Stochastic gradient descent

GD minimizes:

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \underbrace{\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)}_{\ln\left(1+e^{-y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n}\right)} \longleftarrow \text{in logistic regression}$$

by iterative steps along $-\nabla E_{\text{in}}$:

$$\Delta \mathbf{w} = -\, \eta \, \nabla E_{\text{in}}(\mathbf{w})$$

∇E_{in} is based on all examples $(\mathbf{x}_n, y_n)$

"**batch**" GD

The stochastic aspect

Pick <u>one</u> $(\mathbf{x}_n, y_n)$ at a time. Apply GD to $\mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$

"Average" direction:

$$\mathbb{E}_n\left[-\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)\right] = \frac{1}{N}\sum_{n=1}^{N} -\nabla \mathbf{e}\left(h(\mathbf{x}_n), y_n\right)$$

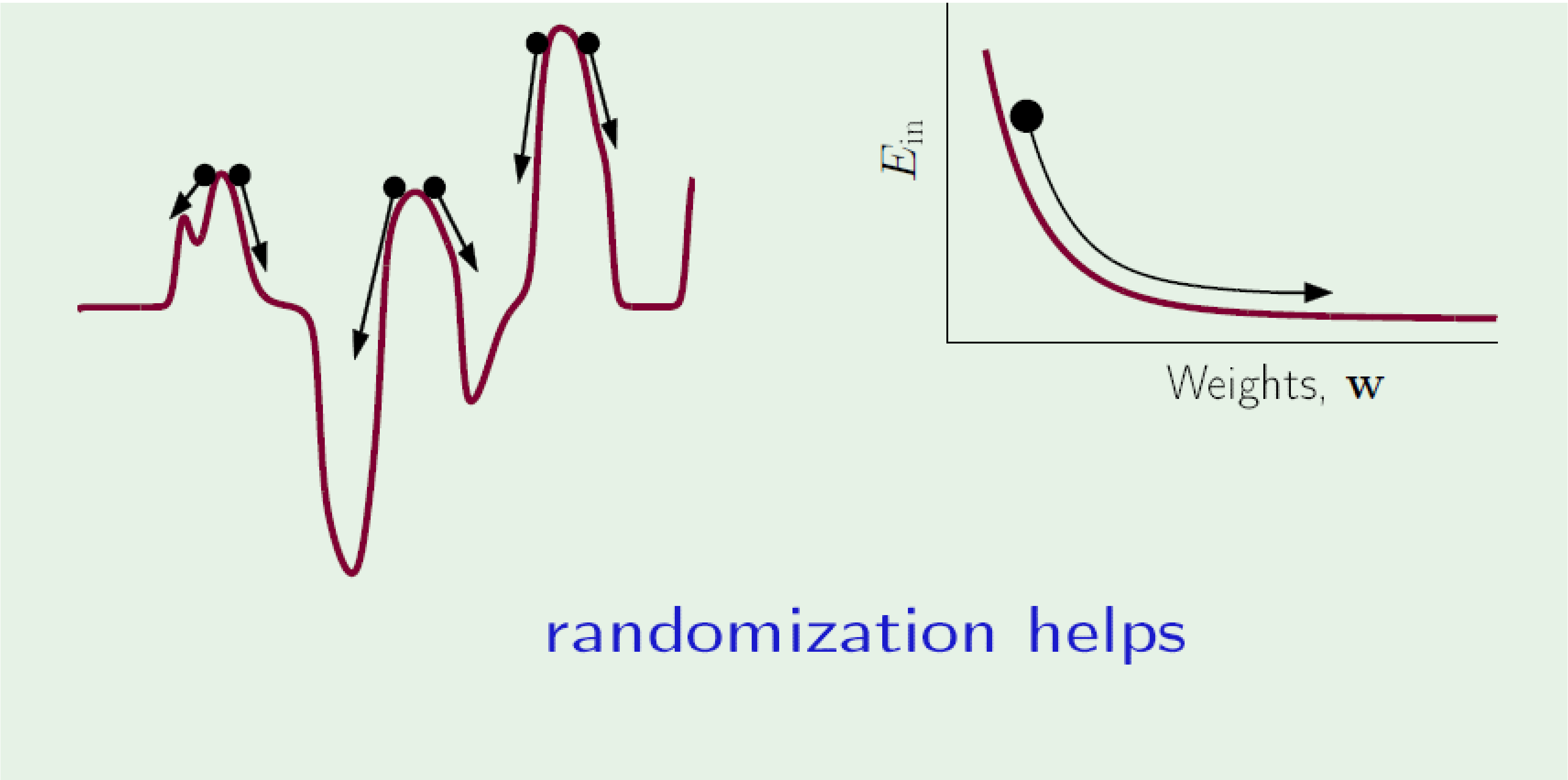
$$= -\nabla\, E_{\text{in}}$$

randomized version of GD

stochastic gradient descent (SGD)

Benefits of SGD

1. cheaper computation
2. randomization
3. simple



Linear Regression

- Training datasets
  $\mathcal{D} = \{(x^{(1)}, t^{(1)}), \dots, (x^{(i)}, t^{(i)}), \dots, (x^{(N)}, t^{(N)})\}$

  (target $t^{(i)}$: real value)
- Linear model
  $y(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M$
- Parameters

  $w_0, w_1, w_2, \dots, w_M$
- Loss function

$$\mathrm{e}(w) = \frac{1}{N} \sum_{i=1}^{N} [t^{(i)} - y(x^{(i)})]^2$$

- Goal: minimize $\mathrm{e}(w)$

Closed-form solution
Directly calculate the pseudo-inverse

Logistic Regression

- Training datasets
  $\mathcal{D} = \{(x^{(1)}, t^{(1)}), \dots, (x^{(i)}, t^{(i)}), \dots, (x^{(N)}, t^{(N)})\}$

  (target $t^{(i)}$: 0 or 1)
- Linear model
  $y(x) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M)$
- Parameters

  $w_0, w_1, w_2, \dots, w_M$
- Loss function

$$\mathrm{e}(w) = \frac{1}{N} \sum_{i=1}^{N} \ln \left[1 + e^{-y_n w^T x_n} \right]$$

- Goal: minimize $\mathrm{e}(w)$

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Linear Regression

- Training datasets
  $$\mathcal{D} = \{(x^{(1)}, t^{(1)}), \dots, (x^{(i)}, t^{(i)}), \dots, (x^{(N)}, t^{(N)})\}$$
  (target $t^{(i)}$: real value)
- Linear model
  $$y(x) = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M$$
- Parameters
  $$w_0, w_1, w_2, \dots, w_M$$
- Loss function
  $$\mathrm{e}(w) = \frac{1}{N}\sum_{i=1}^{N}[t^{(i)} - y(x^{(i)})]^2$$
- Goal: minimize $\mathrm{e}(w)$

Closed-form solution
Directly calculate the pseudo-inverse

Logistic Regression

- Training datasets
  $$\mathcal{D} = \{(x^{(1)}, t^{(1)}), \dots, (x^{(i)}, t^{(i)}), \dots, (x^{(N)}, t^{(N)})\}$$
  (target $t^{(i)}$: 0 or 1)
- Linear model
  $$y(x) = \sigma(w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_M x_M)$$
- Parameters
  $$w_0, w_1, w_2, \dots, w_M$$
- Loss function
  $$\mathrm{e}(w) = \frac{1}{N}\sum_{i=1}^{N} \ln\left[1 + e^{-y_n w^T x_n}\right]$$
- Goal: minimize $\mathrm{e}(w)$

Steps:
- Initialize w (e.g., randomly)
- Repeatedly update w based on the gradient
  $$w = w - \epsilon \nabla_w \ell(w)$$
  where ϵ is the learning rate.

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Key Concepts in Supervised Learning

- Loss function **(measure error, or judge the fit)**
- Optimization **(how to find a good fit)**
- Generalization **(fit to unseen test data)**
- Regularization **(avoid overfitting)**

(very very … very important!!!)

Linear Models for Regression

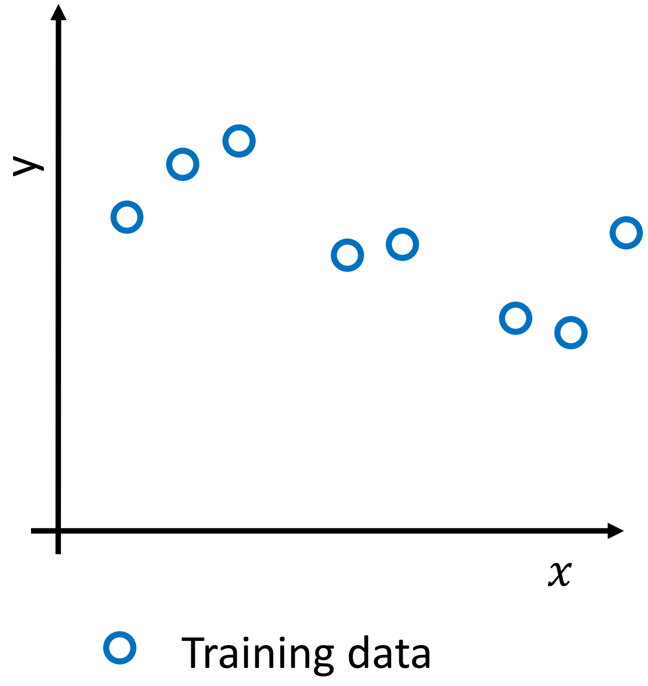
- **Let’s discuss those key questions in the following cases**

  - **Generalization in Supervised Learning (one key concept)**

  - Linear regression using polynomial fitting

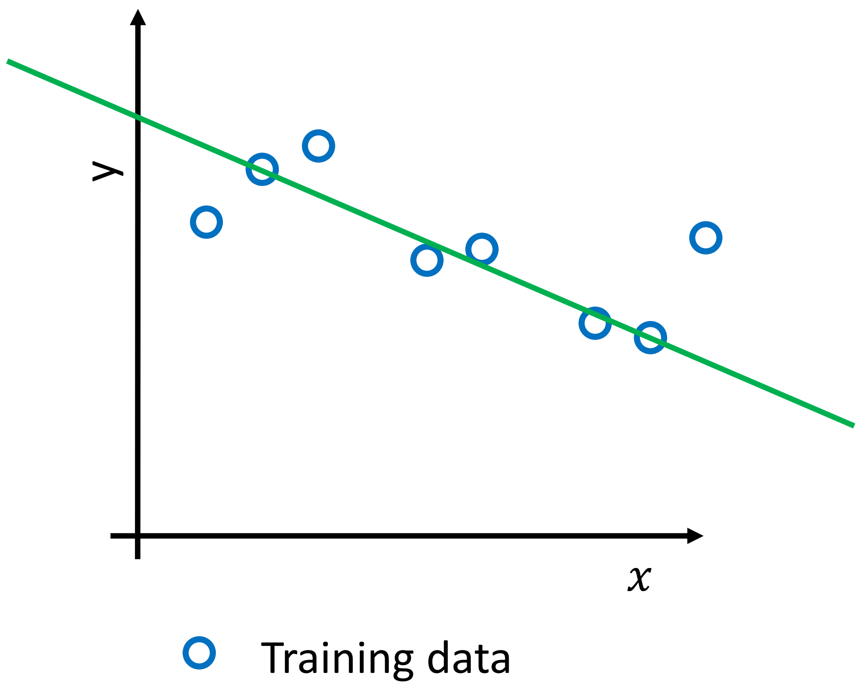
Generalization

- Generalization
  - Model's ability to predict new data



Generalization

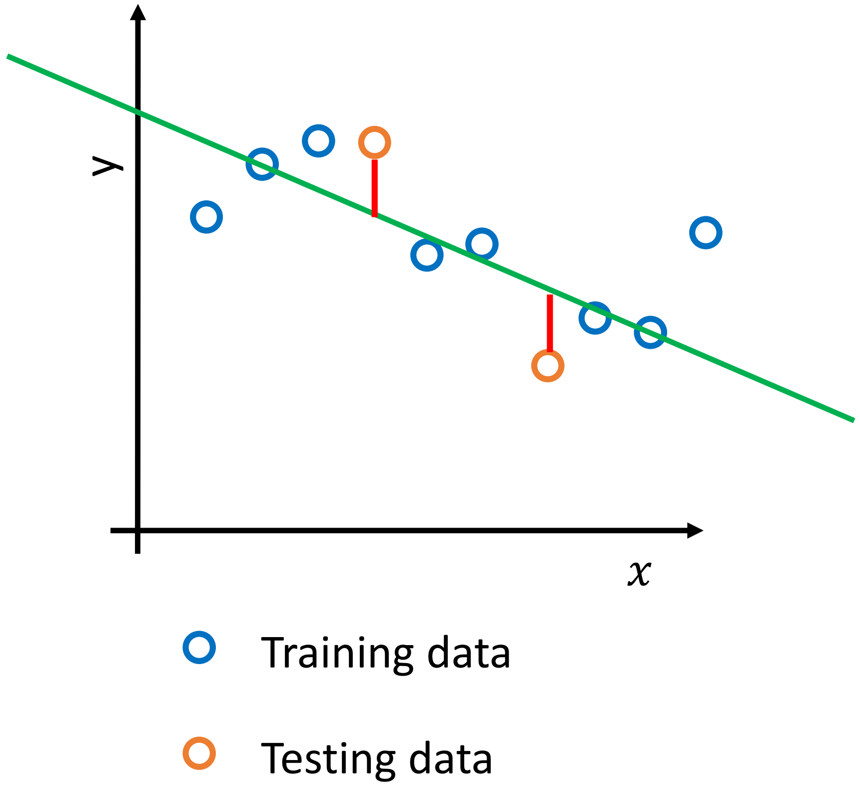
- Generalization
  - Model's ability to predict new data



