# Topic 2 Model Selection continued

PROF. LINDA SELLIE

# Outline

- Motivating example:  What polynomial degree should e?
- Polynomial transformation
- Underfitting and overfitting
- Understanding error: Bias and varian
- Learning curves
- Validation and model selection
- Limited dataset:
  - Do we need a test set?
  - K-fold cross-validation
- Regularization

Yea!

Uh oh….

How to create a more complex hypothesis

Understanding what wrong

Understanding where the error comes from, and how to estimate $E_{\text{out}}[g(\mathbf{x})]$

Understanding what went wrong

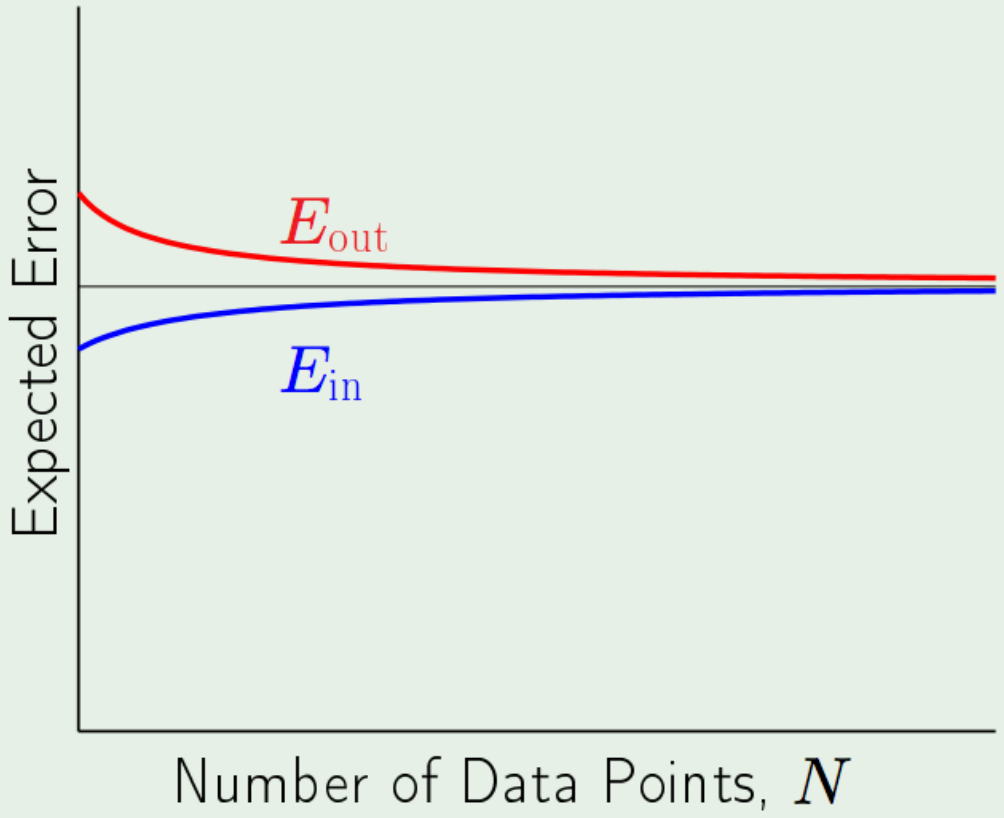
y different hypothesis classes how can we choose wisely? e estimate $E_{\text{out}}[g(\mathbf{x})]$?

# Lesson learned

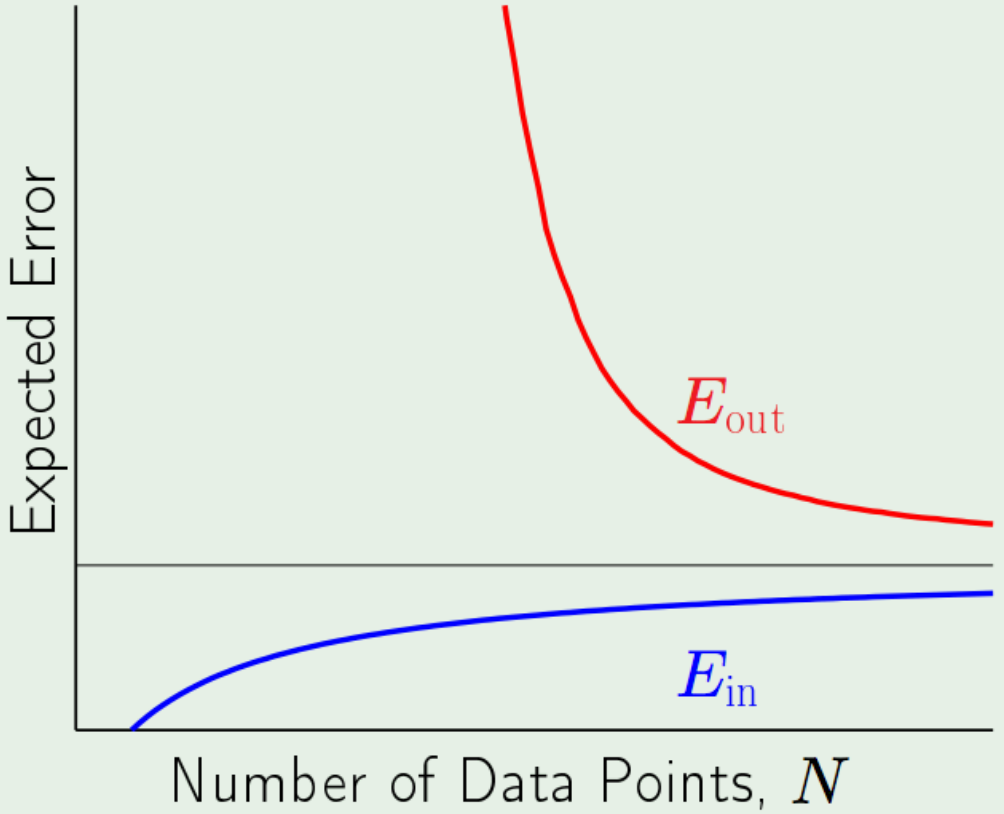
Match the '**model complexity**'

to the **data resources**, not to the **target complexity**

# The curves




Simple Model




Complex Model

Our goal is to minimize the generalization error (aka risk)
For linear regression, the goal is to minimize:

$$E_{\text{out}}(g(\mathbf{x})) = E[(y - g(\mathbf{x}))^2]$$

To do this we need to know the joint distribution of X and Y

How can we approximate this value?

Use our sample data!

…we could use our training examples to calculate our in-sample loss

$$E_{\text{in}}(g(\mathbf{x})) = \sum_{i=1}^{N} (y^{(i)} - g(\mathbf{x}^{(i)}))^2]$$

Empirical risk minimization by choosing the parameters with the highest likelihood

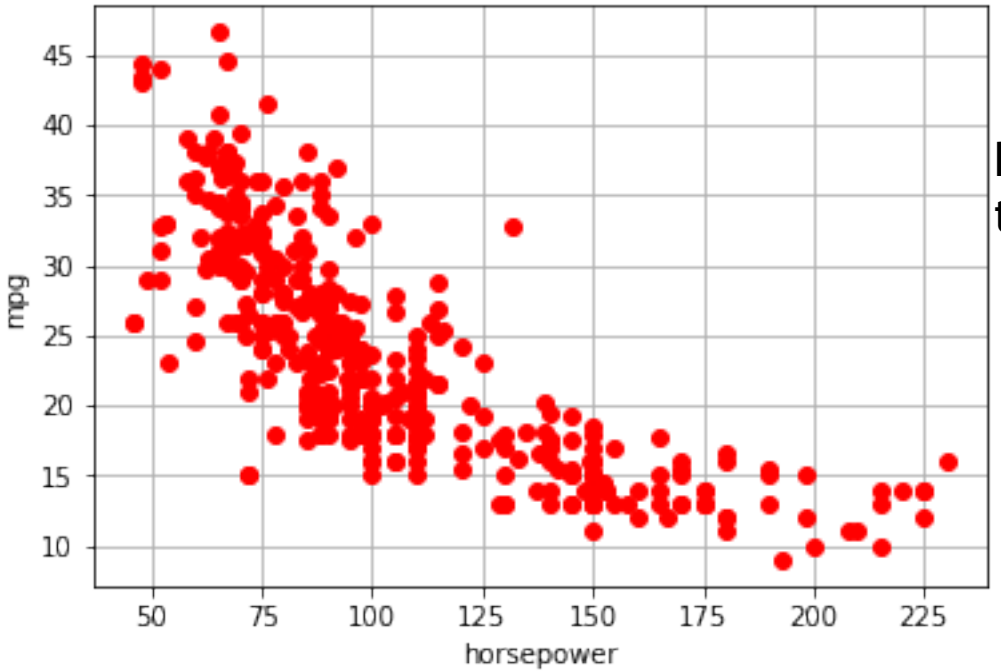
This is a very optimistic estimate!

