CSCI567 Machine Learning (Fall 2017)

Prof. Fei Sha

U of Southern California

Lecture on Aug. 31, 2017

Outline

1. Administration
2. Review of Last Lecture
3. Nonlinear basis functions
4. Basic ideas of overcome overfitting
5. Bias/Variance Analysis

Outline

1. Administration
2. Review of Last Lecture
3. Nonlinear basis functions
4. Basic ideas of overcome overfitting
5. Bias/Variance Analysis

Administrative stuff

- This course will be offered in Spring 2018 and will be taught by another faculty member.
- TA office hours have been announced.
- We have been starting to enroll students into Piazza and Github.

Outline

1 Administration

2 Review of Last Lecture

3 Nonlinear basis functions

4 Basic ideas of overcome overfitting

5 Bias/Variance Analysis

Linear regression

Setup

- Input: $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$ (covariates, predictors, features, etc)
- Output: $y \in \mathbb{R}$ (responses, targets, outcomes, outputs, etc)
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$
  We will use x_{nd} representing the dth dimension of the nth sample $\boldsymbol{x}_n$
- Model: $f : \boldsymbol{x} \to y$, with $f(\boldsymbol{x}) = w_0 + \sum_d w_d x_d = w_0 + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$, with ^{T} standing for vector transpose.

Finding the best model parameters by minimizing prediction errors

Design matrix and target vector

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \boldsymbol{x}_2^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_{\mathsf{N}}^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{\mathsf{N} \times D}, \quad \tilde{\boldsymbol{X}} = \begin{pmatrix} 1 & \boldsymbol{X} \end{pmatrix} \in \mathbb{R}^{\mathsf{N} \times (D+1)}, \quad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_{\mathsf{N}} \end{pmatrix}$$

Residual sum squares in matrix form

$$RSS(\tilde{\boldsymbol{w}}) = \left\{ \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}} - 2 \left(\tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y} \right)^{\mathrm{T}} \tilde{\boldsymbol{w}} \right\} + \mathsf{const}$$

Optimal solution

Normal equation

$$\tilde{\boldsymbol{w}}^{LMS} = \left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\right)^{-1}\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$$

Outline

1 Administration

2 Review of Last Lecture

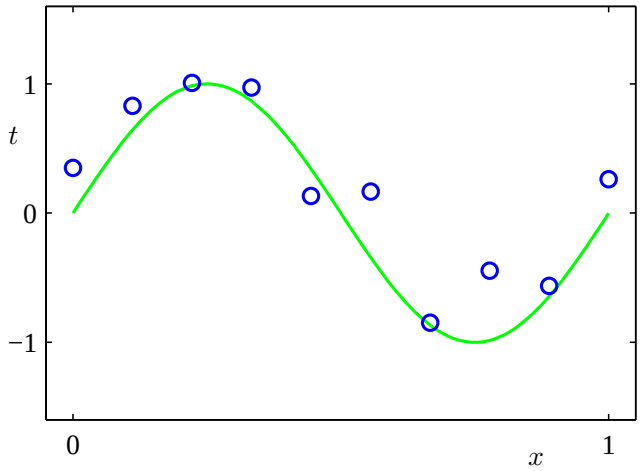
3 Nonlinear basis functions

4 Basic ideas of overcome overfitting

5 Bias/Variance Analysis

What if data do not fits to a line

Example of nonlinear regression



General nonlinear basis functions

We can use a nonlinear mapping

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^D \rightarrow \boldsymbol{z} \in \mathbb{R}^M$$

where M is the dimensionality of the new feature/input $\boldsymbol{z}$ (or $\boldsymbol{\phi}(\boldsymbol{x})$). Note that M could be either greater than D or less than or the same.

Note that $\boldsymbol{z}$ is a vector

$$v_1 = \phi_1(\boldsymbol{x}), v_2 = \phi_2(\boldsymbol{x}), \cdots, v_M = \phi_M(\boldsymbol{x})$$

Nonlinear regression thru nonlinearly transformed features

With the new features – we call them nonlinear basis functions – we can apply our learning techniques:

- linear methods: prediction is based on $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$
- more broadly, other methods: nearest neighbors, decision trees, etc

to minimize our errors on the transformed training data

Regression with nonlinear basis

Residual sum squares

$$\sum_n [\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) - y_n]^2$$

where $\boldsymbol{w} \in \mathbb{R}^M$, the same dimensionality as the transformed features $\boldsymbol{\phi}(\boldsymbol{x})$.