Generalization

- Generalization
  - Model's ability to predict new data

In fact, what we really care about is the error on new data (in Testing datasets)

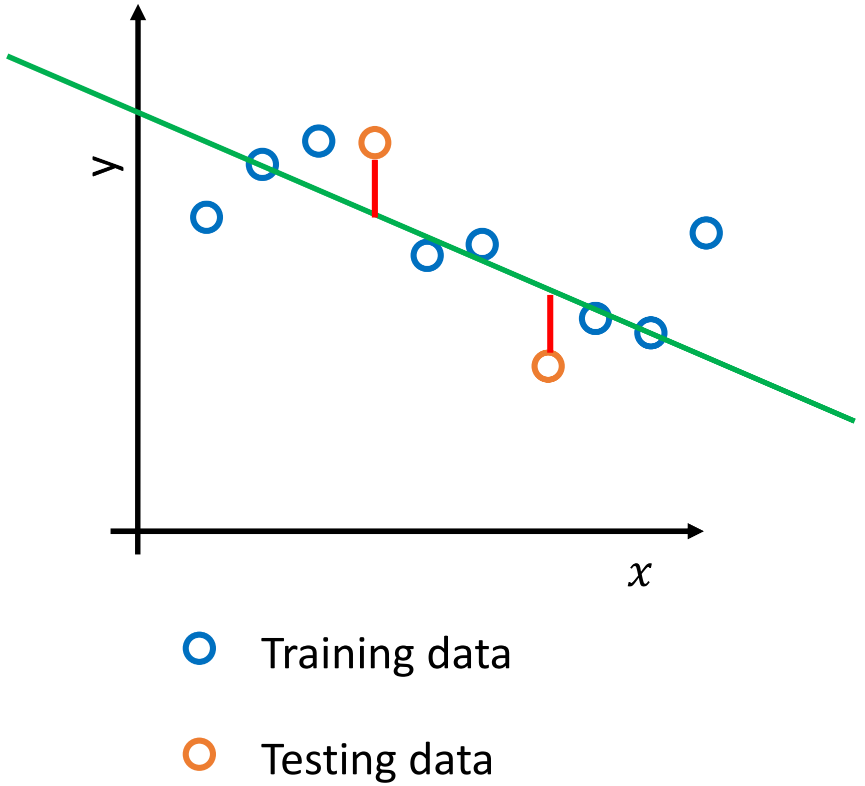


Generalization

- Generalization
  - Model's ability to predict new data

In fact, what we really care about is the error on new data (in Testing dataset)

Datasets in Machine Learning
- Training dataset
- Validation Dataset (important! we will discuss it soon.)
- Testing dataset



Courtesy of Dr. Andrew Ng

Hyperplane

Linear regression models we learned:

- One variable

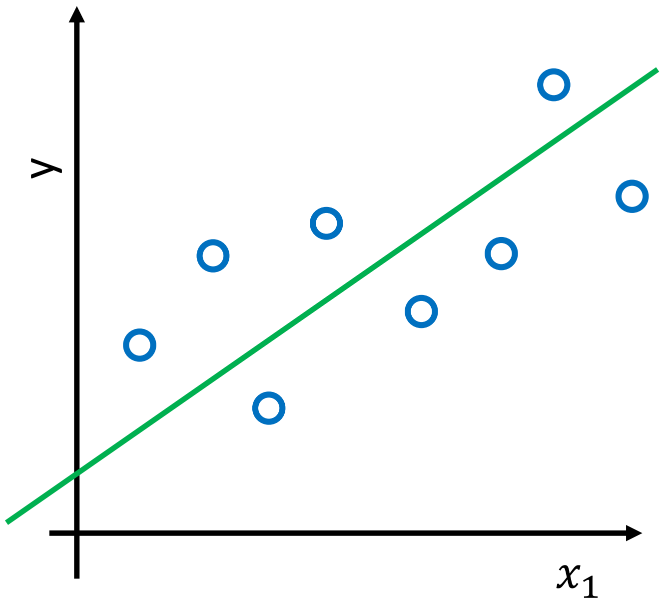
$$y(x) = w_0 + w_1 x_1$$

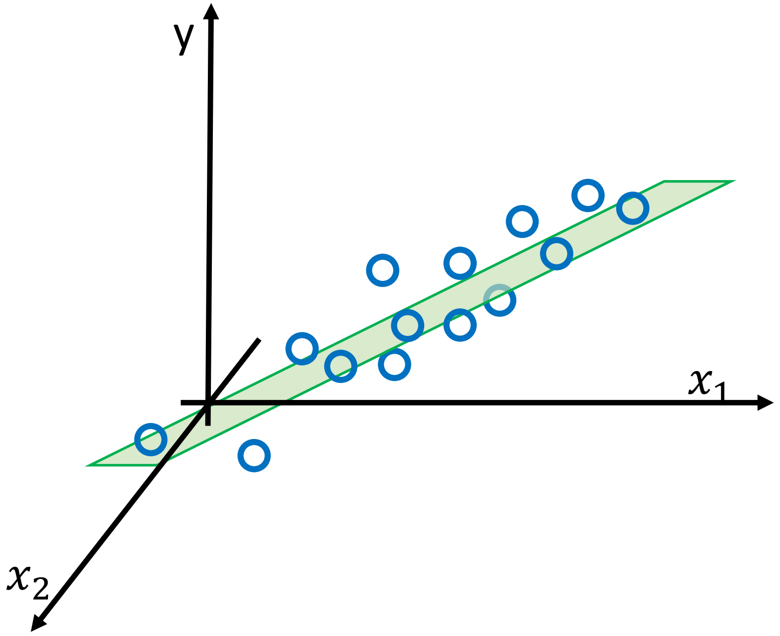
- Two variables

$$y(x) = w_0 + w_1 x_1 + w_2 x_2$$

- More variables

$$y(x) = w_0 + w_1 x_1 + \cdots + w_N x_N$$

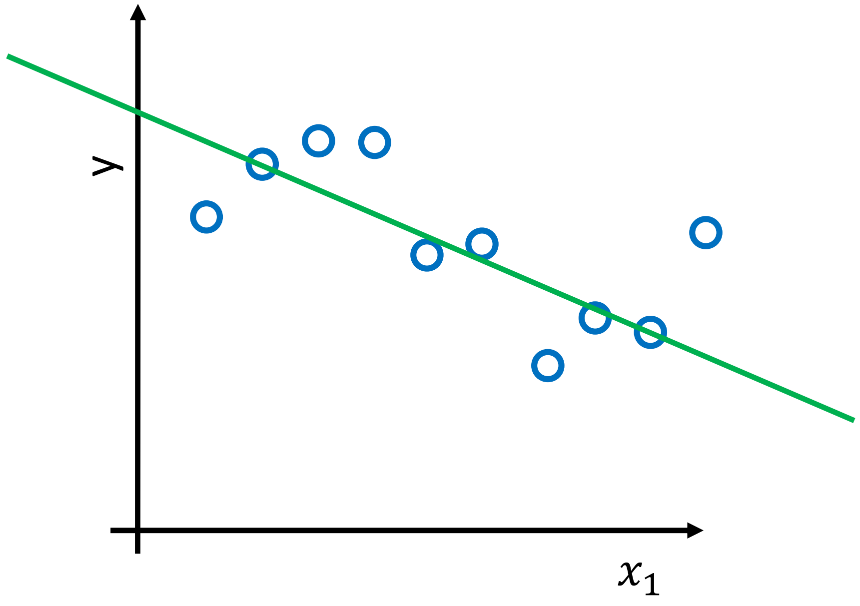




?

Generalization

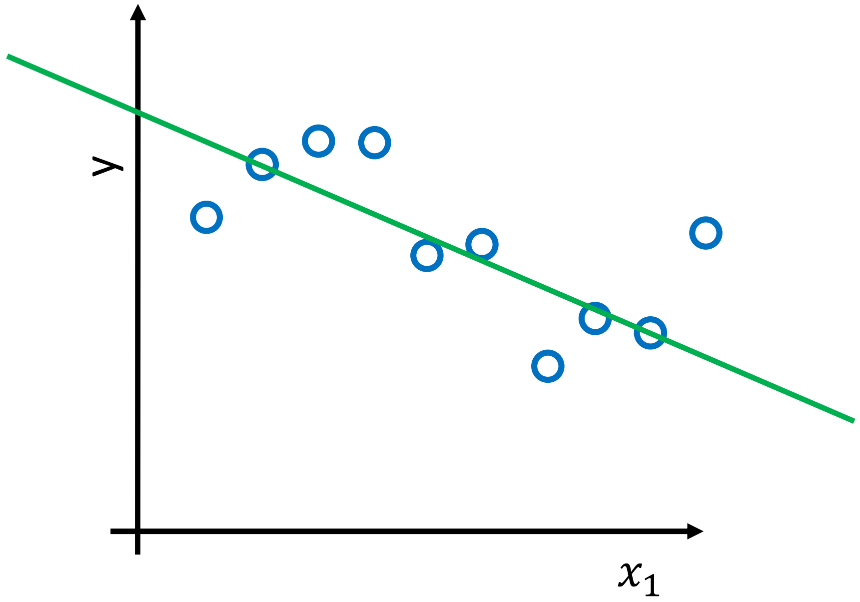
- What if our linear model is not good? (for the data to right)



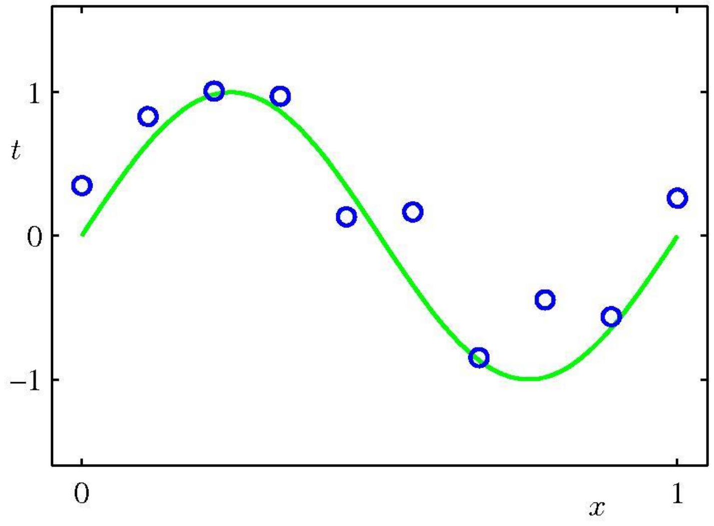
Courtesy of Dr. Sanja Fidler

Generalization

- What if our linear model is not good? (for the data to right)



- We can use a more complicated model (polynomial)



Courtesy of Dr. Sanja Fidler

Linear Models for Regression

- **Let’s discuss those key questions in the following cases**

  - Generalization in Supervised Learning (one key concept)

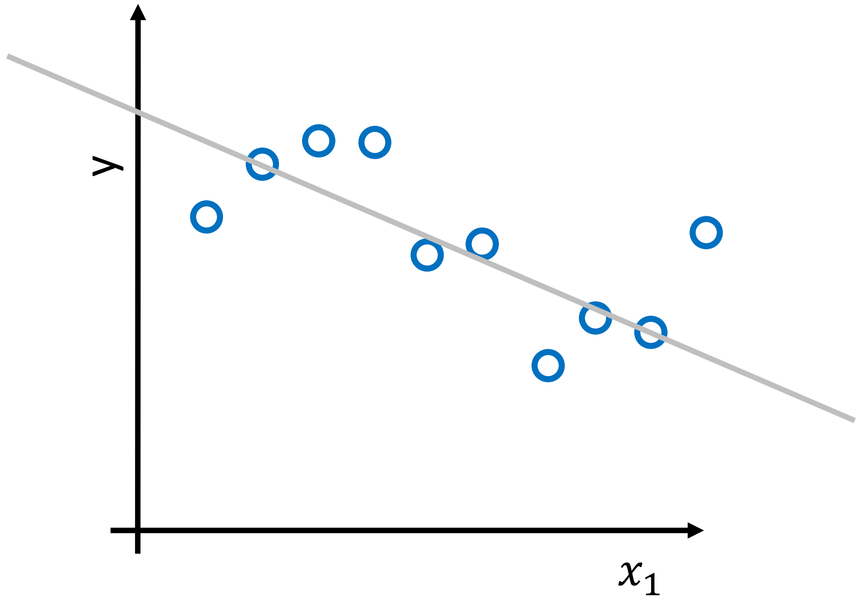
  - **Linear regression using polynomial fitting**

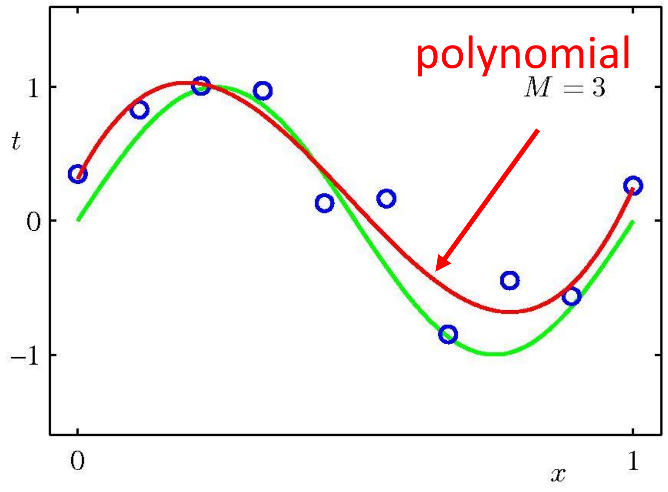
Fitting a Polynomial

- Example: an M-th order polynomial function of one dimensional feature x:

$$y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$$

where x^j is the j-th power of x.