The training error (cost) doesn't give the real world cost

$$E_{\text{out}}(g(\mathbf{x})) = E[(y - g(\mathbf{x}))^2]$$

$$E_{\text{in}}(g(\mathbf{x})) << E_{\text{out}}(g(\mathbf{x}))$$

# Data

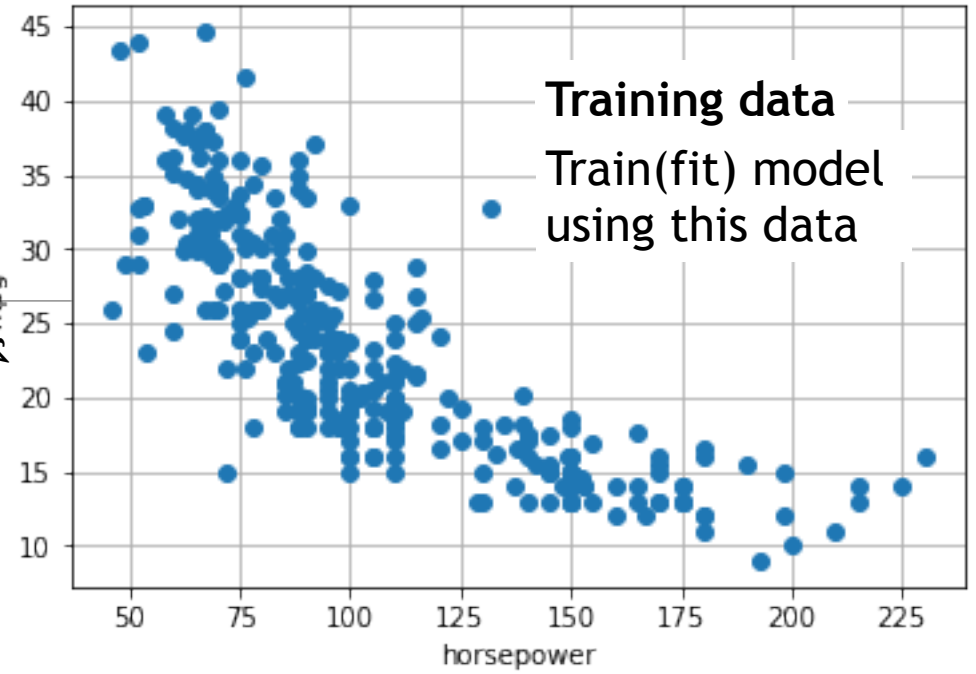



Always shuffle data before train test split

Randomly split the data

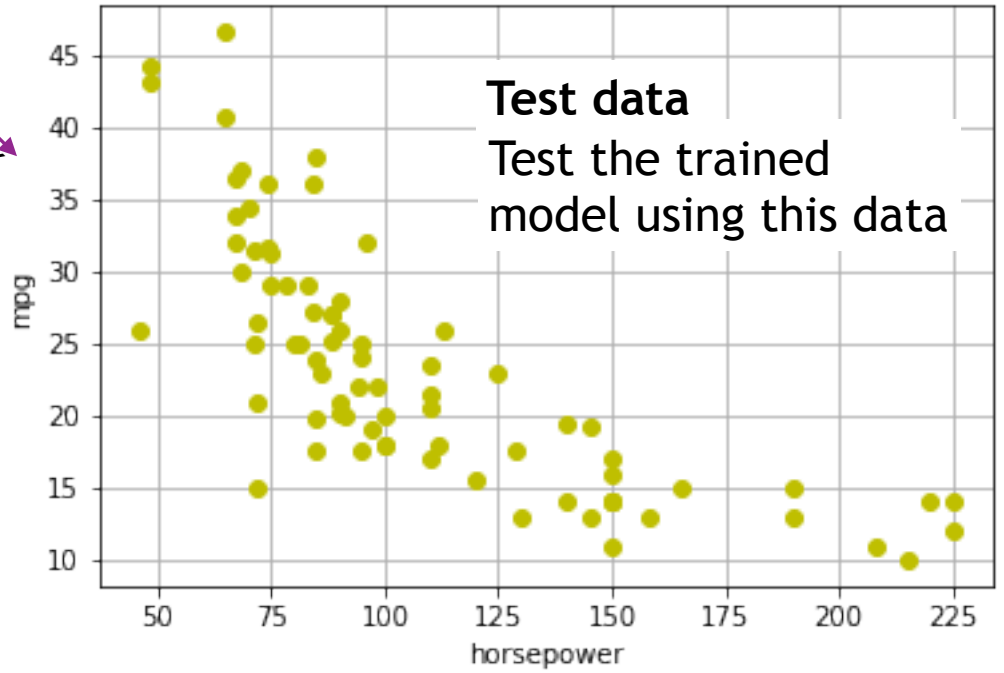
Examples K+1 through N

Examples 1 through K



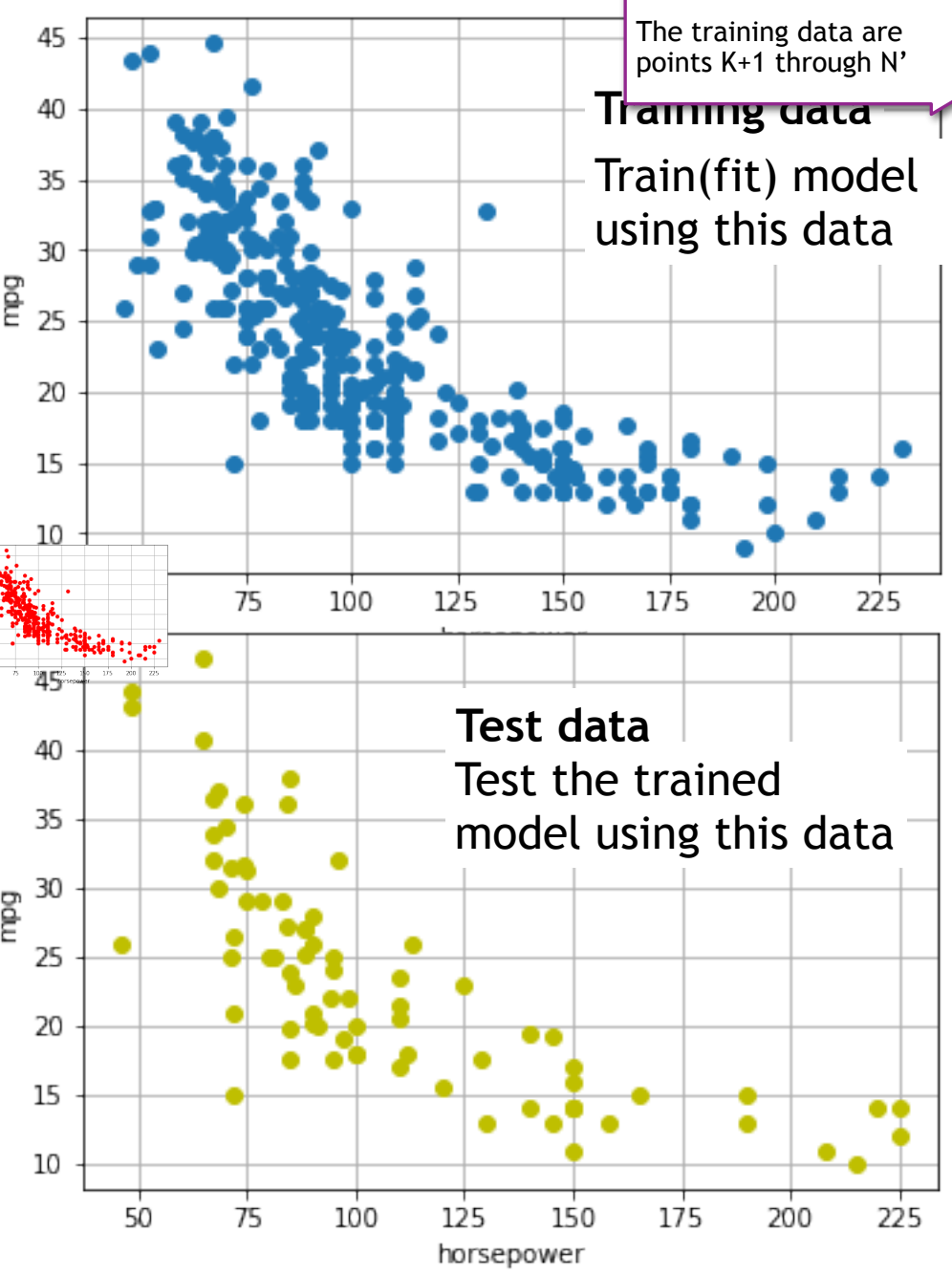

**Training data**

Train(fit) model using this data




**Test data**

Test the trained model using this data




# Fit model using the training data

Find the model that best fits **all** the training data

Determine $\hat{w}$ our estimated model parameters

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{|\text{training}|} \sum_{j \in \text{training}} \left(\text{mpg}^{(j)} - (w_0 + w_1 \text{horsepower}^{(j)})\right)^2$$

# Estimate the generalization error, $E_{\text{out}}(w)$, by using the test data

$$E_{\text{test}}(\mathbf{w}) = \frac{1}{|\text{test}|} \sum_{j \in \text{test}} \left(\text{mpg}^{(j)} - (w_0 + w_1 \text{horsepower}^{(j)})\right)^2$$

For binary classification, how good is our estimate for $E_{out}$

Is $|E_{out} - E_{test}|$ likely to be small?

“Hoeffding’s inequality is a powerful technique—perhaps the most important inequality in learning theory”

from http://cs229.stanford.edu/extra-notes/hoeffding.pdf

# Generalization Bound for classification

iid: each example "has the same probability distribution as the others and all are mutually independent."

Suppose our test set contained K randomly chosen examples

then by using Hoeffding's inequality

the probability our $E_{out}$ differs from $E_{test}$ by more than $\epsilon > 0$ occurs with probability at most $2e^{-2\epsilon^2 K}$

Example:

If K=500 and $\epsilon = 0.1$, then setting $\delta = 2e^{-2(0.1)^2(500)} = 0.0001$ then with probability $1 - \delta$ the true error is within $0.1$ of the average error on the test set.

# Generalization

Cannot get a range - instead get a confidence interval

*Hoeffding inequality* (stated without proof): for any sample size K, where each random variable is bounded in [a,b] the probability that the average value, v, of the random variables will deviate from its average $\mu$ by more than ϵ is:

$$P\left[|v-\mu| > \epsilon\right] \leq 2e^{-2\epsilon^2 K/(b-a)^2} = \delta \text{ for any } \epsilon > 0$$

Thus if $K \geq \frac{\log(2/\delta)}{2\epsilon^2}(b-a)^2$ then with probability $1-\delta$

v is $\epsilon$ close to $\mu$

We are assuming the K examples are drawn iid from a distribution

Example:

Let g be a binary classifier (g outputs 0,1), let $v$ be the average error of g on the test set of size K, and let $\mu$ be the true error of g. The probability that $|v-\mu| > \epsilon$ is at most $2e^{-2\epsilon^2 K}$

If K=500 and $\epsilon = 0.1$, then setting $\delta = 2e^{-2(0.1)^2(500)}$ then with probability $1-\delta$ the true error is within $0.1$ of the average error on the test set.