Regression with nonlinear basis

Residual sum squares

$$\sum_n [\boldsymbol{w}^{\mathrm{T}} \boldsymbol{\phi}(\boldsymbol{x}_n) - y_n]^2$$

where $\boldsymbol{w} \in \mathbb{R}^M$, the same dimensionality as the transformed features $\boldsymbol{\phi}(\boldsymbol{x})$.

The LMS solution can be formulated with the new design matrix

$$\boldsymbol{\Phi} = \begin{pmatrix} \boldsymbol{\phi}(\boldsymbol{x}_1)^{\mathrm{T}} \\ \boldsymbol{\phi}(\boldsymbol{x}_2)^{\mathrm{T}} \\ \vdots \\ \boldsymbol{\phi}(\boldsymbol{x}_N)^{\mathrm{T}} \end{pmatrix} \in \mathbb{R}^{N \times M}, \quad \boldsymbol{w}^{\mathrm{LMS}} = \left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi}\right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{y}$$

Example with regression

Polynomial basis functions

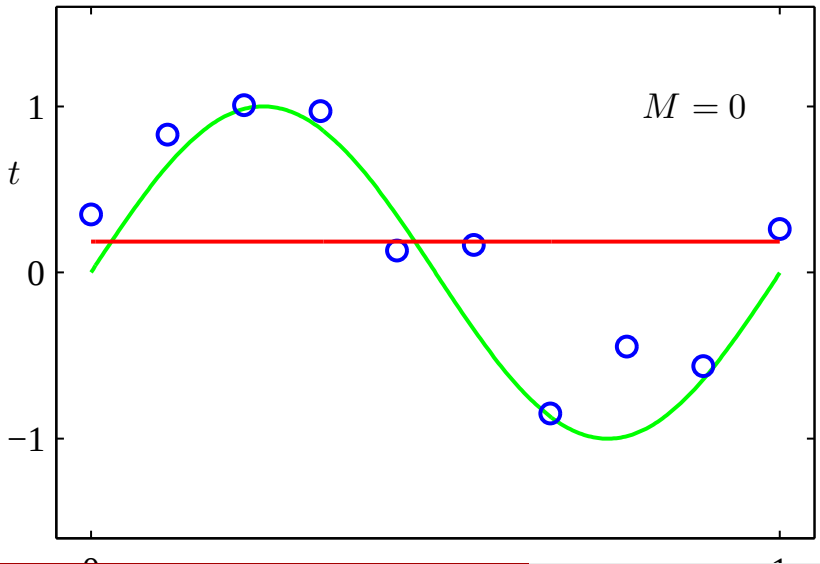
$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