Courtesy of Dr. Sanja Fidler

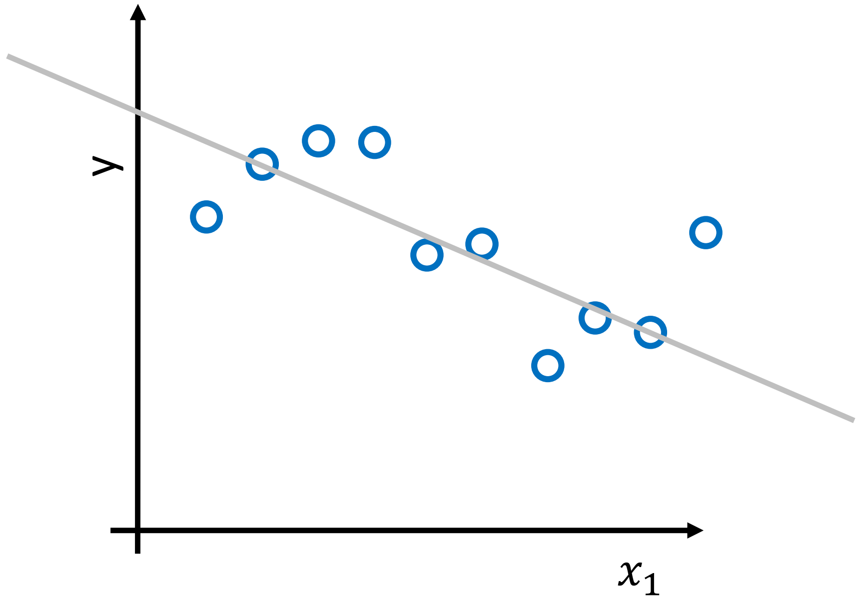
Fitting a Polynomial

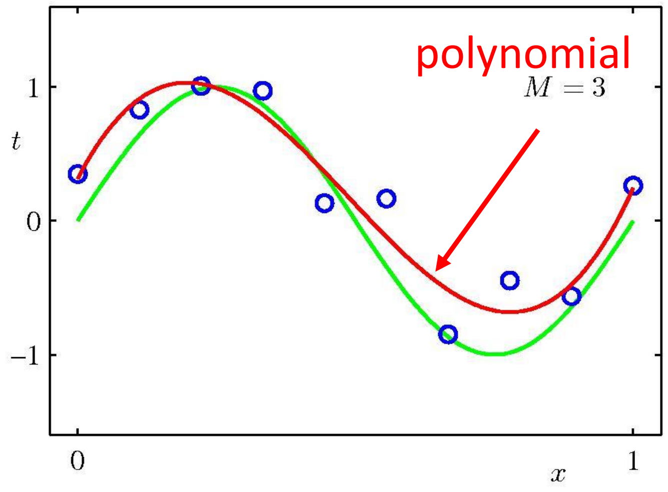
- Example: an M-th order polynomial function of one dimensional feature x:

$$y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$$

where x^j is the j-th power of x.

- Note: We can optimize for the weights w by using the same approach as we did for previous linear model.

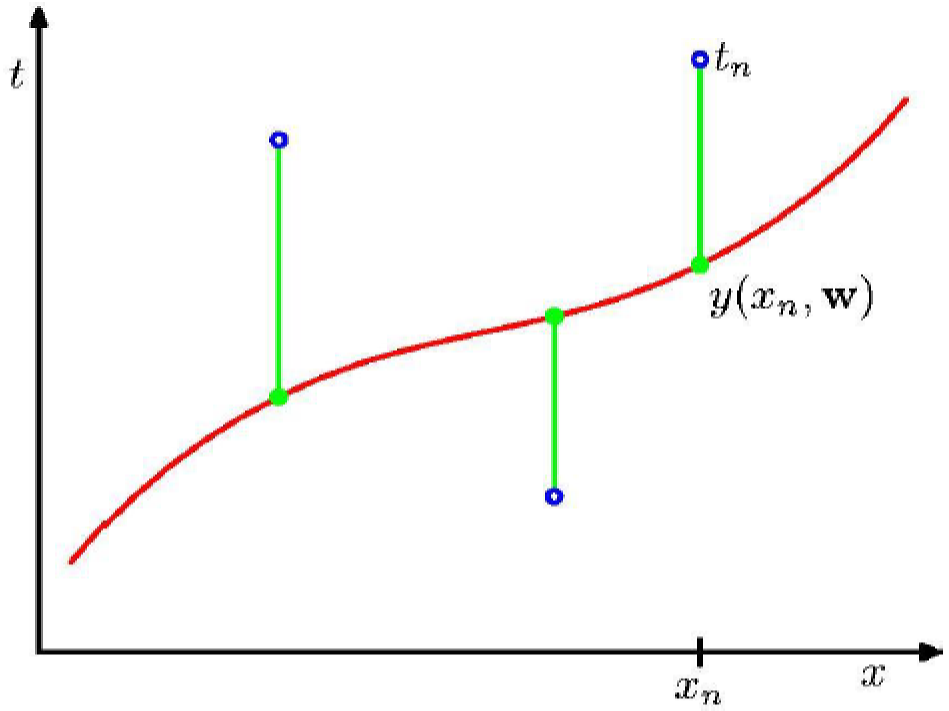




Courtesy of Dr. Sanja Fidler

Sum-of-Squares Error Function

The values of the coefficients will be determined by fitting the polynomial to the training data. This can be done by minimizing an error function that measures the misfit between the function y(x,w), for any given value of w, and the training set data points.

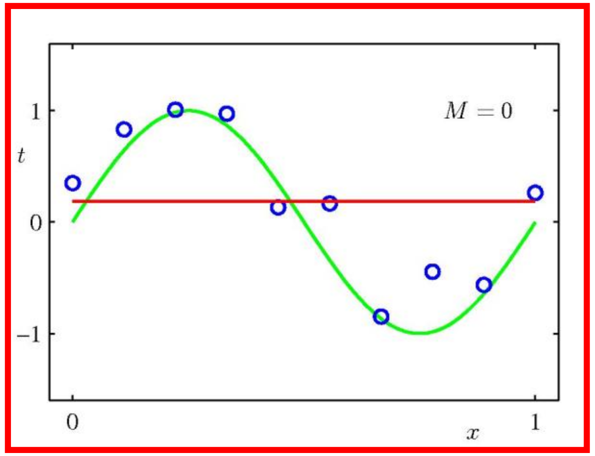


The sum of the squares error function:

$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

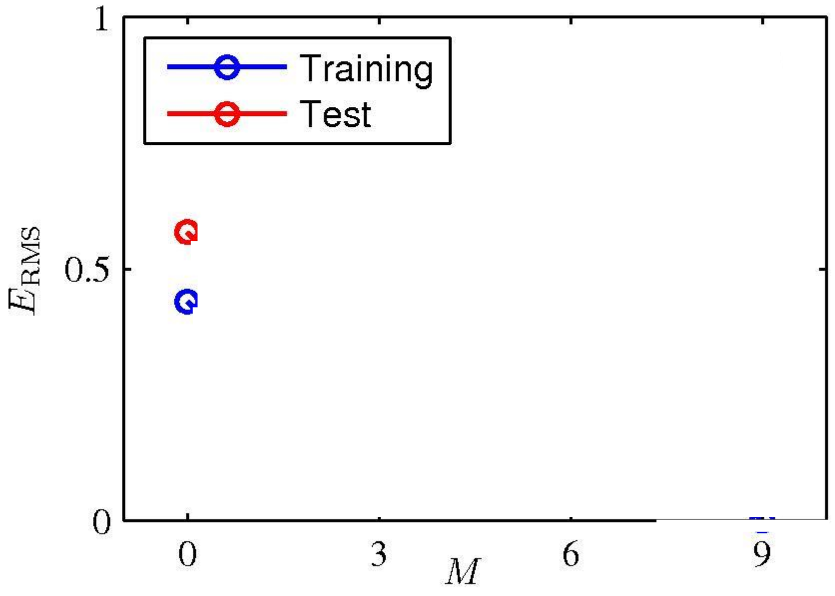
Overfitting

Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^\star)/N}$



- Let's use polynomial model to explain overfitting

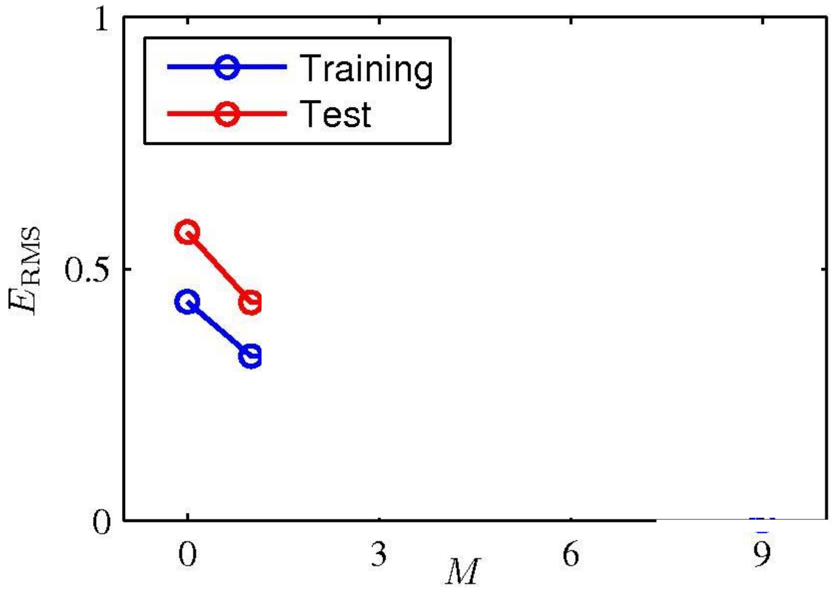
$$y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$$

