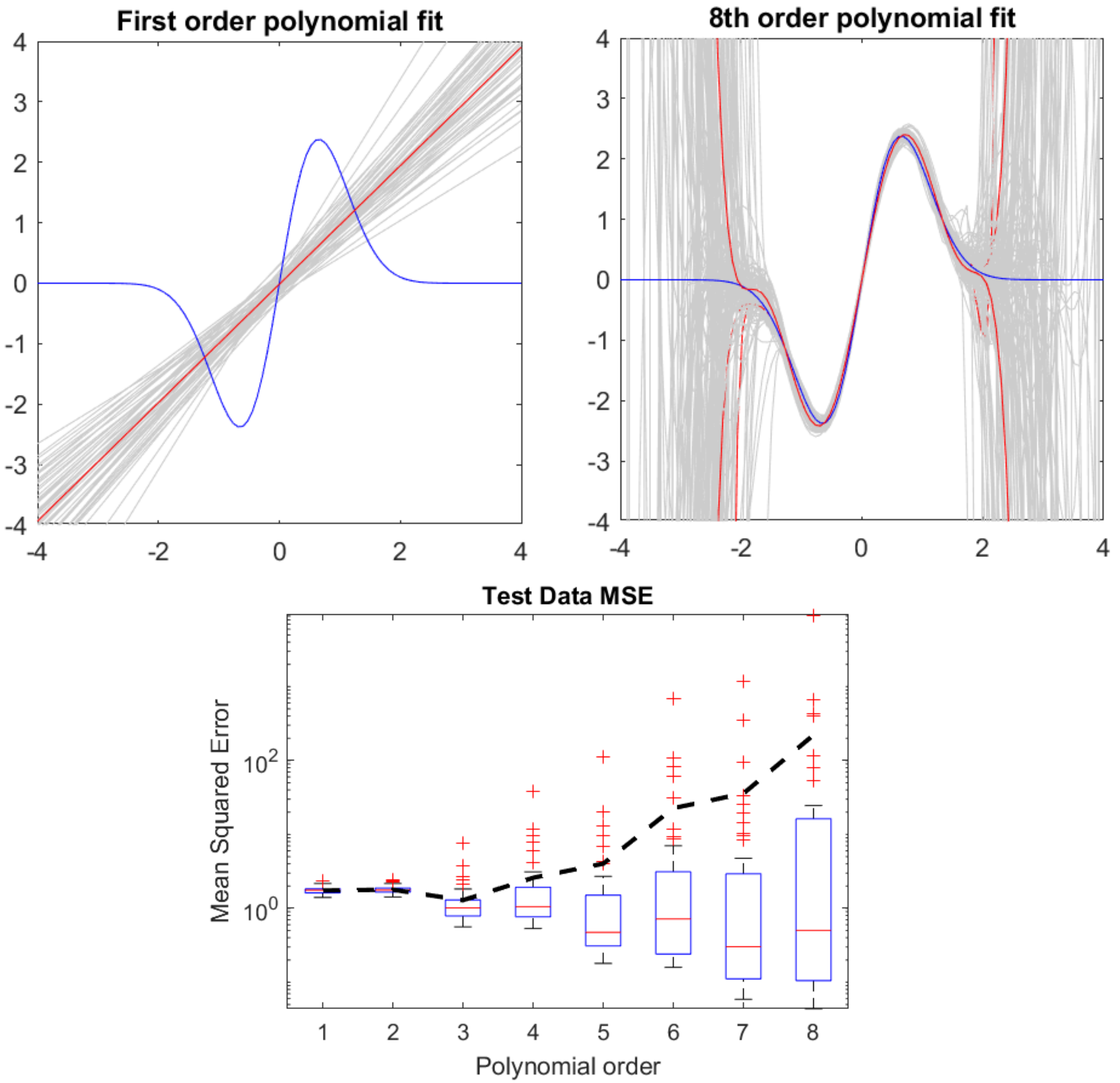


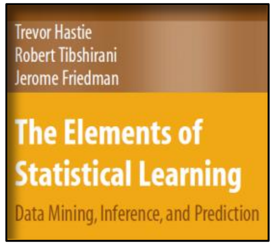
In MATLAB: Example 4

Bias-variance trade-off

Rigid models (few parameters) are biased, cannot fit the data

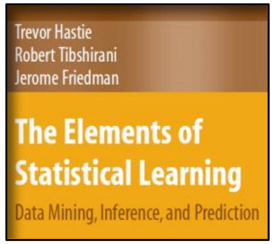
Flexible models (many parameters) fits to noise (overfits), resulting in large variance in predictions