0.999909

# ...ization

Our estimated average error on our test set

Bound using numbers: K, $\epsilon$ and range of output values of function

Cannot get a range - instead get a confidence interval

*Hoeffding inequality* (stated without proof): for any sample size K, where each random variable is bounded in [a,b] the probability that the average value, v, of the random variables will deviate from its average $\mu$ by more than ϵ is:

$$P\left[|v-\mu| > \epsilon\right] \leq 2e^{-2\epsilon^2 K/(b-a)^2} = \delta \text{ for any } \epsilon > 0$$

Thus if $K \geq \frac{\log(2/\delta)}{2\epsilon^2}(b-a)^2$ then with probability $1-\delta$

v is $\epsilon$ close to $\mu$

We are assuming the K examples are drawn iid from a distribution

Example:

Let g be a binary classifier (g outputs 0,1), let $v$ be the average error of g on the test set of size K, and let $\mu$ be the true error of g. The probability that $|v-\mu| > \epsilon$ is at most $2e^{-2\epsilon^2 K}$

If K=500 and $\epsilon = 0.1$, then setting $\delta = 2e^{-2(0.1)^2(500)}$ then with probability $1-\delta$ the true error is within $0.1$ of the average error on the test set.

0.999909

# Generalization

*Hoeffding inequality* (stated without proof) for any sample size K, where each random variable is bounded in [a,b] the probability that the average value, v, of the random variables will deviate from its average $\mu$ by more that $\epsilon$ is:

$$P\left[\,|v-\mu| > \epsilon\right] \le 2e^{-2\epsilon^2 K/(b-a)^2} = \delta \text{ for any } \epsilon > 0$$

Thus if $K \ge \frac{\log(2/\delta)}{2\epsilon^2}(b-a)^2$ then with probability $1-\delta$

v is $\epsilon$ close to $\mu$

We are assuming the K examples are drawn iid from a distribution

Example:

Let g be a binary classifier (g outputs 0,1), let $v$ be the average error of g on the test set of size K, and let $\mu$ be the true error of g. The probability that $|v-\mu| > \epsilon$ is at most $2e^{-2\epsilon^2 K}$

If K=100 and $\epsilon = 0.2$, then $\delta = 2e^{-2\cdot(0.2)^2\cdot 100}$.

With probability $1-\delta$ = 0.999 our estimated test set error is within $0.2$ of the out of sample error

# Outline

- ❑ Motivating example: What polynomial degree should e?
- ❑ Polynomial transformation
- ❑ Underfitting and overfitting
- ❑ Understanding error: Bias and varian
- ❑ Learning curves
- ❑ Validation and model selection
- ❑ Limited dataset:
  - Do we need a test set?
  - K-fold cross-validation
- ❑ Regularization

Yea!

Uh oh….

How to create a more complex hypothesis

Understanding what wrong

Understanding where the error comes from, and how to estimate $E_{\text{out}}[g(\mathbf{x})]$

Understanding what went wrong

y different hypothesis classes how can we choose wisely? e estimate $E_{\text{out}}[g(\mathbf{x})]$?

# Estimating the generalization error:

One model:

- Data → Training, Test
- Data → k-fold Cross Validation

Comparing several models and/or different hyper-parameters:

- Data → Training, Validation, Test
- Data → Training, Validation
- Data → k-fold cross Validation, Test
- Data → k-fold cross Validation

How to choose the best model (aka hypothesis class)

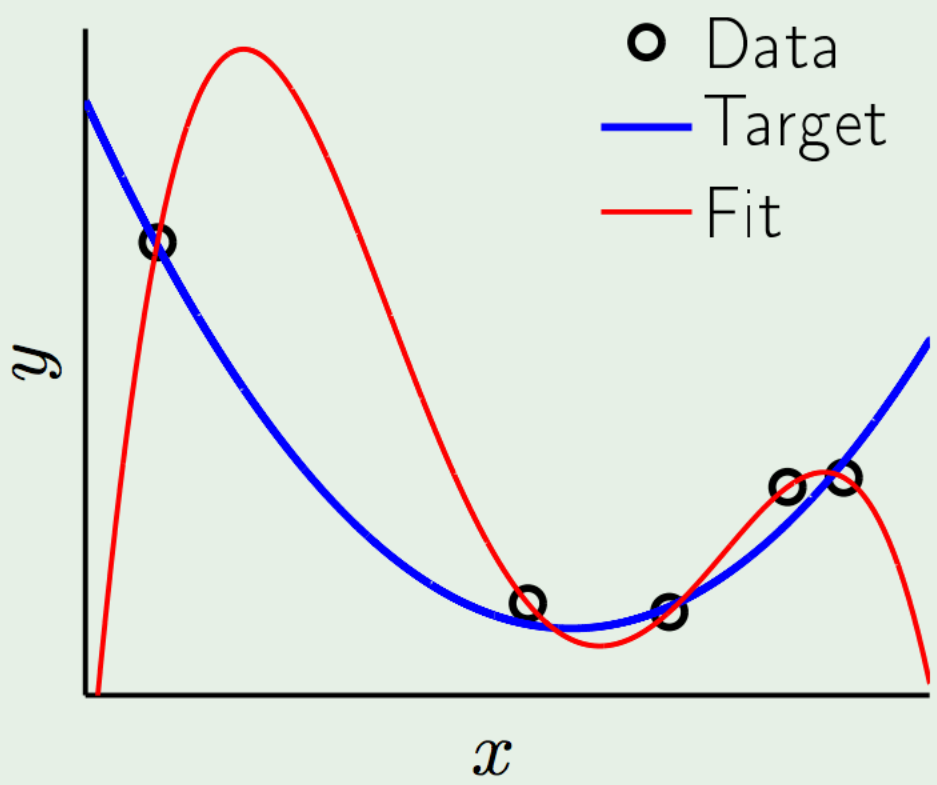
How to prevent overfitting?

# Two cures

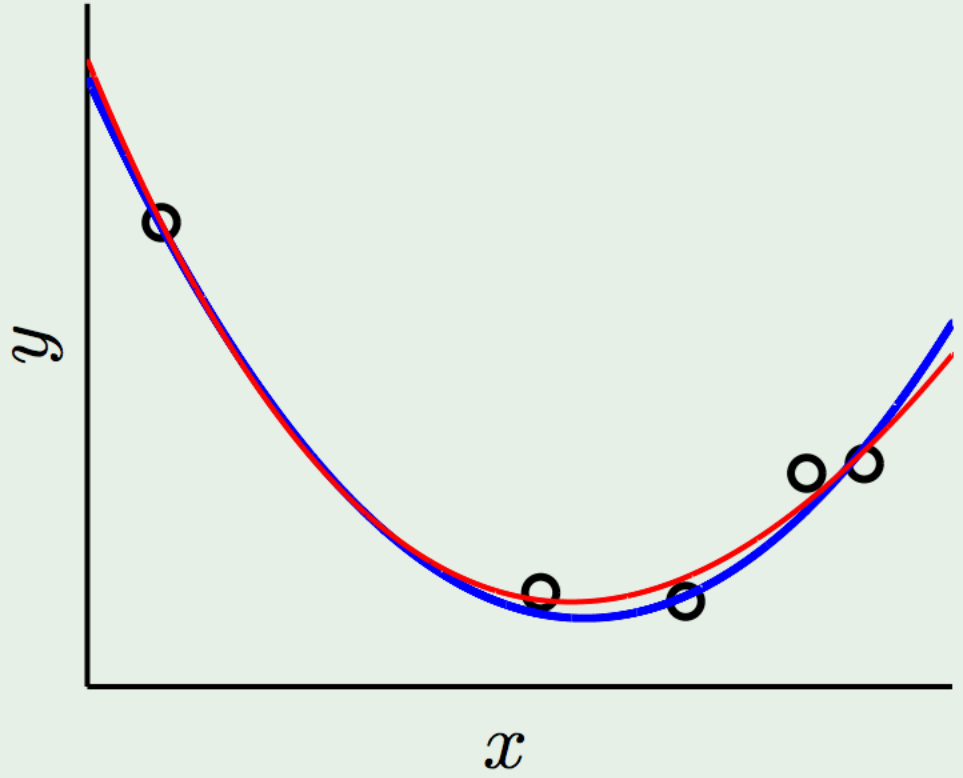
**Regularization:** Putting the brakes

**Validation:** Checking the bottom line

# Putting the brakes




free fit




restrained fit

# How do we choose the hypothesis class/model class $\{\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M\}$ to avoid overfitting or underfitting?