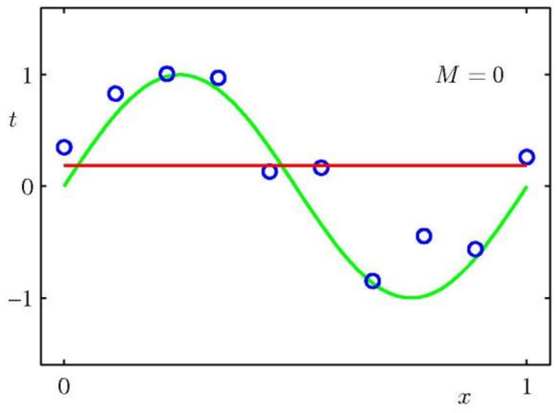


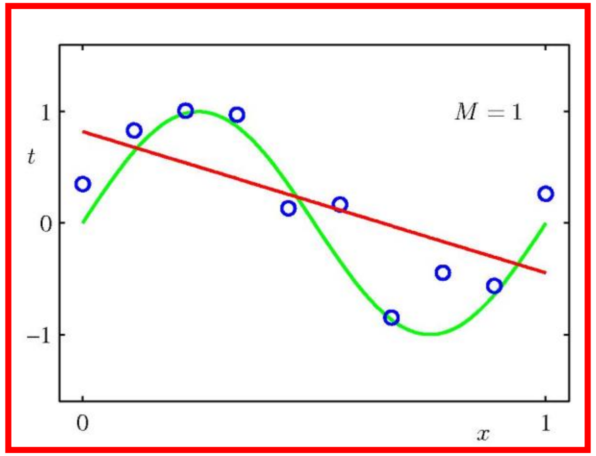
Overfitting

- Let's use polynomial model to explain overfitting

$$y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$$



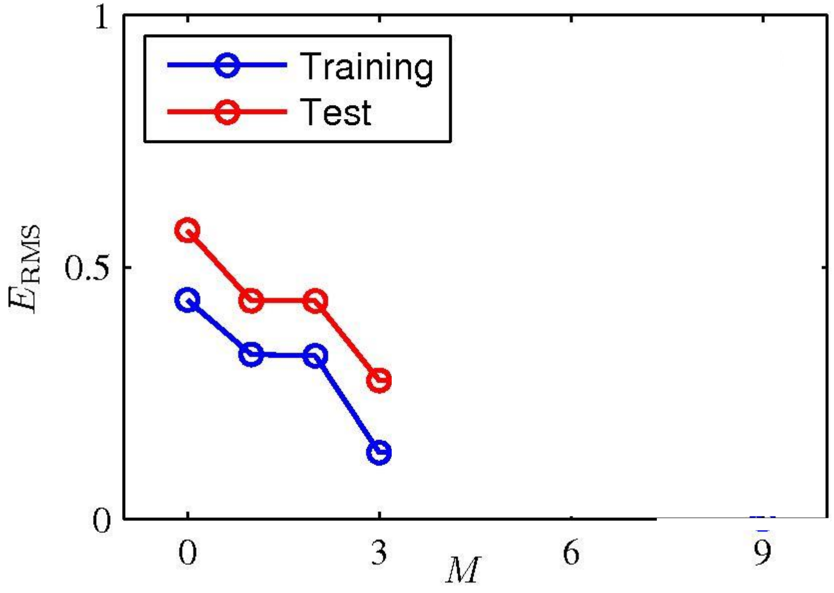


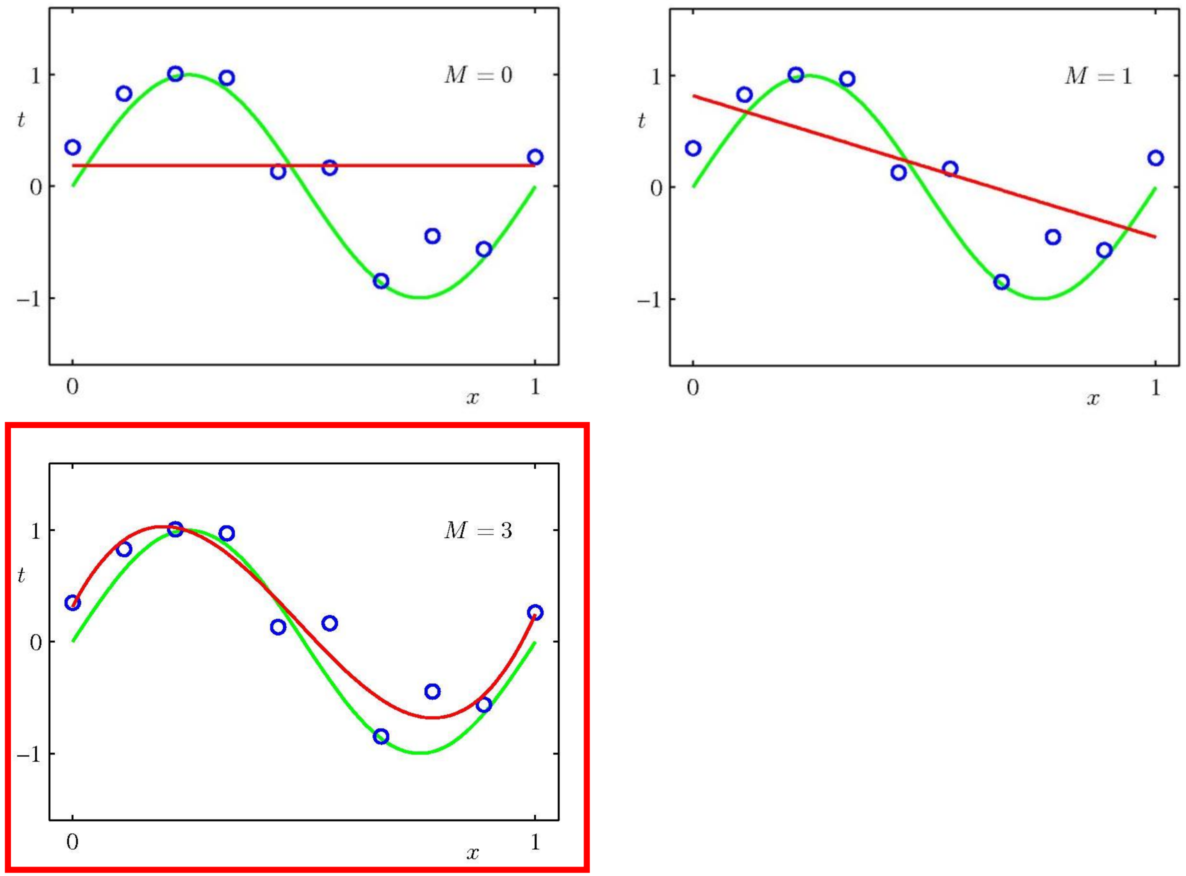


Overfitting

- Let's use polynomial model to explain overfitting

$$y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$$

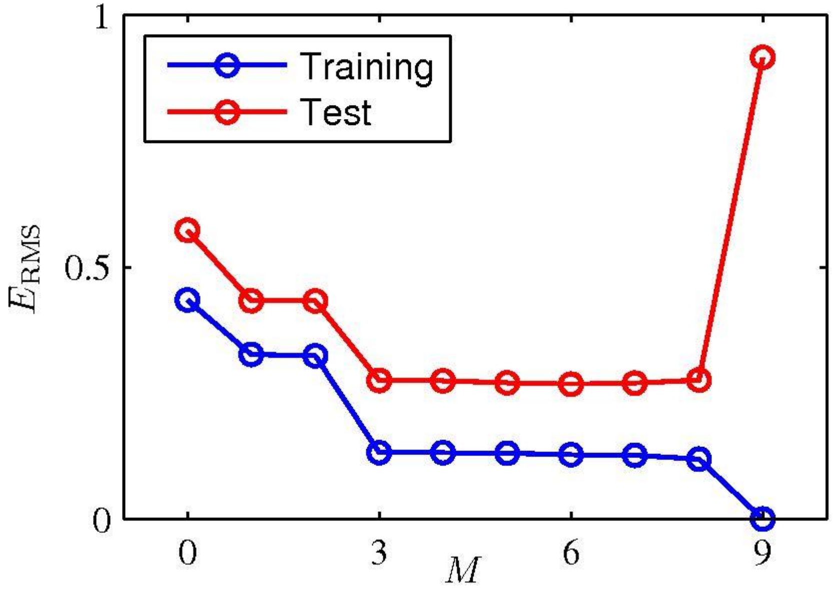


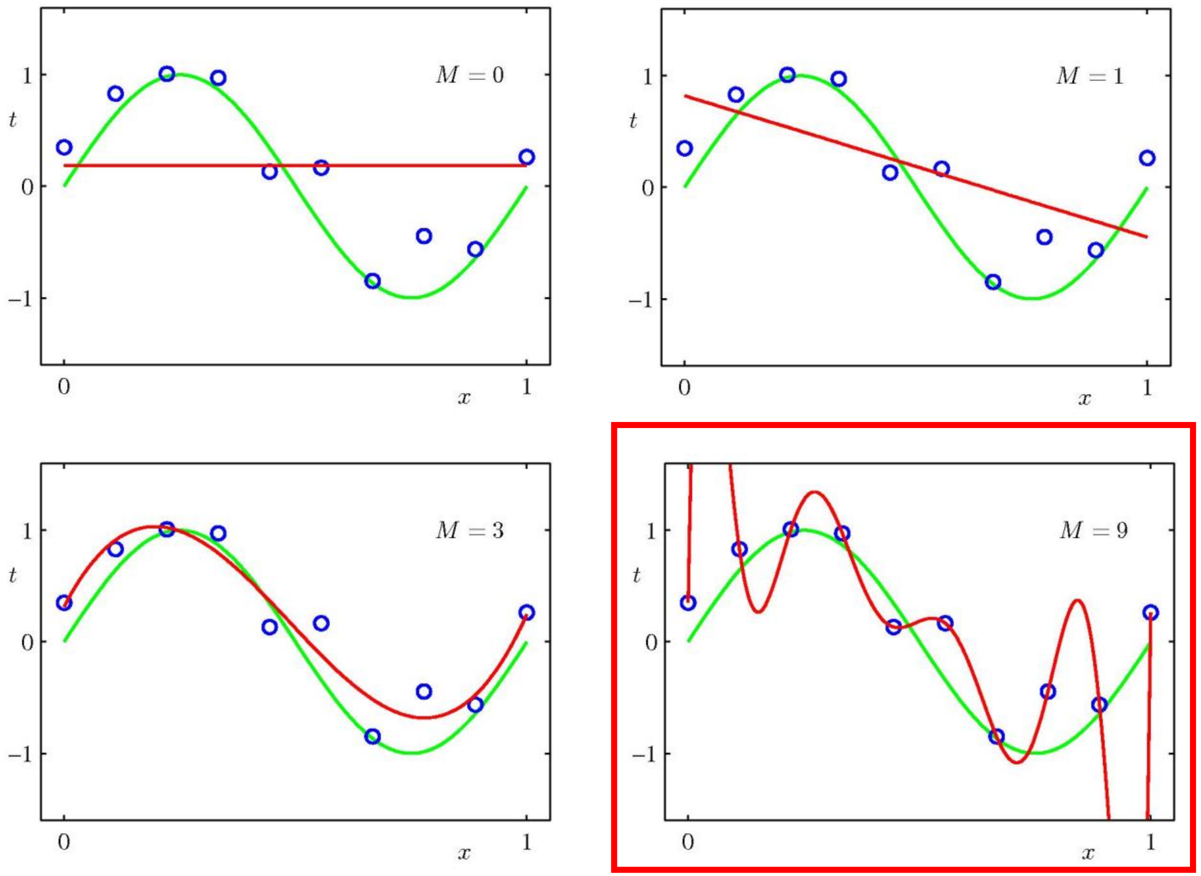


Overfitting

- Let's use polynomial model to explain overfitting

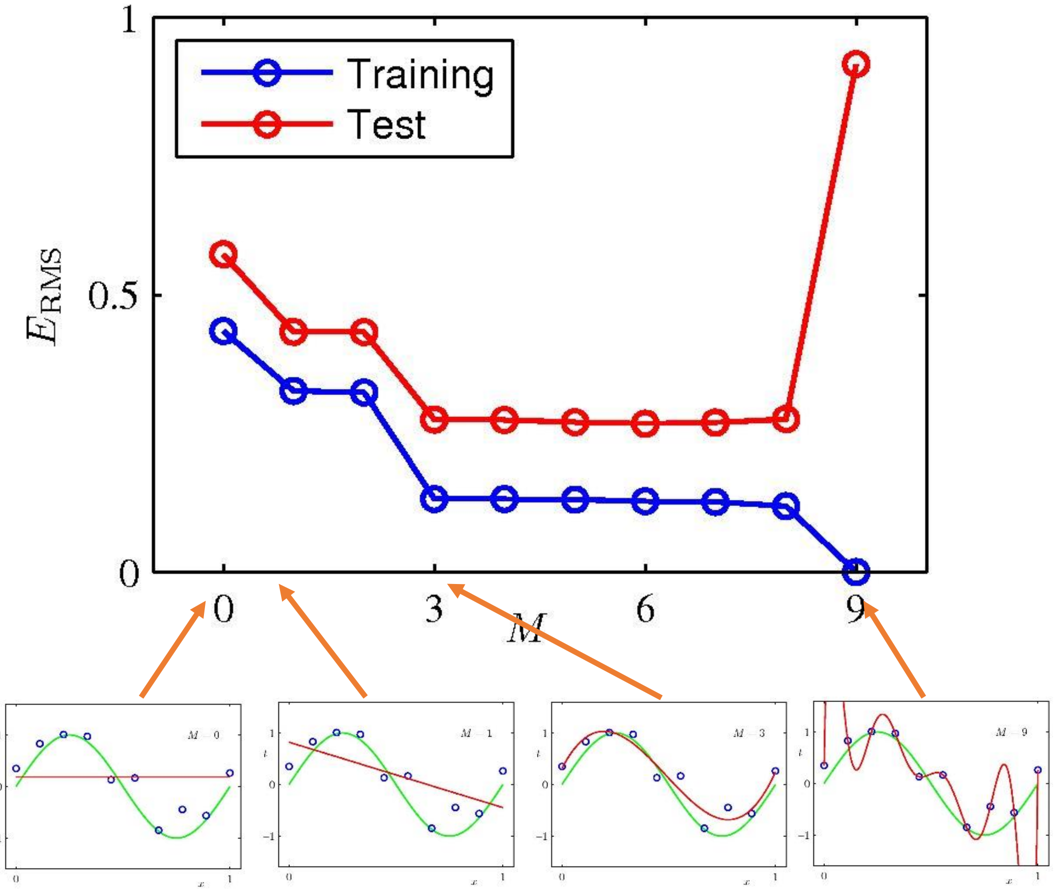
$$y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$$





Overfitting

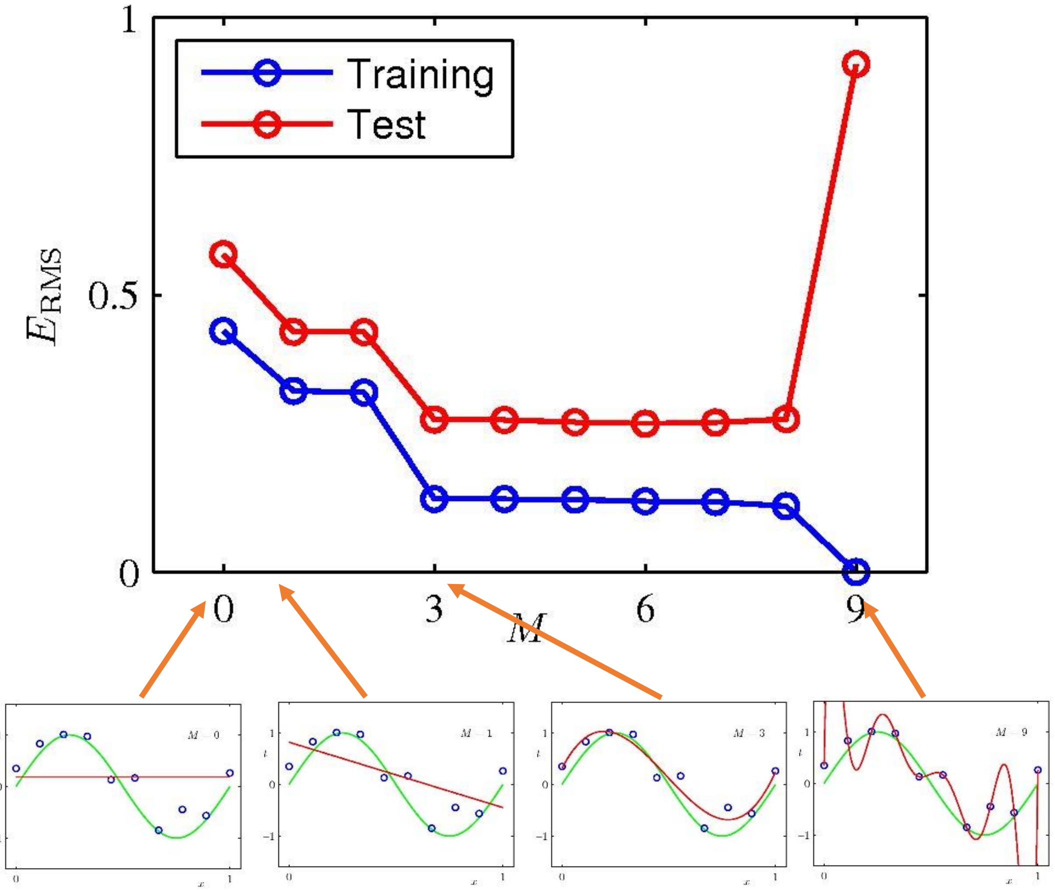
- Observations
  - A more complex model yields lower error on training data. (If we choose truly find the best function, the error on training data may go to zero)
  - A more complex model may perform very badly on testing data (Our model with M = 9 overfits the data)



Courtesy of Dr. Hung-yi Lee

Question?