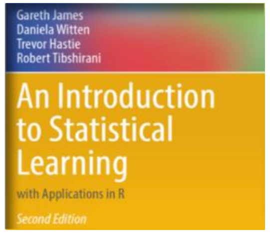


p 219

Practical estimates of the test error



p 241



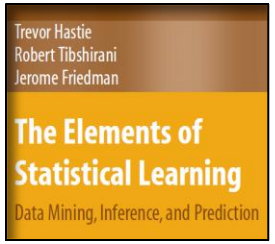
p 197

- We do not have unlimited data, nor can we easily generate multiple datasets
- "Holdout" approach: split data in two sections

Use to train model and obtain $\hat{\beta}$

Training data	Test data

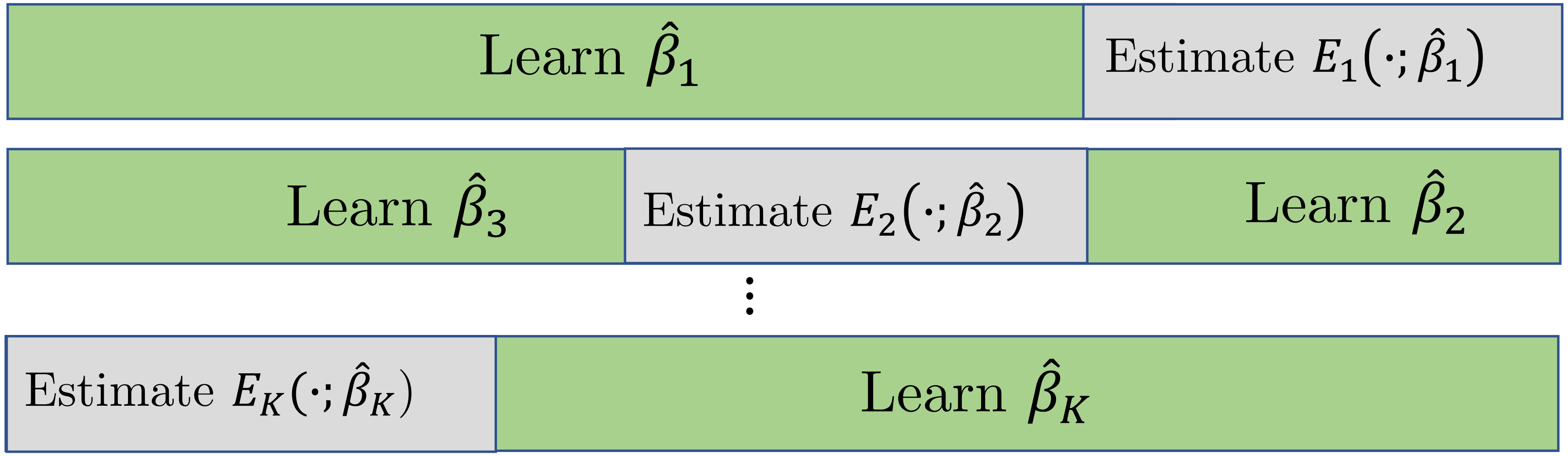
Use to estimate E_{test}



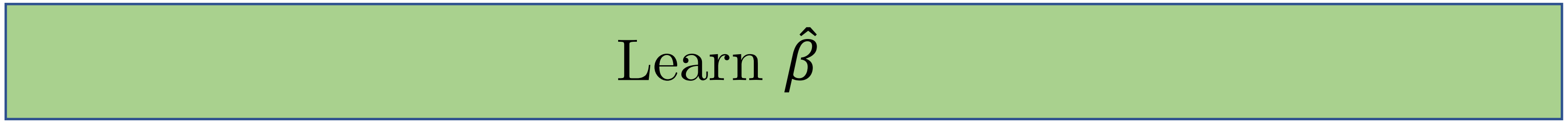
p 241

Practical estimates of the test error

- We do not have unlimited data, nor can we easily generate multiple datasets
- "Holdout" approach: split data in two sections
- "Cross-validation": split data into K "folds", train and test model K times