WE NEED TO "TUNE" THE MODEL PARAMETER

# Example Question

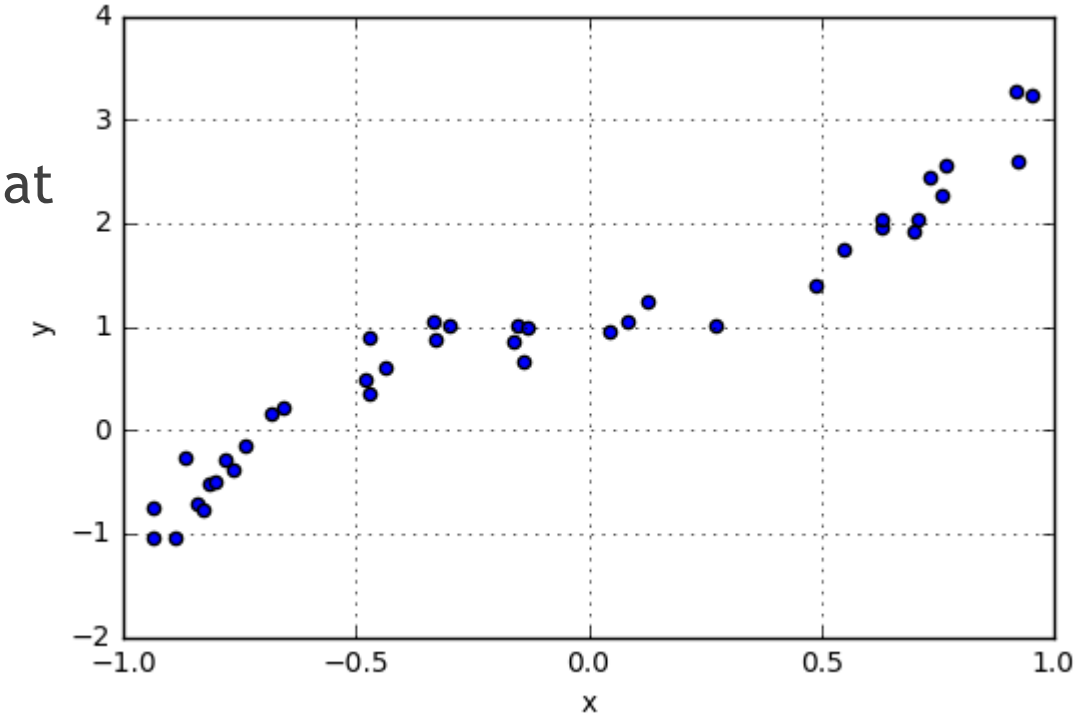
- You are given some data. The data has only one feature.
- You decide to find the **polynomial transformation** that best fits your data

$$\hat{y}^{(i)} = \tilde{\mathbf{w}}^T \Phi_d(\mathbf{x}^{(i)})$$

$$\hat{y}^{(i)} = \tilde{w}_0 + \tilde{w}_1 x^{(i)} + \tilde{w}_2 x^{(i)2} + \cdots + \tilde{w}_{\tilde{d}} x^{(i)\tilde{d}}$$

- What model order d should you use?

Thoughts?



# Using RSS on Training Data?

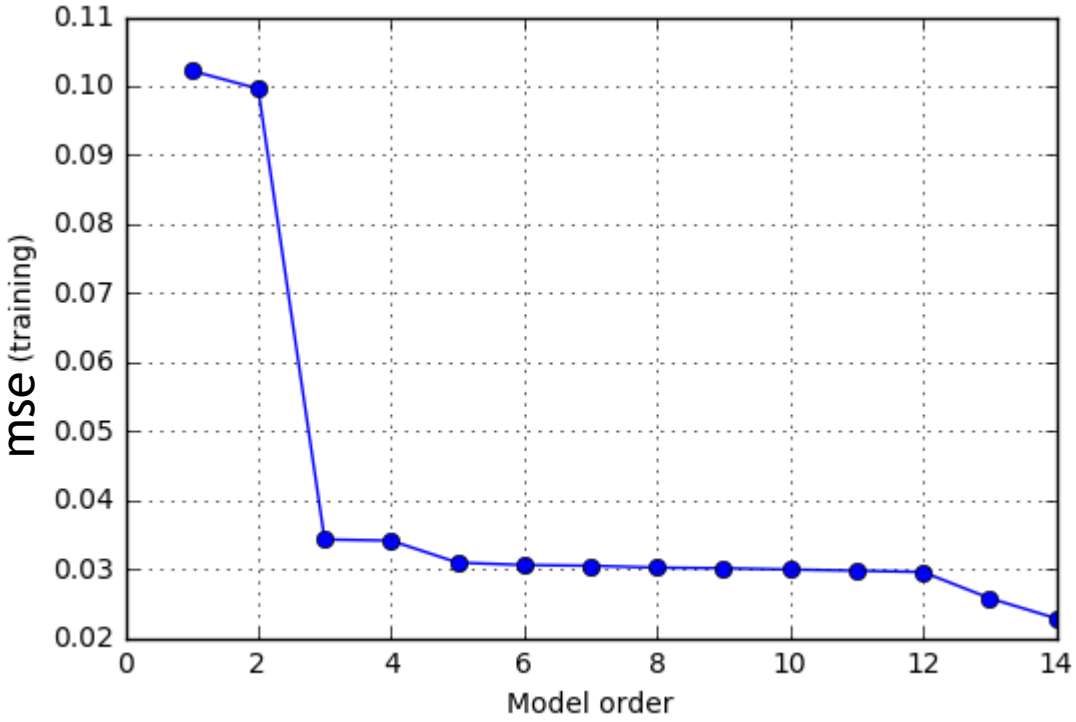
- Simple (but bad) idea:
  - For each model order, d,
    1. Compute $\tilde{\mathbf{w}}$ on transformed data, $\Phi_d(\mathbf{x})$. Predict labels on the transformed training data,

$$\hat{y}^{(i)} = \tilde{\mathbf{w}}^T \Phi_d(\mathbf{x})$$

    2. Compute MSE

$$MSE(d) = \frac{1}{N} \sum_i (y^{(i)} - \hat{y}^{(i)})^2$$

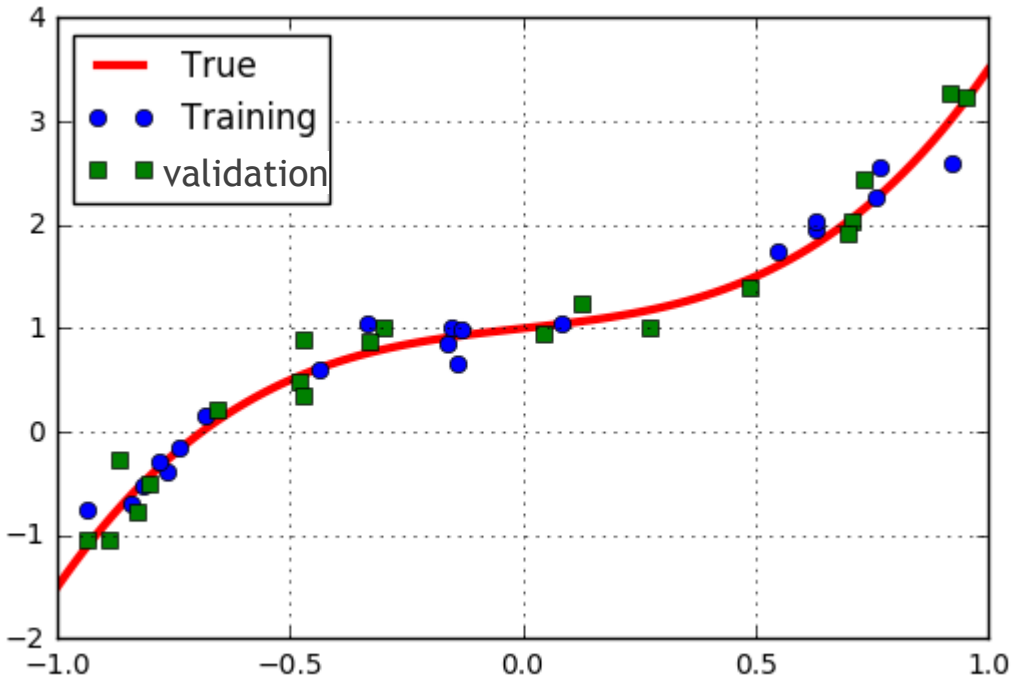
    3. Find d with lowest MSE
- This doesn't work
  - MSE(d) is always decreasing (Question: **Why**?)
  - Minimizing MSE(d) will pick d as large as possible
  - Leads to overfitting
- What went wrong?
- How can we do better?



# Polynomial Example: Training Validation Split

- Example: Split data into 20 samples for training, 20 for validation

Shuffle your data before splitting it into training, validation and test data



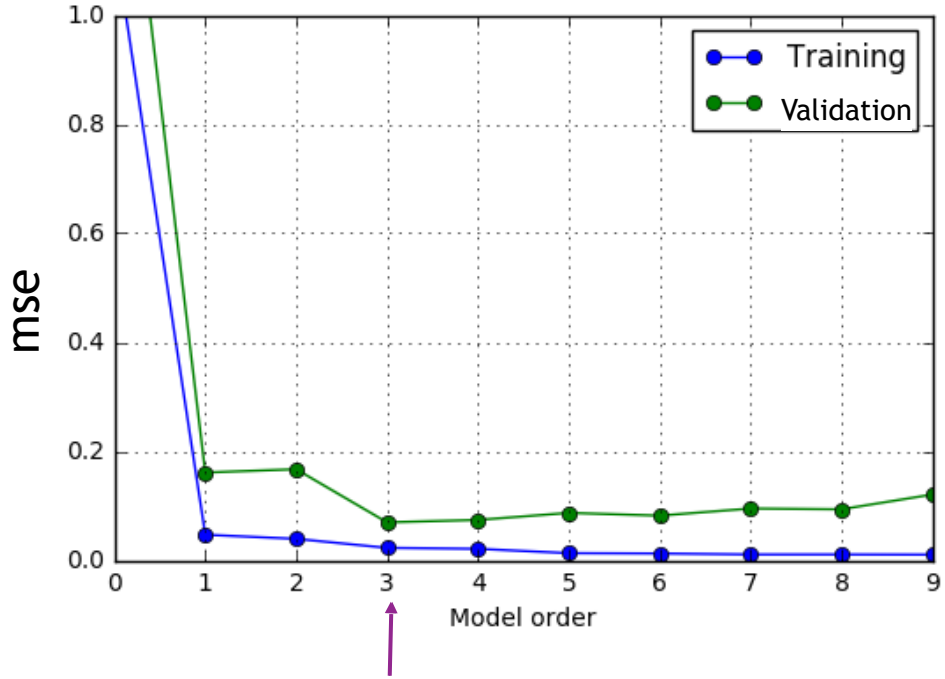

```
# Number of samples for training and Validation
ntr = nsamp // 2
nts = nsamp - ntr

# Training
xtr = xdat[:ntr]
ytr = ydat[:ntr]

# Validation
xVal = xdat[ntr:]
yVal = ydat[ntr:]
```

# Finding the Model Order

- Estimated optimal model order = 3




MSE validation minimized at 3
MSE training always decreases

# Model selection with lots of data

- Split the data into **training**, **validation**, and **test data**

Shuffle first!