- Consider $y(x, w) = w_0 + \sum_{j=1}^{M} w_j x^j$
  - Why a more complex model can yield lower error on training data?
  - Why a more complex model can perform very badly on testing data (overfitting)?

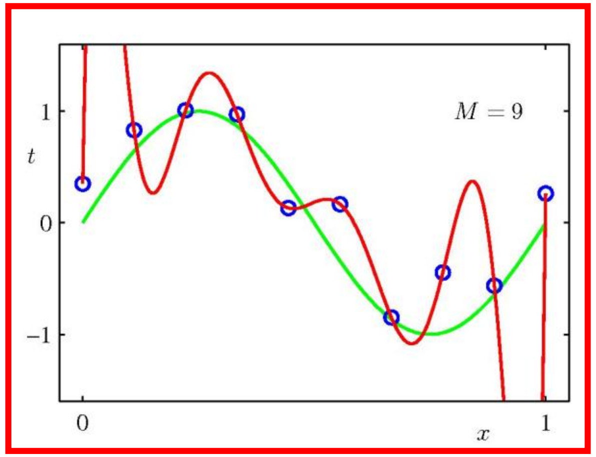


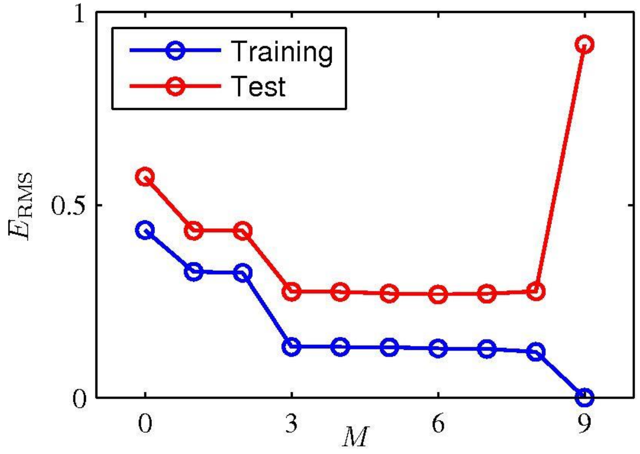
Courtesy of Dr. Hung-yi Lee

Overfitting

- Let's look at the estimated weights for various M

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

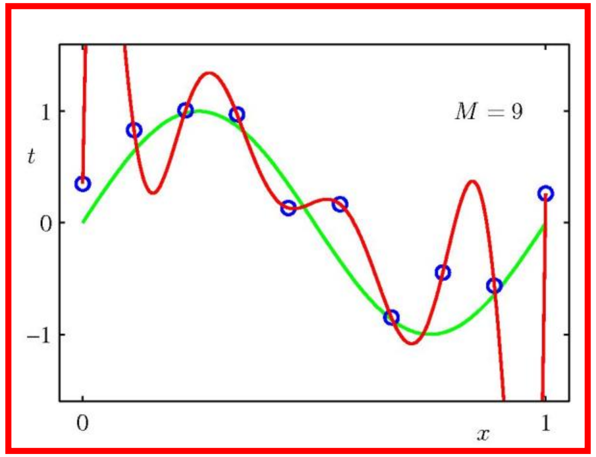


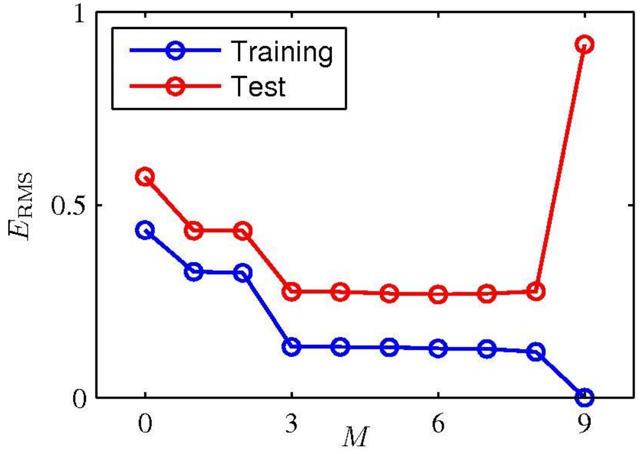


Overfitting

- Let's look at the estimated weights for various M

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43



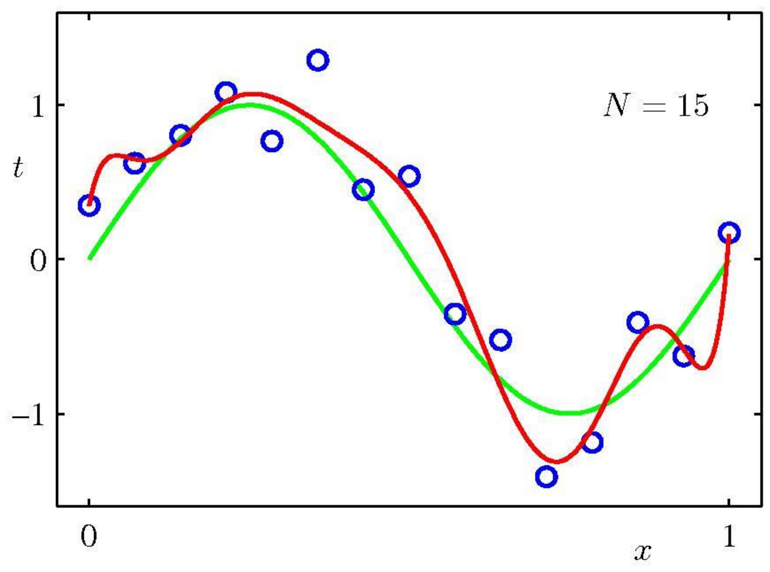


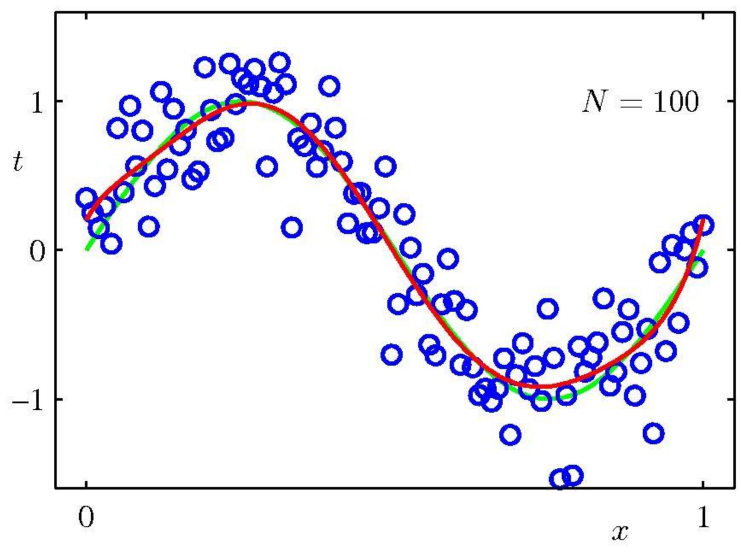
$$y(x,w) + large\ error \Leftarrow w_0 + \sum_{j=1}^{M} w_j(x + small\ noise)^j$$

The weights are becoming huge to compensate for the noise.
(a small noise on x will have much fluctuation on prediction y)

Overfitting

- Possible solutions
  - One workaround: Use more data





Overfitting

- Possible solutions
  - Second workaround: Use regularization (Very important!!!)

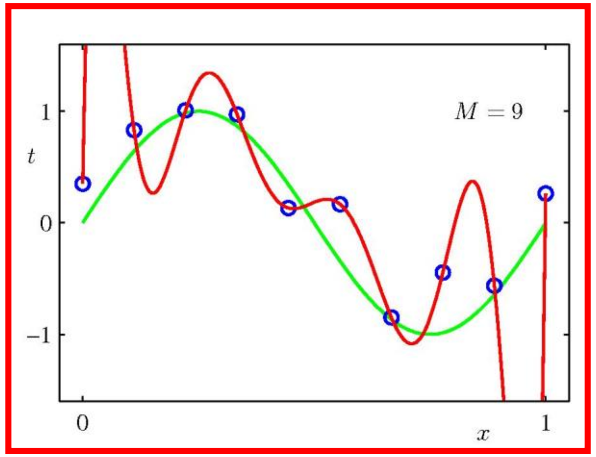
- Regularization
  - Redesign Loss function by introducing regularization term

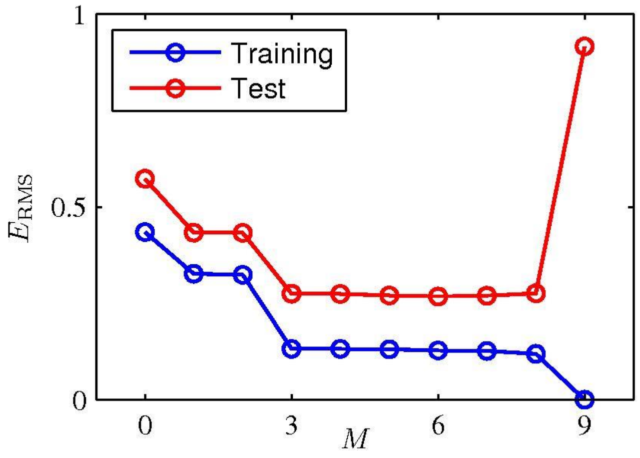
$$\ell(w) = \frac{1}{2N}\left[\sum_{i=1}^{N}[t^{(i)} - y\left(x^{(i)}\right)]^2 + \lambda \sum_{i=1}^{M} w_i^2\right]$$

Overfitting

- Let's look at the estimated weights for various M

	$M = 0$	$M = 1$	$M = 6$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43





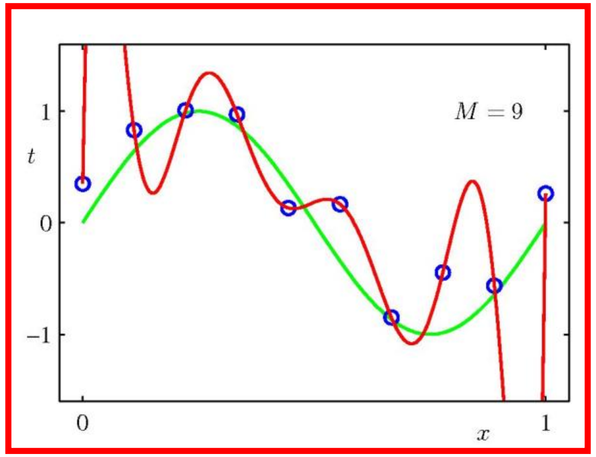
Overfitting

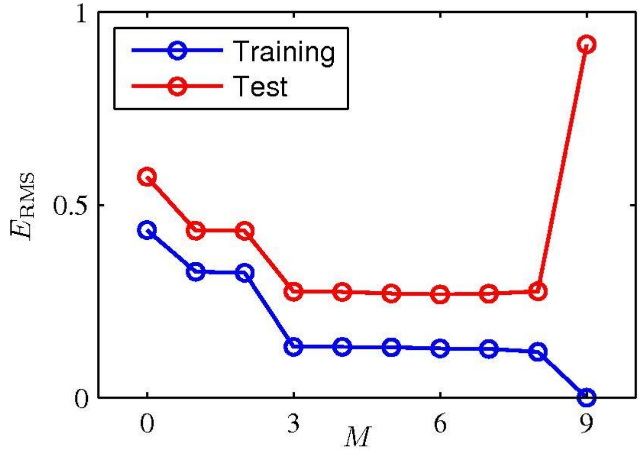
- Let's look at the estimated weights for various M

	$M=0$	$M=1$	$M=6$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

$$y(x,w) + large\ error \Leftarrow w_0 + \sum_{j=1}^{M} w_j(x + small\ noise)^j$$

The weights are becoming huge to compensate for the noise.
(a small noise on x will have much fluctuation on prediction y)





Overfitting

- One way of dealing with this is to encourage the weights to be small. This is called regularization.

- **Standard approach**
  - New loss function

$$\ell(w) = \frac{1}{2N}\left[\sum_{i=1}^{N}[t^{(i)} - y\big(x^{(i)}\big)]^2 + \lambda \sum_{i=1}^{M} w_i^2\right]$$

- The penalty on the squared weights is known as ridge regression (Statistics)

Overfitting

- When w_i's are small, prediction y will be not sensitive to small change of x
- Smooth functions are preferred.

- One way of dealing with this is to encourage the weights to be small. This is called regularization.

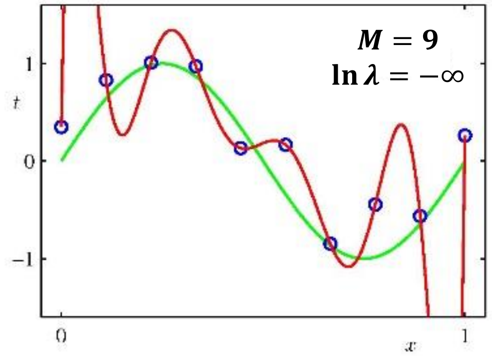
- **Standard approach**
  - New loss function

$$\ell(w) = \frac{1}{2N}\left[\sum_{i=1}^{N}[t^{(i)} - y(x^{(i)})]^2 + \lambda \sum_{i=1}^{M} w_i^2\right]$$

- The penalty on the squared weights is known as ridge regression (Statistics)

Overfitting

- When w_i's are small, prediction y will be not sensitive to small change of x
- Smooth functions are preferred.