$$E(\cdot\,;\hat{\beta}) \approx E_{CV} = \frac{1}{K}\sum E_k$$



In MATLAB: Example 5

- Go to the next cell and generate two data sets

```
%% Example 5:  Estimate … using cross-validation
```

- This script uses the built-in function "`crossval`" to estimate the CVerror.*
  - Call the functions using the syntax "`crossval(@fun, data)`"
  - "`data`" is a table containing the full dataset, including predictors and responses
  - "`fun`" is a user defined function which accepts:
    - TRAINING data as the first input,
    - TEST data as the second input, and
    - provides an error estimate as output, e.g.:

```
function MSE = EstimateError(Train, Test)
```

  - "`@fun`" must use "`Train`" to train the model, and "`Test`" to calculate the error estimate.
  - "`crossval`" will automatically split the variable "`data`" into "`Train`" and "`Test`", K-times

Hint: in the main script, I've added the design matrix "`X`" to the table "`Data`", giving the function "EstimateError" access to the design matrix through "`Data.X`". Why?

In Python: Example 5

- Go to the next cell and generate two data sets

  #%% Example 5: Estimate "test error" for p-order polynomial model using cross-validation

- This script uses "`model_selection.cross_validate`" to estimate the CVerror.*
  - Call the functions using the syntax "`crossval(mdl, X, y)`"
  - "`mdl`" is a model object that has previously been defined and fitted on X and y
  - Additional arguments:
    - "cv = 5" will use 5-fold cross-validation
    - "scoring = 'neg_mean_squared_error' specifies how CV will score a model

In MATLAB: Example 5

```
%% Example 5:  Estimate the "test error" … using cross-validation
 ⋮
AllData = GenerateData(f, sig_eps, 100,true,[-4 4 -4 4],'on');
p = 4;
AllData.X = x2fx(AllData.t, (1:p)'); % Add design matrix to the table "Train"
Error_CV = crossval(@EstimateError, AllData);
mdl = fitlm(AllData.X, AllData.y);

X_fit = x2fx(Fit.t, (1:p)');
Fit.poly = predict(mdl, X_fit);
 ⋮


function MSE = EstimateError(Train, Test)
???
???
end
```