- For each model (e.g., degree $d$)
  - **train** on the **training data** to find **parameters** $\mathbf{w}_d$
  - Estimate the **error** of $\mathbf{w}_d$ on the **validation data**

Each model has its own weights/parameters. We are using a subscript to distinguish the different weights/parameters for the different models

- Pick the best performing model (hypothesis) to be the model with the lowest validation error. Call the best hyperparameters $d^*$
- **Estimate out of sample error** $E_{out}$ of the best model using **test data** (e.g., $\mathbf{w}_{d*}$)
- Typical splits:


| | | |
|---|---|---|
| 10% | 10% | 80% |
| 10% | 20% | 70% |
| 25% | 25% | 50% |

https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch03.pdf

# Model selection with lots of data

- Split the data into **training**, **validation**, and **test data**

Shuffle first!

- For each model (e.g., degree $d$)
  - **train** on the **training data** to find **parameters** $\mathbf{w}_d$
  - Estimate the **error** of $\mathbf{w}_d$ on the **validation data**

Each model has its own

Use the validation data to make decisions: i.e., which hyperparameter. Choosing the right hyperparameter is more than just selecting which transformation of the data is best.

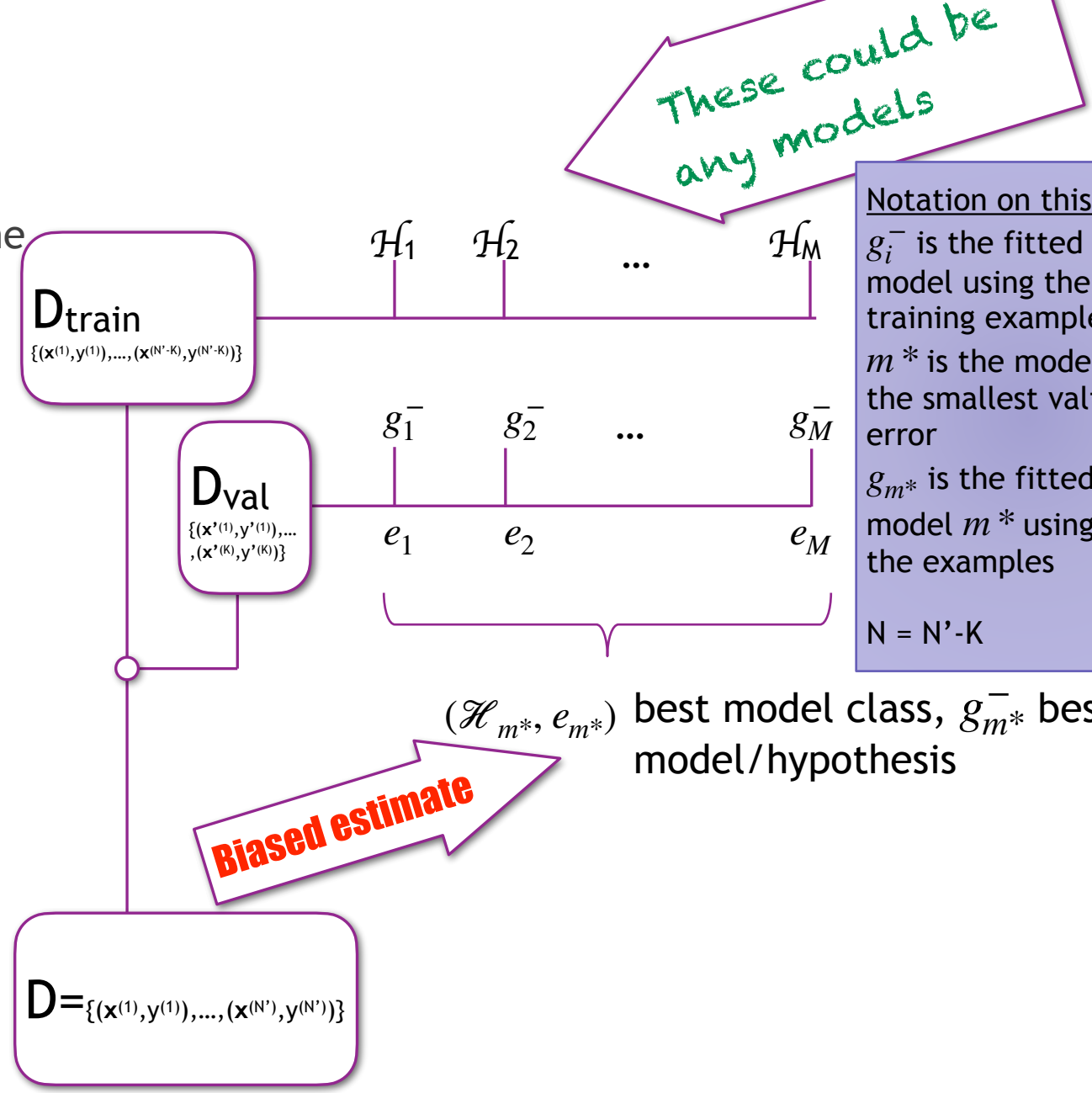
- Pick the best performing model (hypothesis) to be the model with the lowest validation error. Call the best hyperparameters $d^*$
- **Estimate out of sample error** $E_{out}$ of the best model using **test data** (e.g., $\mathbf{w}_{d*}$)
- Typical splits:


| | | |
|---|---|---|
| 10% | 10% | 80% |
| 10% | 20% | 70% |
| 25% | 25% | 50% |

https://www.stat.cmu.edu/~cshalizi/uADA/12/lectures/ch03.pdf

# Selecting a model using a validation set

- For each $H_i$, fit its optimal hypothesis $g_i^-$ using the training set $D_{train}$
- For each $g_i^-$ estimate the out-of-sample error $e_i$ using $D_{val}$
- Pick the $\mathcal{H}_{m*}$ that had the smallest validation error.
- If we have a test set, use it to estimate the error of $g_{m*}^-$



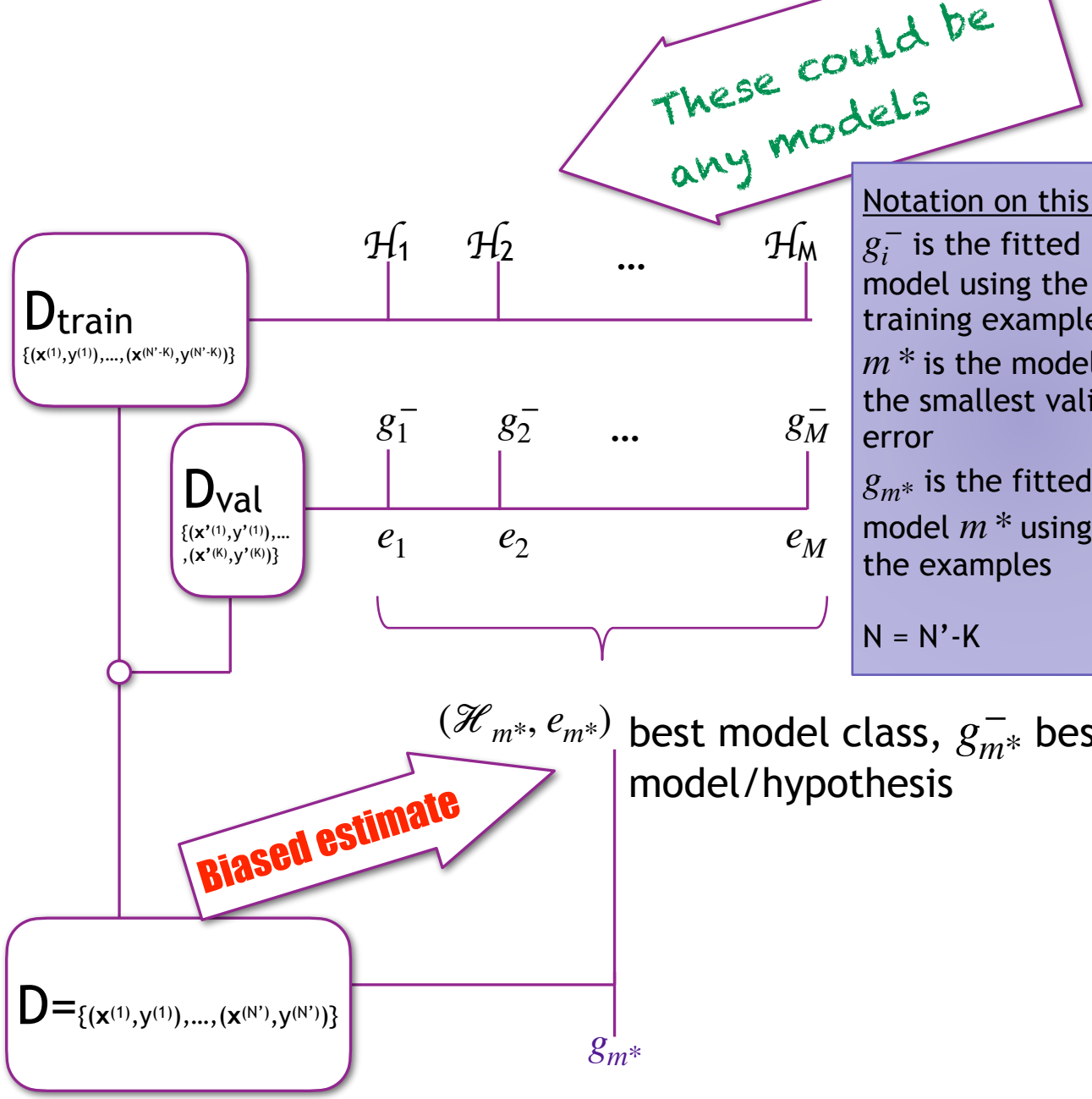

Approach taken from pages 19-24 and 143in *Learning from Data*

# Selecting a model using a validation set

If we didn't have a large training set:

- We can then train the selected model using all the data $D$ (i.e. use both $D_{train}$ and $D_{val}$). Let $g_{m*}$ be the optimal hypothesis found this way.
- If we have a test set, we can use that to estimate the error of $g_{m*}$

What if we don't have enough data for a test set?