- One way of dealing with this is to encourage the weights to be small. This is called regularization.

- **Standard approach**
  - New loss function

$$\ell(w) = \frac{1}{2N}\left[\sum_{i=1}^{N}[t^{(i)} - y(x^{(i)})]^2 + \lambda \sum_{i=1}^{M} w_i^2\right]$$

- The penalty on the squared weights is known as ridge regression (Statistics)

Overfitting

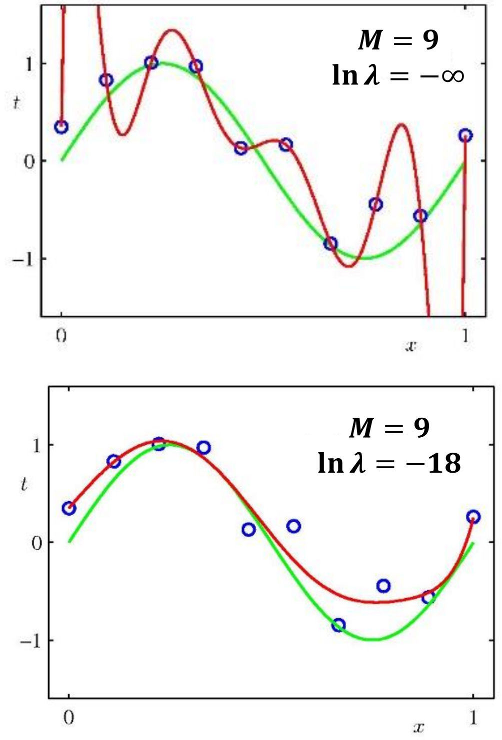
- When w_i's are small, prediction y will be not sensitive to small change of x
- Smooth functions are preferred.

- One way of dealing with this is to encourage the weights to be small. This is called regularization.

- **Standard approach**
  - New loss function

$$\ell(w) = \frac{1}{2N}\left[\sum_{i=1}^{N}[t^{(i)} - y(x^{(i)})]^2 + \lambda \sum_{i=1}^{M} w_i^2\right]$$

- The penalty on the squared weights is known as ridge regression (Statistics)



Overfitting

- When w_i's are small, prediction y will be not sensitive to small change of x
- Smooth functions are preferred.

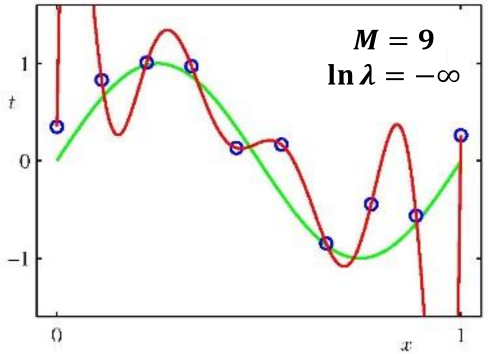
- One way of dealing with this is to encourage the weights to be small. This is called regularization.

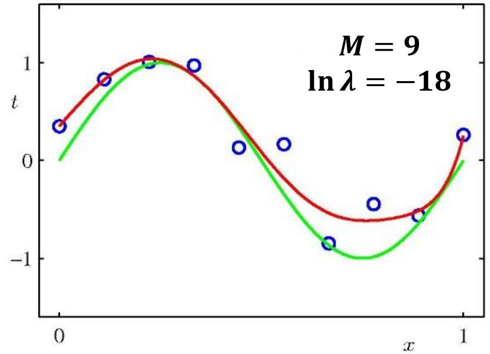
- **Standard approach**
  - New loss function

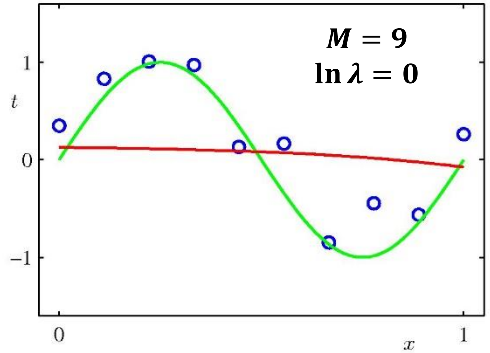
$$\ell(w) = \frac{1}{2N}\left[\sum_{i=1}^{N}[t^{(i)} - y(x^{(i)})]^2 + \lambda \sum_{i=1}^{M} w_i^2\right]$$

- The penalty on the squared weights is known as ridge regression (Statistics)

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01







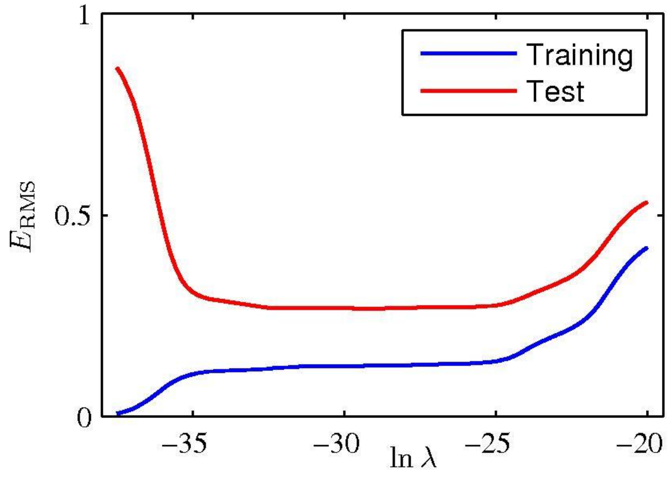
Overfitting

- One way of dealing with this is to encourage the weights to be small. This is called regularization.

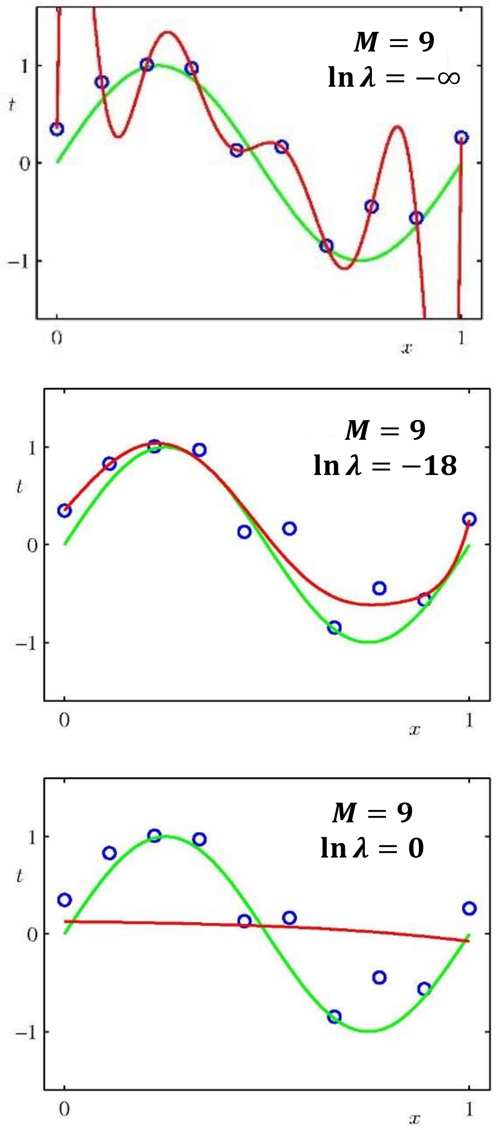
- **Standard approach**
  - New loss function

$$\ell(w) = \frac{1}{2N}\left[\sum_{i=1}^{N}[t^{(i)} - y(x^{(i)})]^2 + \lambda \sum_{i=1}^{M} w_i^2\right]$$

- The penalty on the squared weights is known as ridge regression (Statistics)



However, choose value λ **carefully**
(we prefer smooth function, but don't be too smooth)



Key Concepts

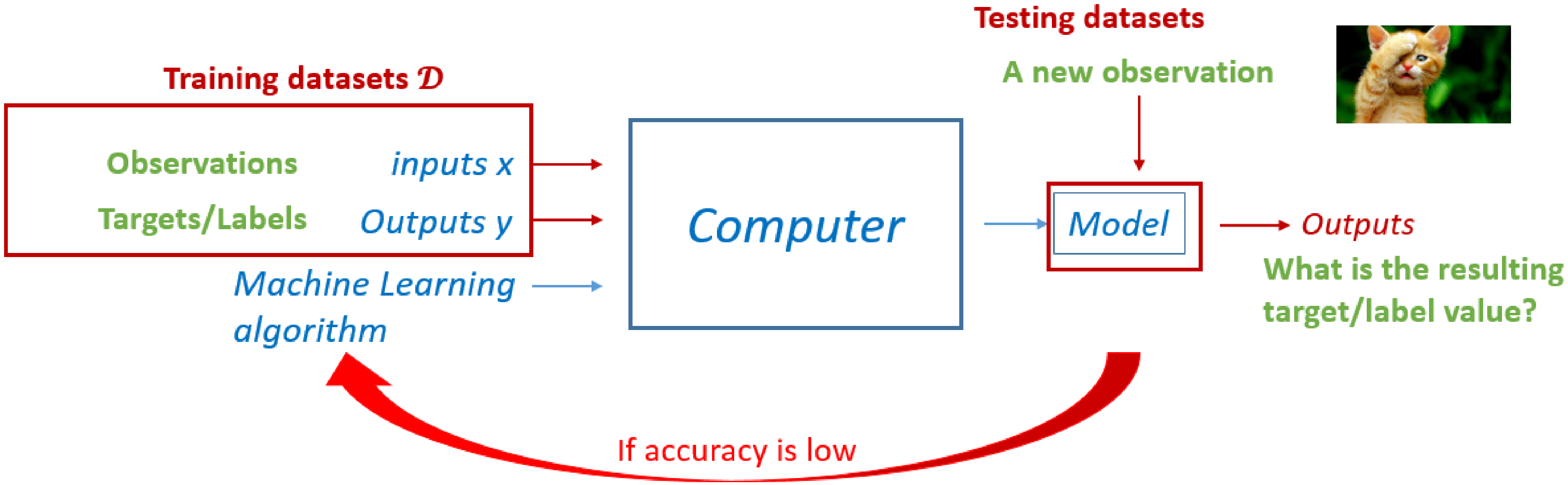
Key concepts in Supervised Learning, not just for linear models

- To find a good model
  - Loss function **(measure error, or judge the fit)**
  - Optimization **(how to find a good fit)**
  - Generalization **(fit to unseen test data)**
  - Regularization **(avoid overfitting)**

- Prediction error: validation

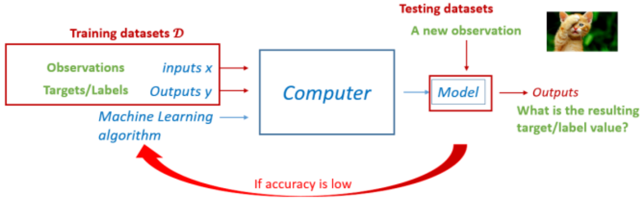
What we really care about is the prediction error on new data

Validation


Testing datasets
A new observation
Training datasets $\mathcal{D}$
Observations
inputs x
Targets/Labels
Outputs y
Computer
Model
Outputs
What is the resulting target/label value?
Machine Learning algorithm
If accuracy is low

Validation

- What you should NOT do in Machine learning



Validation

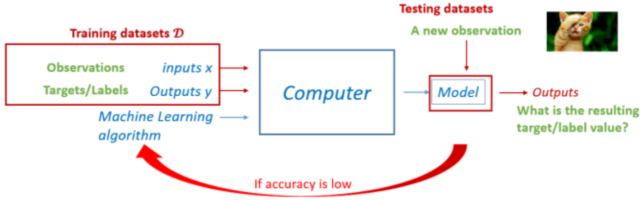
- What you should NOT do in Machine learning