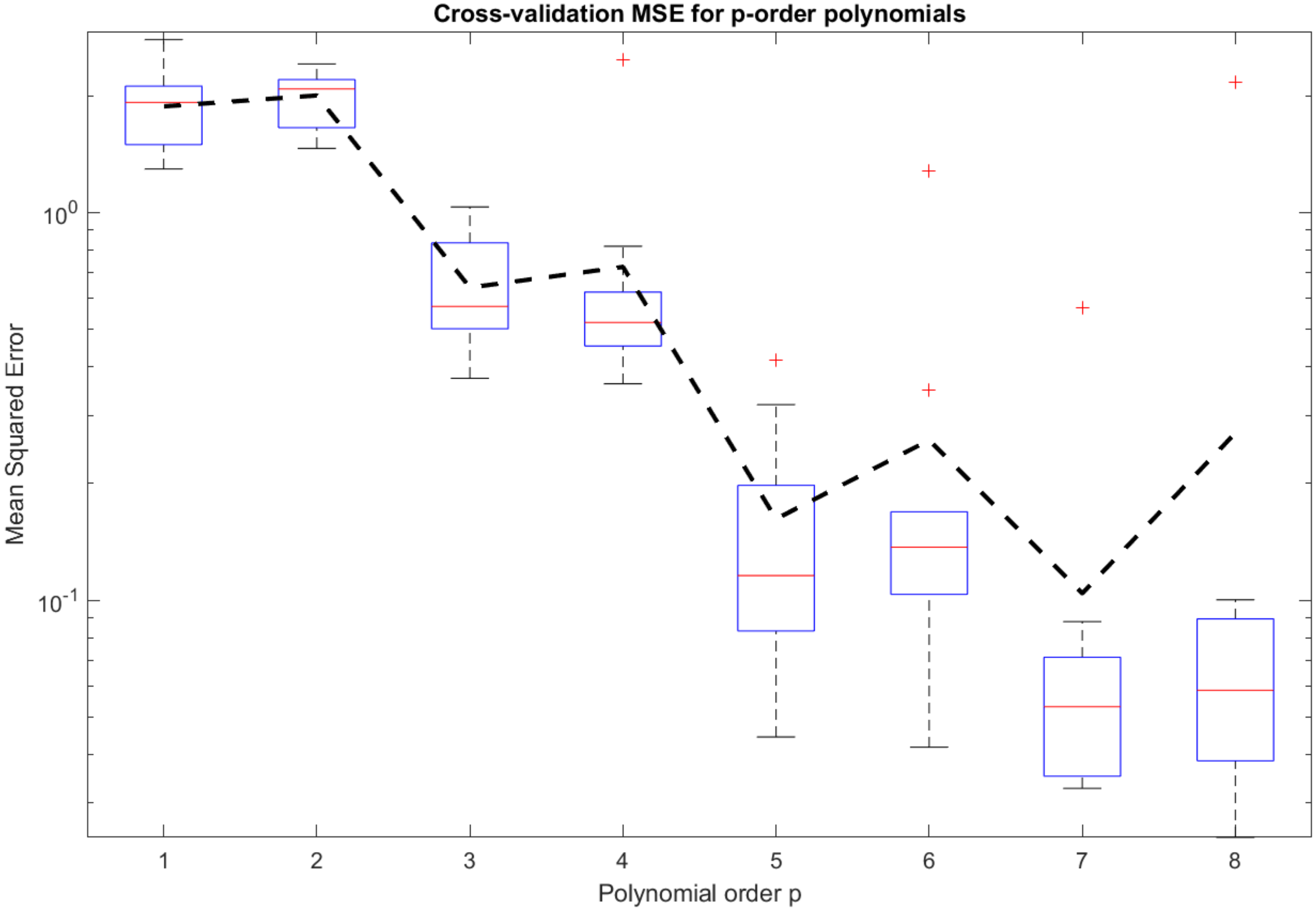
In MATLAB: Example 5

```
%% Example 5:  Estimate the "test error" … using cross-validation
 ⋮
AllData = GenerateData(f, sig_eps, 100,true,[-4 4 -4 4],'on');
p = 4;
AllData.X = x2fx(AllData.t, (1:p)'); % Add design matrix to the table "Train"
Error_CV = crossval(@EstimateError, AllData);
mdl = fitlm(AllData.X, AllData.y);

X_fit = x2fx(Fit.t, (1:p)');
Fit.poly = predict(mdl, X_fit);
⋮

function MSE = EstimateError(Train, Test)
mdl = fitlm(Train.X, Train.y);
MSE = mean( (Test.y - predict(mdl, Test.X)).^2 );
end
```

In MATLAB: Example 5



E_{CV} is dependent on the data-set and therefor a random variable.

It is only an estimate of the expected value $\mathbb{E}(E_{test})$

Recap

- The goal of machine learning is to develop a model with the lowest expected prediction error on new data, e.g.

$$\widehat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \mathbb{E}\left(E\left(\mathbf{y}_{new}, \hat{y}(\mathbf{x}_{new};\boldsymbol{\beta})\right)\right)$$

- Expected prediction error is influenced by
  - Model bias (high for rigid models), and
  - Model variance (high for flexible models)

- Expected prediction error can be *estimated* by cross-validation