Approach taken from pages 19-24 and 143in *Learning from Data*

# Outline

- Motivating example: What polynomial degree should e?
- Polynomial transformation
- Underfitting and overfitting
- Understanding error: Bias and varian
- Learning curves
- Validation and model selection
- Limited dataset:
  - Do we need a test set?
  - K-fold cross-validation
- Regularization

Yea!

Uh oh….

How to create a more complex hypothesis

Understanding what wrong

Understanding where the error comes from, and how to estimate $E_{\text{out}}[g(\mathbf{x})]$

Understanding what went wrong

y different hypothesis classes how can we choose wisely? e estimate $E_{\text{out}}[g(\mathbf{x})]$?

# Strategies for dealing with a small dataset

# Outline

- ❑ Motivating example: What polynomial degree should e?
- ❑ Polynomial transformation
- ❑ Underfitting and overfitting
- ❑ Understanding error: Bias and varian
- ❑ Learning curves
- ❑ Validation and model selection
- ❑ Limited dataset:
  - Do we need a test set?
  - K-fold cross-validation
- ❑ Regularization

Yea!

Uh oh….

How to create a more complex hypothesis

Understanding what wrong

Understanding where the error comes from, and how to estimate $E_{\text{out}}[g(\mathbf{x})]$

Understanding what went wrong

y different hypothesis classes how can we choose wisely? e estimate $E_{\text{out}}[g(\mathbf{x})]$?

# Thought experiment

Two hypothesis $g_1, g_2$

$$E_{\text{out}}(g_1) = E_{\text{out}}(g_2) = \frac{1}{2}$$

Given the error $e_1, e_2$ estimates for the hypothesis
where we assume (for this thought experiment) that $e_1, e_2$ is uniform on $[0,1]$

pick $g \in \{g_1, g_2\}$ where $e = \min(e_1, e_2)$

**1. If we have enough examples in the validation set, is e is a good estimate of E_out?**

- yes, it is unbiased
- it is an optimistic estimate, but relatively good estimate
- no, it is not a good estimate

This example comes from Abu-Mostafa

# Thought experiment

| $e_1$ | $e_2$ | $e = \min\{e_1, e_2\}$ |
|---|---|---|
| $e_1 > 0.5$ | $e_2 > 0.5$ | $e > 0.5$ |
| $e_1 < 0.5$ | $e_2 > 0.5$ | $e < 0.5$ |
| $e_1 > 0.5$ | $e_2 < 0.5$ | $e < 0.5$ |
| $e_1 < 0.5$ | $e_2 < 0.5$ | $e < 0.5$ |

Two hypothesis $g_1, g_2$

$$E_{\text{out}}(g_1) = E_{\text{out}}(g_2) = \frac{1}{2}$$

Given the error $e_1, e_2$ estimates for the hypothesis
where we assume (for this thought experiment) that $e_1, e_2$ is uniform on $[0,1]$

pick $g \in \{g_1, g_2\}$ where $e = \min(e_1, e_2)$

Notice that $E[e] \leq 0.5$

We have an optimistic biased estimate of the error if we estimate

This example comes from Abu-Mostafa

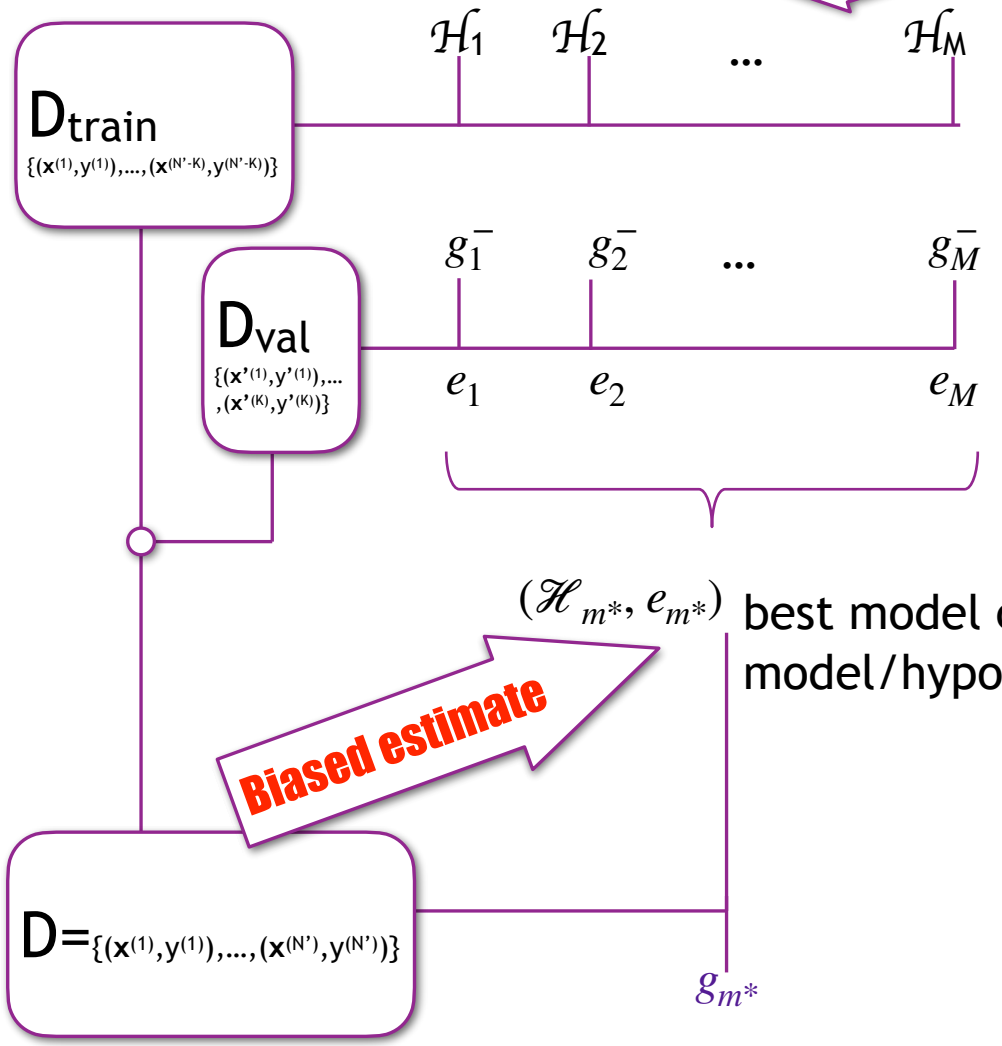
# Using validation error to bound out of sample error

The bound is for errors in the range [0,1]

These could be any models

If our validation set has K items, then with probability $1 - \delta$

$$E_{out}(g_{m*}) \leq^{?} E_{out}(g^-_{m*}) \leq \underbrace{E_{val}(g^-_{m*})}_{e_{m*}} + \sqrt{\frac{\ln 2M + \ln \frac{1}{\delta}}{2K}}$$




Notation on this slide

$g_i^-$ is the fitted $i^{th}$ model using the training examples

$m*$ is the model with the smallest validation error

$g_{m*}$ is the fitted model $m*$ using all the examples

N = N'-K

$(\mathcal{H}_{m*}, e_{m*})$ best model class, $g^-_{m*}$ best model/hypothesis



Approach taken from pages 19-24 and 143in *Learning from Data*

http://cs229.stanford.edu/extra-notes/hoeffding.pdf

# Outline

- ❑ Motivating example: What polynomial degree should e?
- ❑ Polynomial transformation
- ❑ Underfitting and overfitting
- ❑ Understanding error: Bias and varian
- ❑ Learning curves
- ❑ Validation and model selection
- ❑ Limited dataset:
  - Do we need a test set?
  - K-fold cross-validation
- ❑ Regularization

Yea!

Uh oh….

How to create a more complex hypothesis

Understanding what wrong

Understanding where the error comes from, and how to estimate $E_{\text{out}}[g(\mathbf{x})]$

Understanding what went wrong

y different hypothesis classes how can we choose wisely? e estimate $E_{\text{out}}[g(\mathbf{x})]$?

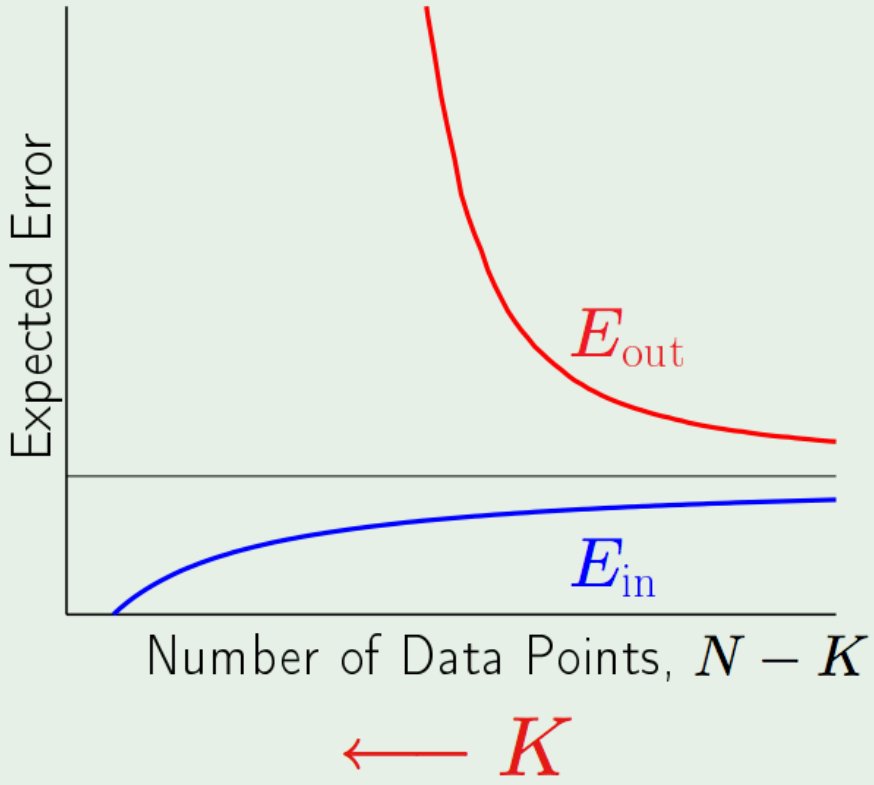
# $K$ is taken out of $N$

Given the data set $\mathcal{D} = (\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})$