(Your) Training dataset

(Your) Testing dataset

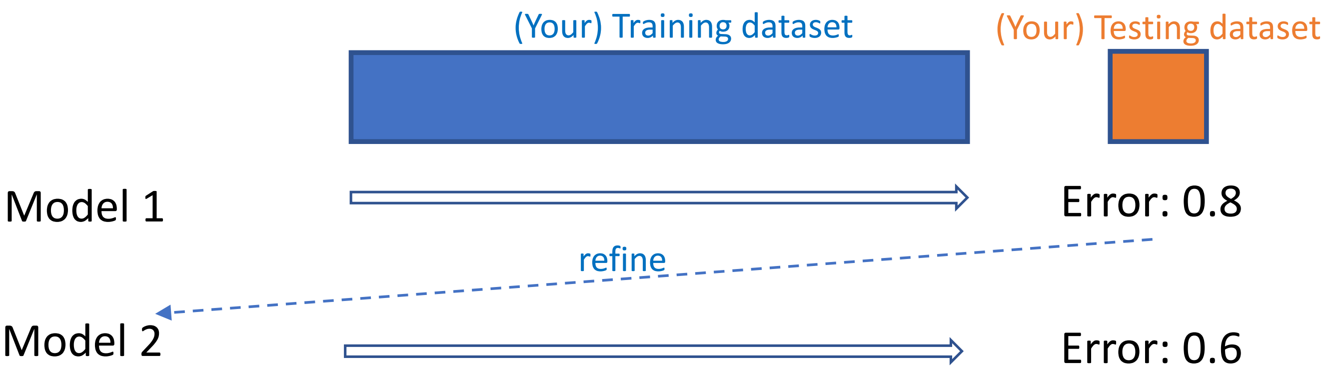
Model 1

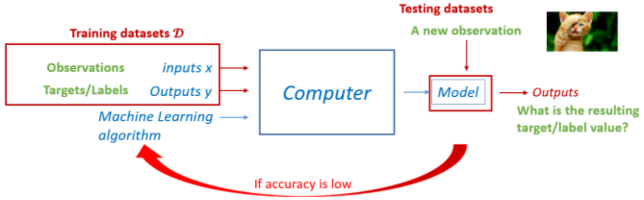
Error: 0.8



Validation

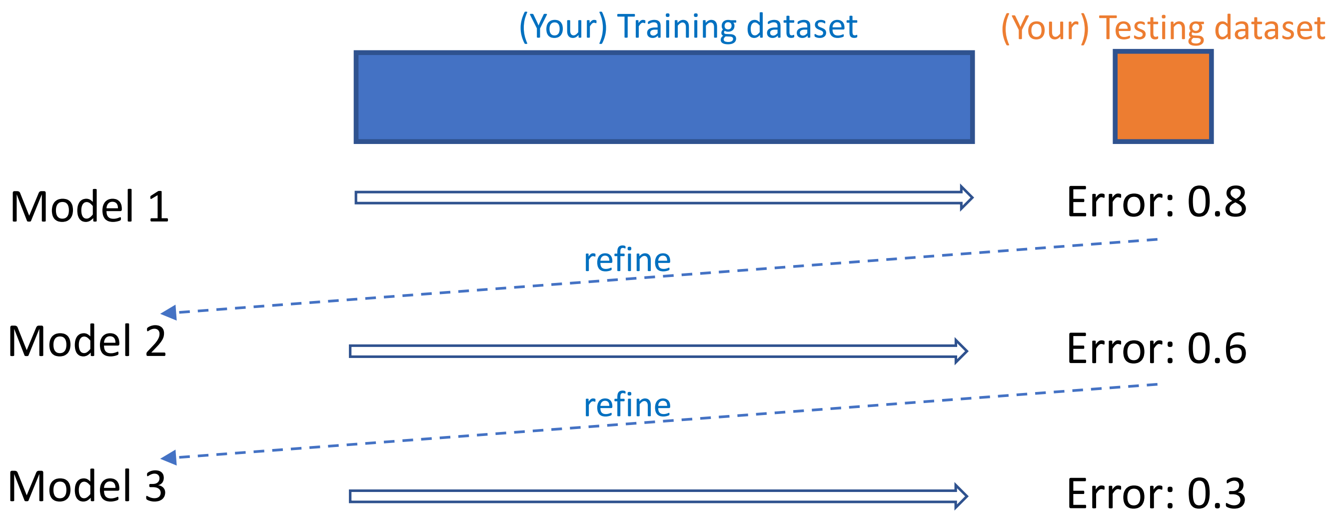
- What you should NOT do in Machine learning

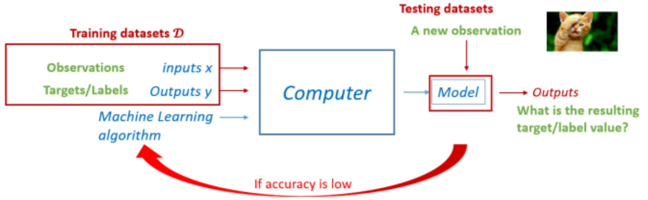




Validation

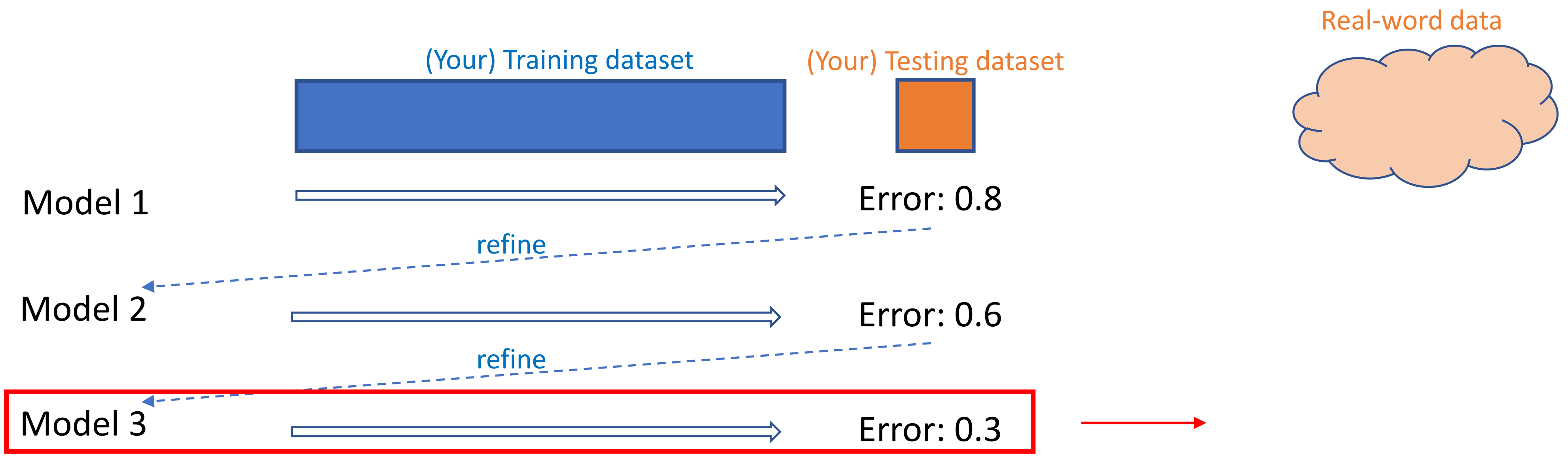
- What you should NOT do in Machine learning

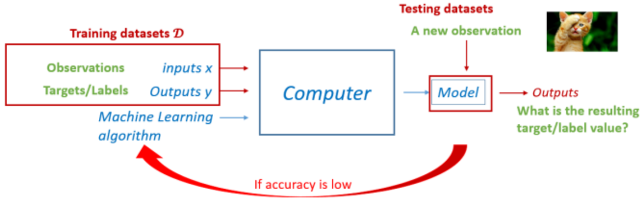




Validation

- What you should NOT do in Machine learning





Validation

- What you should NOT do in Machine learning

Real-word data

(Your) Training dataset

(Your) Testing dataset

Model 1 — Error: 0.8

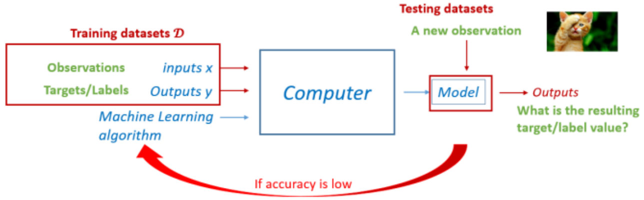
refine

Model 2 — Error: 0.6

refine

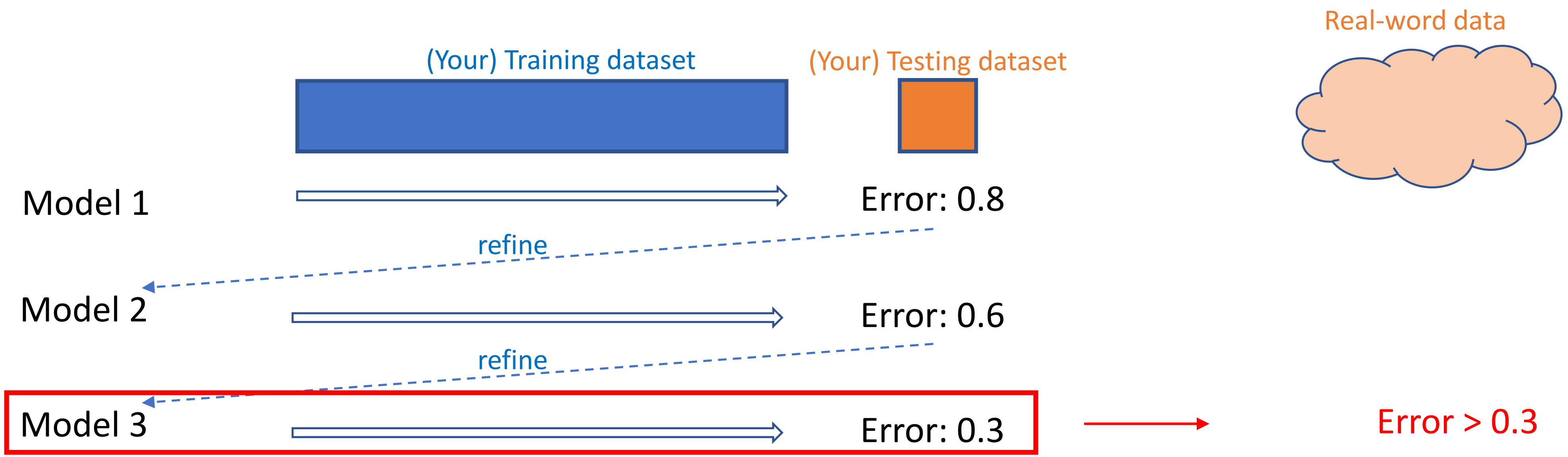
Model 3 — Error: 0.3

Error > 0.3



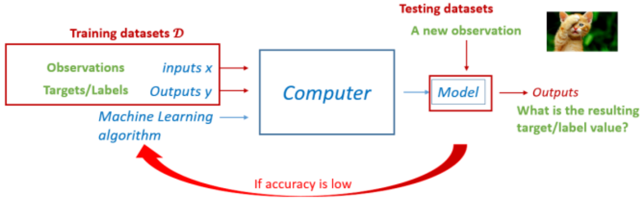
Validation

- What you should NOT do in Machine learning



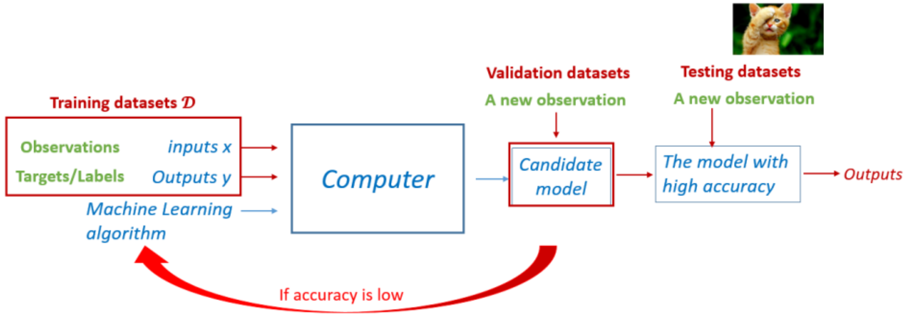
Testing data is usually used to judge the goodness of a fully-trained model. But here, it is used to refine (tune) the model.

- The testing data here is not independent of training process and may not reflect the prediction on real world data.



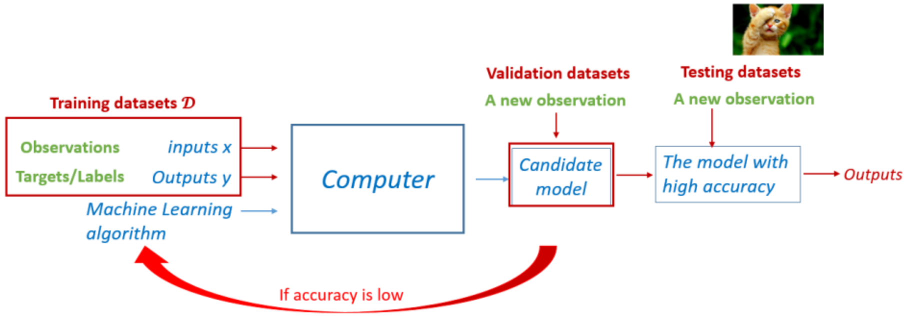
Validation

- What you can do in Machine learning



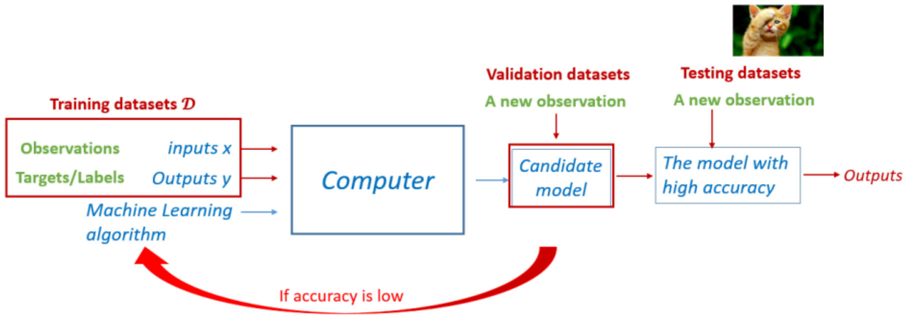
Validation

- What you can do in Machine learning



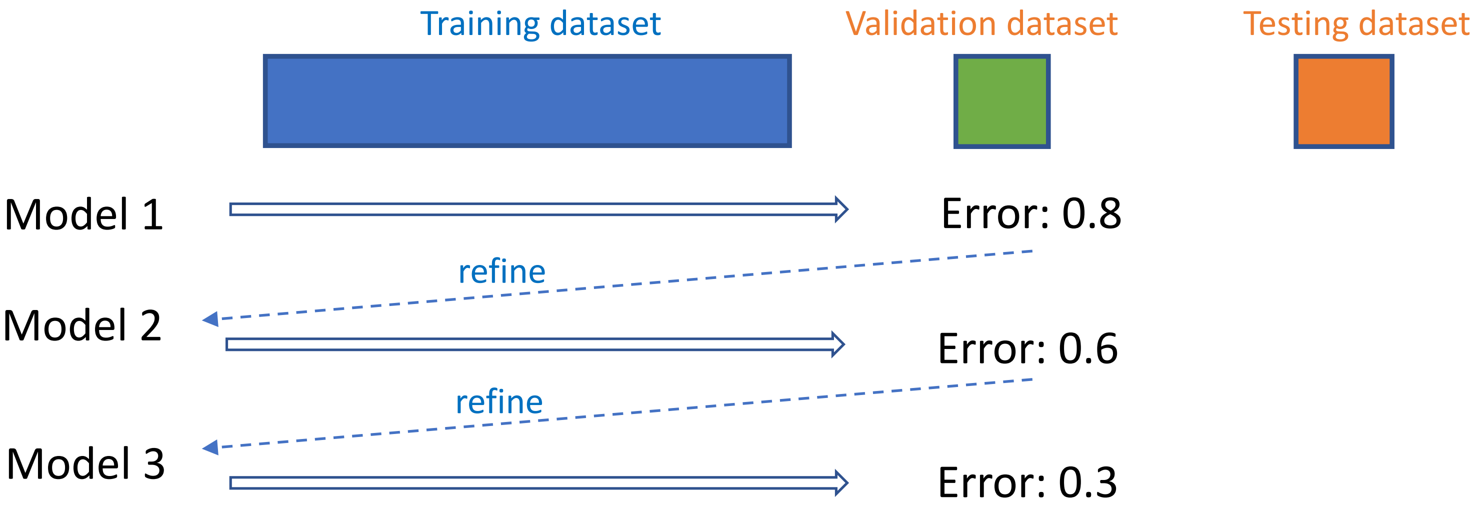
Validation

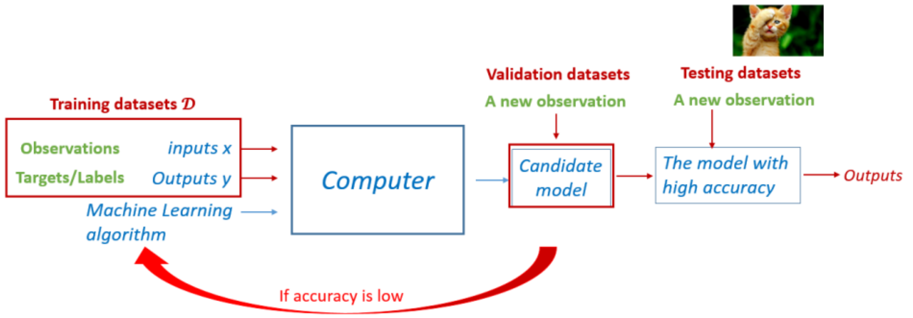
- What you can do in Machine learning



Validation

- What you can do in Machine learning





Validation

- What you can do in Machine learning

Training dataset

Validation dataset

Testing dataset

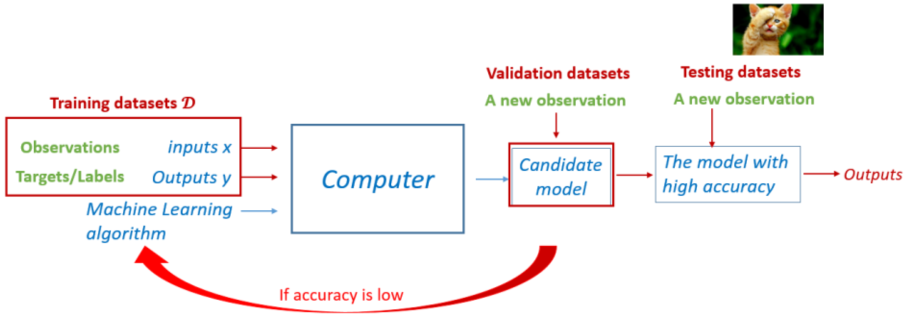
Model 1 → Error: 0.8

refine

Model 2 → Error: 0.6

refine

Model 2 → Error: 0.3 → Error:0.34



Validation

- What you can do in Machine learning

Training dataset | Validation dataset | Testing dataset | Real-word data

Model 1 → Error: 0.8

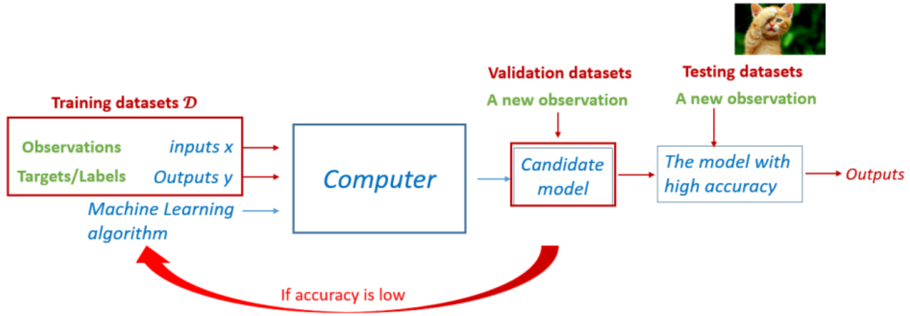
refine

Model 2 → Error: 0.6

refine

Model 2 → Error: 0.3 → Error:0.34

This is your announced accuracy

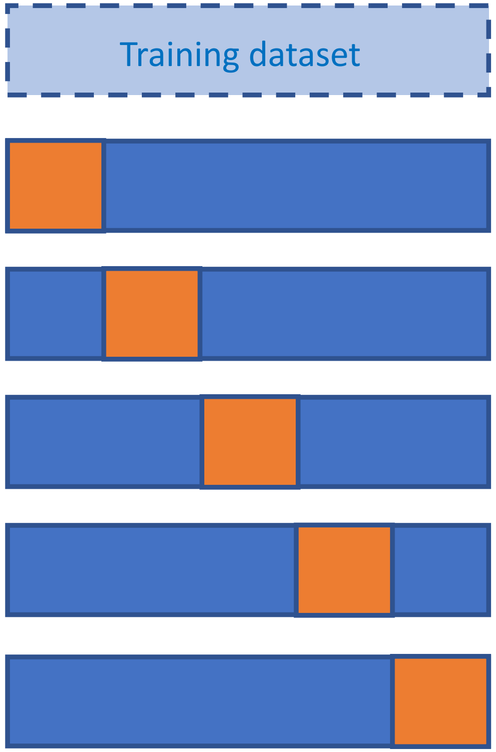


Validation

- In many applications, the supply of data for training and testing may be limited. We can use Cross-Validation (CV).

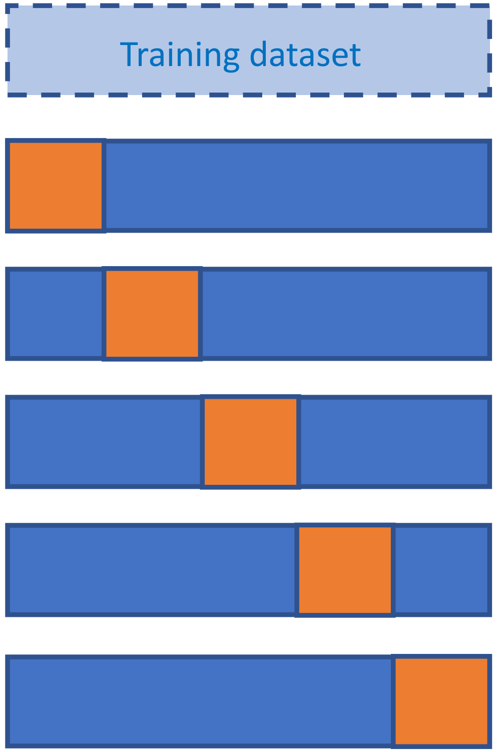
Validation

- In many applications, the supply of data for training and testing may be limited. We can use Cross-Validation (CV).



Validation

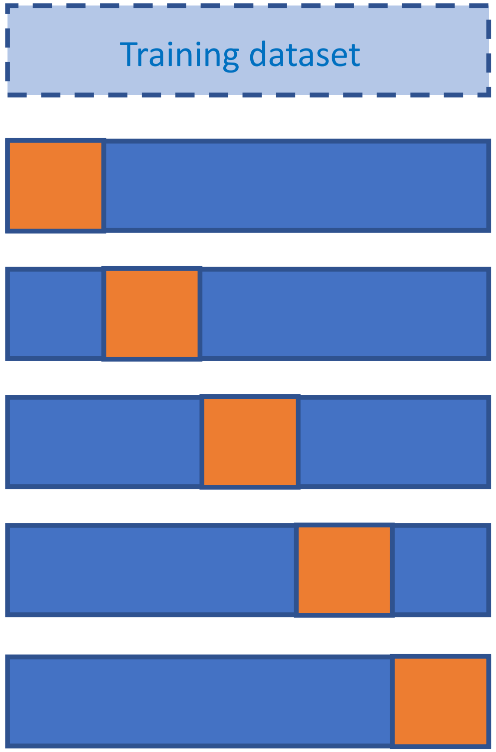
- In many applications, the supply of data for training and testing may be limited. We can use Cross-Validation (CV).



Model 1	Model 2	Model 3
Error: 0.8	Error: 0.7	Error: 0.8
Error: 0.5	Error: 0.6	Error: 0.4
Error: 0.7	Error: 0.3	Error: 0.7
Error: 0.3	Error: 0.9	Error: 0.5
Error: 0.9	Error: 0.4	Error: 0.9

Validation

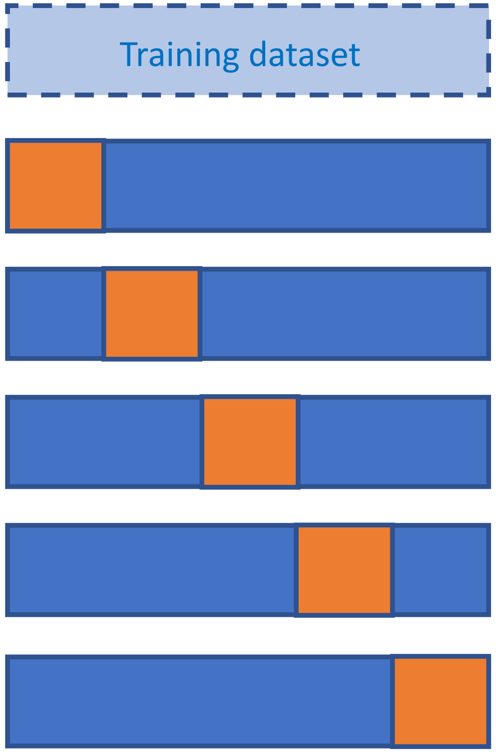
- In many applications, the supply of data for training and testing may be limited. We can use Cross-Validation (CV).



Model 1	Model 2	Model 3
Error: 0.8	Error: 0.7	Error: 0.8
Error: 0.5	Error: 0.6	Error: 0.4
Error: 0.7	Error: 0.3	Error: 0.7
Error: 0.3	Error: 0.9	Error: 0.5
Error: 0.9	Error: 0.4	Error: 0.9
Average: 0.64	Average: 0.58	Average: 0.66

Validation

- In many applications, the supply of data for training and testing may be limited. We can use Cross-Validation (CV).



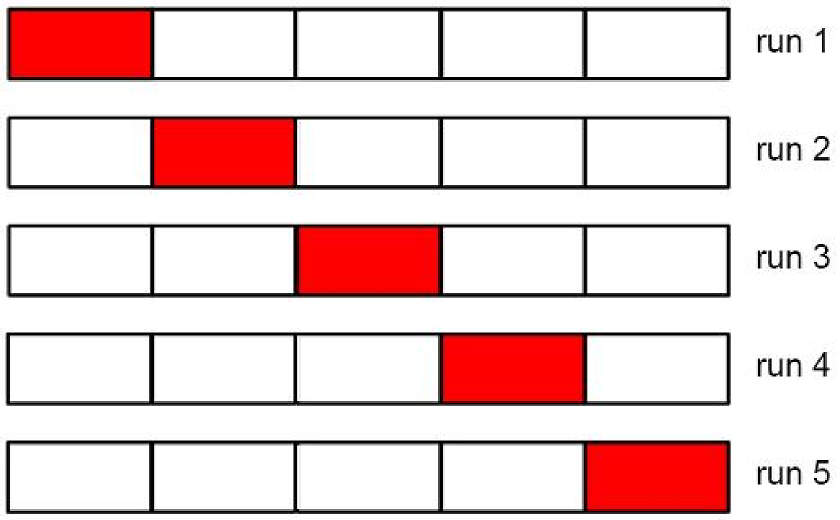
Model 1	Model 2	Model 3
Error: 0.8	Error: 0.7	Error: 0.8
Error: 0.5	Error: 0.6	Error: 0.4
Error: 0.7	Error: 0.3	Error: 0.7
Error: 0.3	Error: 0.9	Error: 0.5
Error: 0.9	Error: 0.4	Error: 0.9
Average: 0.64	Average: 0.58	Average: 0.66

Validation

- Cross Validation
  - We split the training data into K **folds**; then, for each fold k ∈ {1, . . . ,K}, we train on all the folds but the k'th, and test on the k'th, in a round-robin fashion.
  - We then compute the error averaged over all the folds, and use this as a proxy for the test error.

  (Note that each point gets predicted only once, although it will be used for training K−1 times.)

  - It is common to use K = 5; this is called 5-fold CV.



Reading

- LFD: Chapter 4;
- Pattern Recognition and Machine Learning: 1.1 (optional)