$\underbrace{K \text{ points}}_{\mathcal{D}_{\text{val}}} \rightarrow$ validation $\quad \underbrace{N - K \text{ points}}_{\mathcal{D}_{\text{train}}} \rightarrow$ training

Small $K \implies$ bad estimate

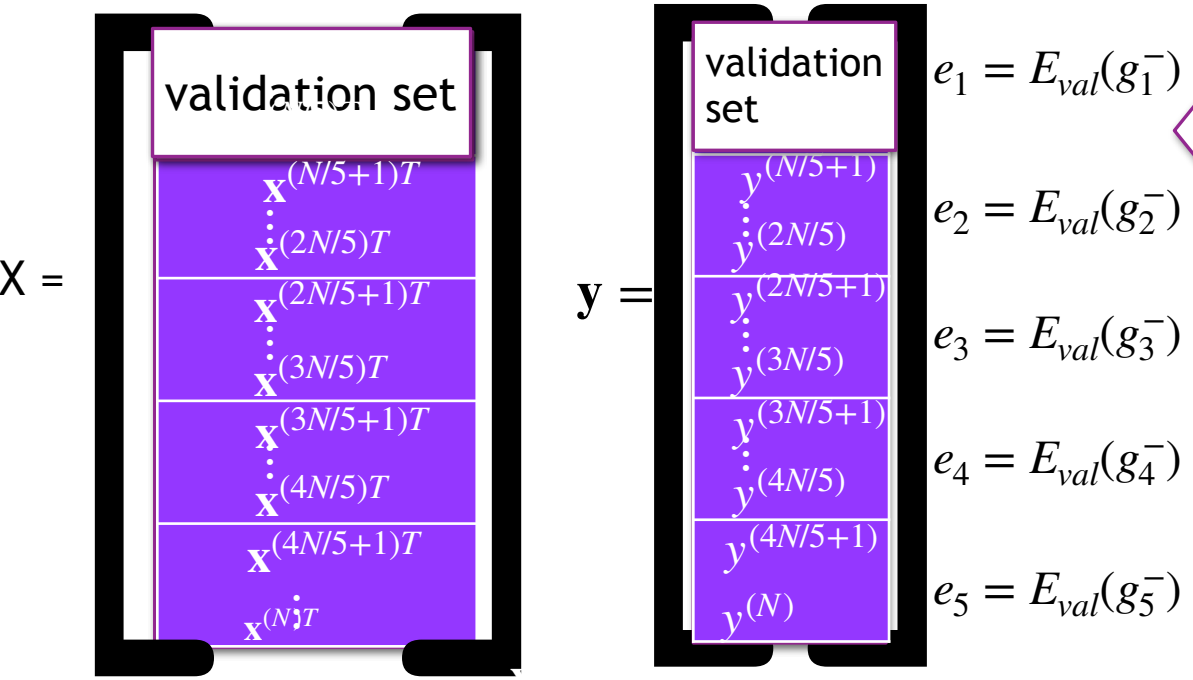
Large $K \implies$ ?




Slide slightly modified from

# Estimating the out of sample error for $\mathcal{H}i$ using K-fold cross-validation

- If there is not enough data, instead of training and validation sets, divide the data into k sets. Commonly K = 5, K = 6, K = 10, or K = N.
- Then train on K-1 of the sets and validate on the remaining set. The estimated error will be the average of the errors.




$e_1 = E_{val}(g_1^-)$

$e_2 = E_{val}(g_2^-)$

$e_3 = E_{val}(g_3^-)$

$e_4 = E_{val}(g_4^-)$

$e_5 = E_{val}(g_5^-)$

not independent error measures

cross validation error

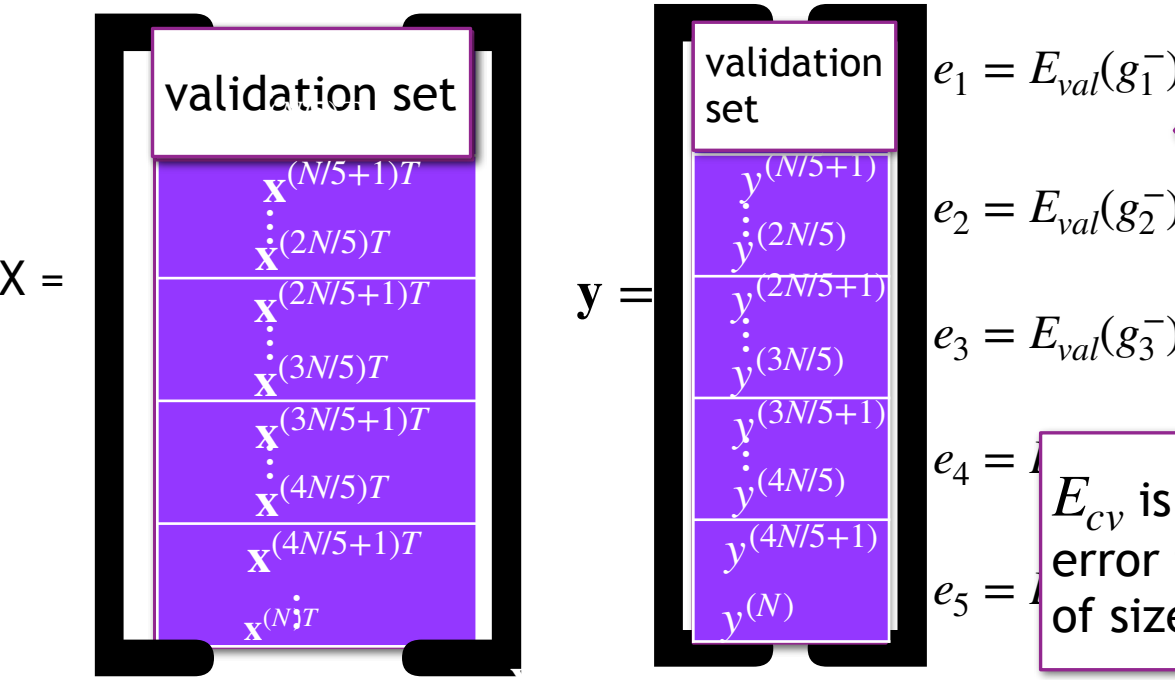
Cross-Validate(D,k)
Divide training data (*randomly*) into K disjoint sets: $D_1, D_2, \ldots, D_K$
For i = 1 to K
  Train on all examples except those in the i*th* set $D - D_i$
  Let $g_i^-$ be the fitted model
  Validation error $e_i = E_{val}(g_i^-)$
Return $E_{cv} = \frac{1}{K}\sum_{i=1}^{K} e_i$

$\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$
$E_{cv,1}$ $E_{cv,2}$ $E_{cv,M}$

During this process, for one model class, you computed $K$ different hypotheses. If you wish to use this model class to predict in the future, run the algorithm again on **all** the data. Or average the result of your K hypotheses.

# Estimating the out of sample error for $\mathcal{H}i$ using K-fold cross-validation

- If there is not enough data, instead of training and validation sets, divide the data into k sets. Commonly K = 5, K = 6, K = 10, or K = N.
- Then train on K-1 of the sets and validate on the remaining set. The estimated error will be the average of the errors.







$e_1 = E_{val}(g_1^-)$

$e_2 = E_{val}(g_2^-)$

$e_3 = E_{val}(g_3^-)$

$e_4 =$

$e_5 =$

not independent error measures

$E_{cv}$ is expected error for all data sets of size N-(N/K)

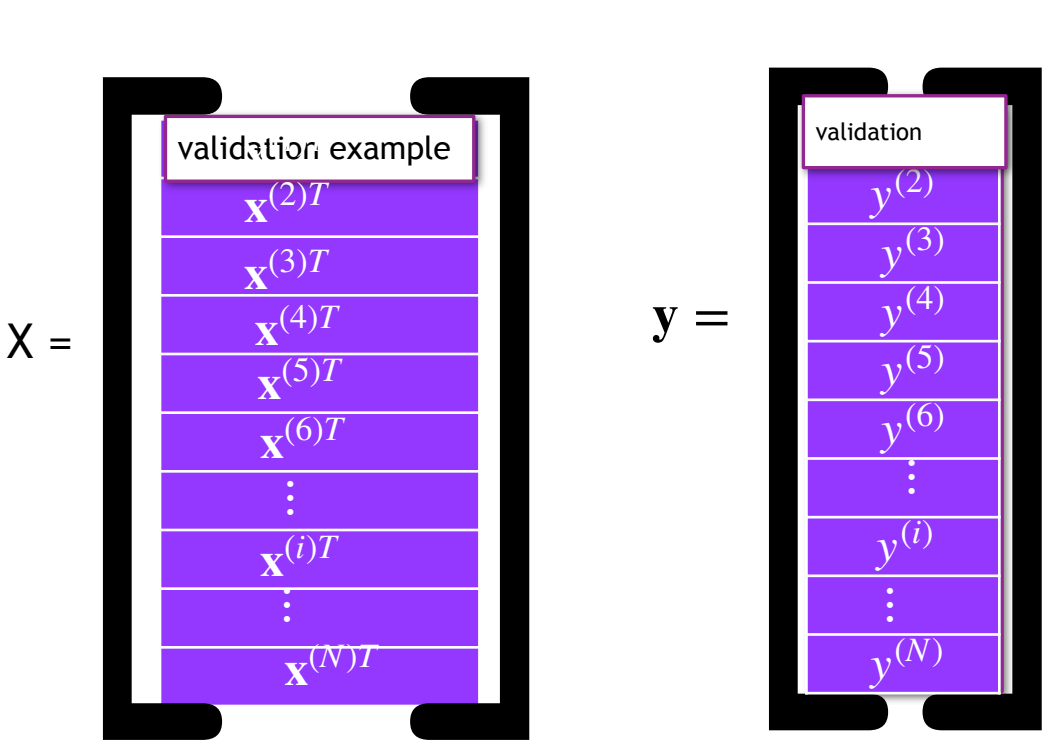
Cross-Validate(D,k)
Divide training data (*randomly*) into K disjoint sets: $D_1, D_2, \ldots, D_K$
For i = 1 to K
  Train on all examples except those in the i*th* set $D - D_i$
  Let $g_i^-$ be the fitted model
  Validation error $e_i = E_{val}(g_i^-)$
Return $E_{cv} = \frac{1}{K}\sum_{i=1}^{K} e_i$

$\mathcal{H}_1, \mathcal{H}_2, \ldots, \mathcal{H}_M$
$E_{cv,1}$ $E_{cv,2}$ $E_{cv,M}$

During this process, for one model class, you computed $K$ different hypotheses. If you wish to use this model class to predict in the future, run the algorithm again on **all** the data. Or average the result of your K hypotheses.

# Leave-one-out cross validation

- If there is very little data, let K = N. This will give the best estimate but the running time may be prohibitive. This is called *leave-one-out cross-validation because* 1 example was left out at a time




Only one example in validation set, N-1 examples in the training data

During this process, you computed $N$ different classifiers.


39

# Extra slides not presented in class

# Polynomial Features in Scikit-Learn

Generate polynomial transformation of the feature space

"Generate a new feature matrix consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, if an input sample is two dimensional and of the form [a, b], the degree-2 polynomial features are [1, a, b, $a^2$, ab, $b^2$]."

```
from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

Parameters include:
**include_bias : *boolean***
The default is True

"Be aware that the number of features in the output array scales polynomially in the number of features of the input array, and exponentially in the degree. High degrees can cause overfitting."

Example from http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html

# K-Fold Validation in Sklearn

```
from sklearn import linear_model
import  sklearn.model_selection
```

```
regr = linear_model.LinearRegression()
```

```
nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)
```

Create a K-fold object

Set shuffle to true

```
dval = np.arange(1,nfold) #[1,..., nfold-1]
nd = len(dval)
```

Select which degree d models to test

[1 2 3 4 5 6 7 8 9]

```
# Loop over the folds
MSEval = np.zeros((nd,nfold)) #create a matrix to hold all the values
for isplit, Ind in enumerate(kf.split( X )):
    I_train, I_val = Ind
    X_train = X[I_train]
    y_train = y[I_train]
    X_val = X[I_val]
    y_val = y[I_val]
```

"Generate indices to split data into training and test set."
**Ind** holds indices for training and validation sets

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  9, 10, 11, 12, 13, 14, 16, 17, 18,
        19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36,
        37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
        54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 69, 70, 71,
        72, 73, 74, 77, 78, 79, 80, 81, 83, 84, 85, 86, 88, 89, 92, 93, 94,
        95, 96, 97, 98, 99]), array([ 8, 15, 31, 68, 75, 76, 82, 87, 90, 91]))
```

```
for it, d in enumerate(dval):
    # polynomial feature transformation
    poly_transformation = PolynomialFeatures(degree=d,include_bias=False)
    X_train_d = poly_transformation.fit_transform(X_train)
    X_val_d = poly_transformation.transform(X_val)
```

If d=1 $\mathbf{x}^{(0)}$=[0.456]
$\Phi_1(\mathbf{x}^{(0)})$= [0.456]
No change (identity transformation)

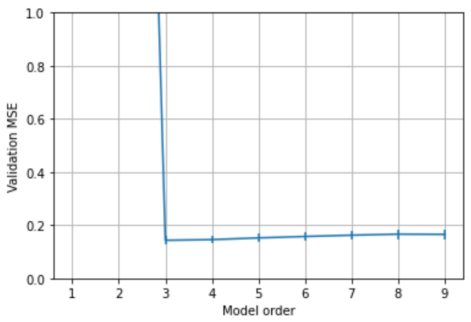
If d = 2,$\mathbf{x}^{(0)}$= [0.456]
$\Phi(\mathbf{x}^{(0)})$=[0.456 0.208]

```
    # Fit the training data
    regr.fit(X_train_d,y_train)

    # Measure MSE on test data
    yhat_val = regr.predict(X_val_d)
    MSEval[it,isplit]= np.mean((yhat_val-y_val)**2)
```

MSEval =

```
[[ 6.77  4.45  7.68  5.44  3.    5.08  4.25  9.32  9.78  5.66]
 [ 6.74  4.42  7.66  5.46  3.02  5.09  4.26 11.03  9.87  5.66]
 [ 0.16  0.23  0.12  0.11  0.06  0.14  0.2   0.2   0.21  0.06]
 [ 0.16  0.23  0.12  0.11  0.05  0.15  0.21  0.19  0.21  0.06]
 [ 0.16  0.23  0.12  0.11  0.05  0.15  0.21  0.19  0.21  0.06]
 [ 0.17  0.23  0.13  0.11  0.05  0.15  0.23  0.2   0.22  0.06]
 [ 0.21  0.24  0.13  0.13  0.06  0.15  0.23  0.22  0.22  0.06]
 [ 0.21  0.25  0.14  0.12  0.06  0.15  0.23  0.22  0.22  0.06]
```




```
MSE=[5.99764938, 6.17617688, 0.14325349, 0.14562991, 0.15220936, 0.15745814, 0.16233707, 0.16630905, 0.16562408]
```

```
MSE = np.mean(MSEval,axis=1)
```

```
imin = np.argmin(MSE)
```

The selected model order is 3

# K-Fold Validation in Sklearn

```
from sklearn import linear_model
import  sklearn.model_selection
```

```
regr = linear_model.LinearRegression()
```

```
nfold = 10
kf = sklearn.model_selection.KFold(n_splits=nfold,shuffle=True)
```

Create a K-fold object

Set shuffle to true

```
dval = np.arange(1,nfold) #[1,..., nfold-1]
nd = len(dval)
```

Select which degree d models to test

[1 2 3 4 5 6 7 8 9]

```
# Loop over the folds
MSEval = np.zeros((nd,nfold)) #create a matrix to hold all the values
for isplit, Ind in enumerate(kf.split( X )):
    I_train, I_val = Ind
    X_train = X[I_train]
    y_train = y[I_train]
    X_val = X[I_val]
    y_val = y[I_val]
```

"Generate indices to split data into training and test set."
**Ind** holds indices for training and validation sets

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  9, 10, 11, 12, 13, 14, 16, 17, 18,
        19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33, 34, 35, 36,
        37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53,
        54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 69, 70, 71,
        72, 73, 74, 77, 78, 79, 80, 81, 83, 84, 85, 86, 88, 89, 92, 93, 94,
        95, 96, 97, 98, 99]), array([ 8, 15, 31, 68, 75, 76, 82, 87, 90, 91]))
```

```
for it, d in enumerate(dval):
    # polynomial feature transformation
    poly_transformation = PolynomialFeatures(degree=d,include_bias=False)
    X_train_d = poly_transformation.fit_transform(X_train)
    X_val_d = poly_transformation.transform(X_val)
```

If d = 3, $\mathbf{x}^{(0)}$= [0.456]
$\Phi_3(\mathbf{x}^{(0)})$=[0.456 0.208 0.095]

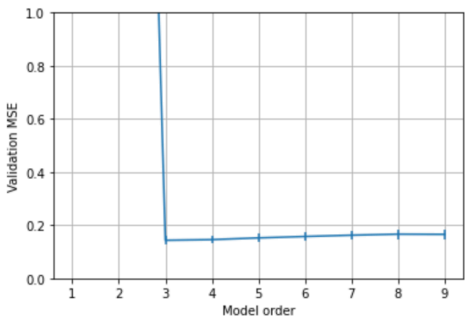
If d = 4, $\mathbf{x}^{(0)}$= [0.456]
$\Phi_4(\mathbf{x}^{(0)})$=[0.456 0.208 0.095 0.043]

```
    # Fit the training data
    regr.fit(X_train_d,y_train)

    # Measure MSE on test data
    yhat_val = regr.predict(X_val_d)
    MSEval[it,isplit]= np.mean((yhat_val-y_val)**2)
```

MSEval =

```
[[ 6.77  4.45  7.68  5.44  3.    5.08  4.25  9.32  9.78  5.66]
 [ 6.74  4.42  7.66  5.46  3.02  5.09  4.26 11.03  9.87  5.66]
 [ 0.16  0.23  0.12  0.11  0.06  0.14  0.2   0.2   0.21  0.06]
 [ 0.16  0.23  0.12  0.11  0.05  0.15  0.21  0.19  0.21  0.06]
 [ 0.16  0.23  0.12  0.11  0.05  0.15  0.21  0.19  0.21  0.06]
 [ 0.17  0.23  0.13  0.11  0.05  0.15  0.23  0.2   0.22  0.06]
 [ 0.21  0.24  0.13  0.13  0.06  0.15  0.23  0.22  0.22  0.06]
 [ 0.21  0.25  0.14  0.12  0.06  0.15  0.23  0.22  0.22  0.06]
```




```
MSE=[5.99764938, 6.17617688, 0.14325349, 0.14562991, 0.15220936, 0.15745814, 0.16233707, 0.16630905, 0.16562408]
```

```
MSE = np.mean(MSEval,axis=1)
```

```
imin = np.argmin(MSE)
```

The selected model order is 3