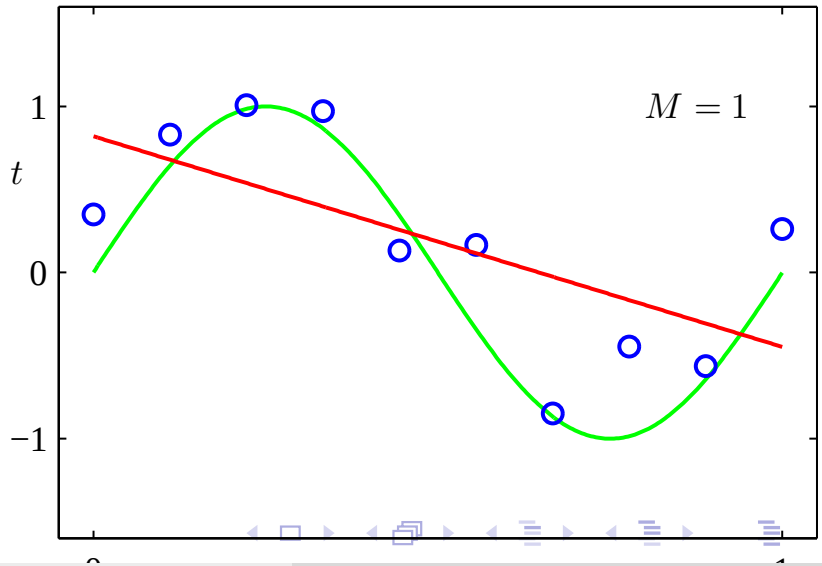
Example with regression

Polynomial basis functions

$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

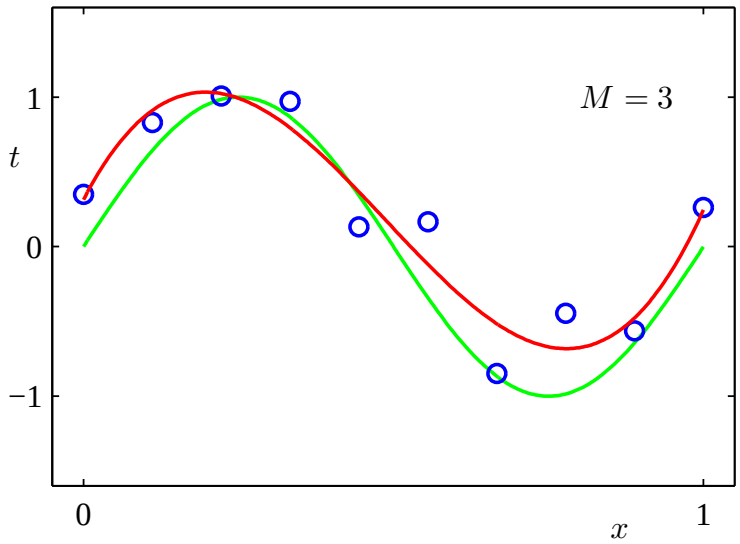
Fitting samples from a sine function: *underrfitting* as $f(x)$ is too simple





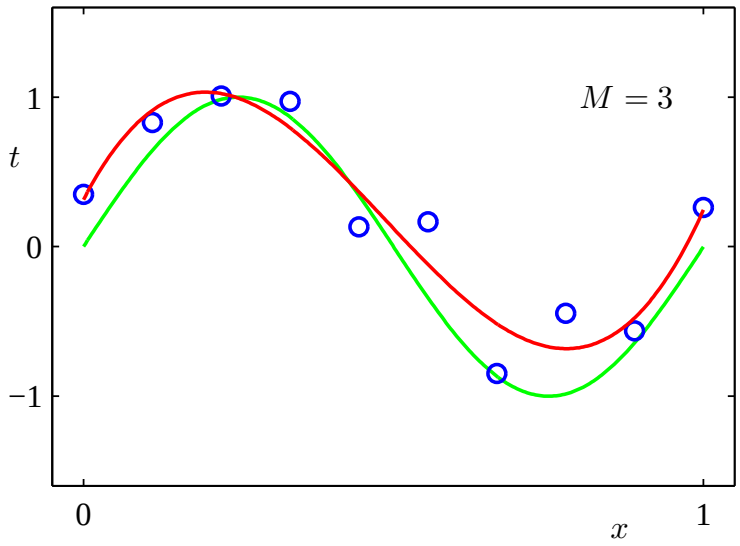
Adding more high-order basis

M=3

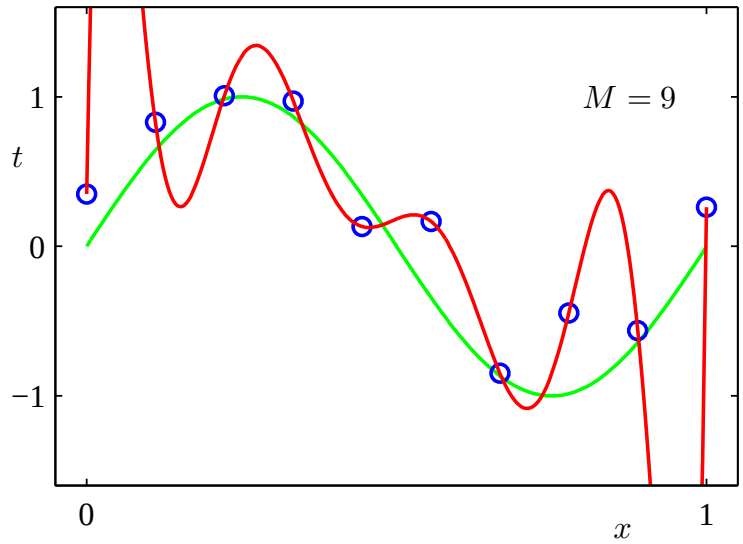


Adding more high-order basis

M=3



M=9: *overfitting*



Being too adaptive leads *better* results on the training data, but *not so great* on data that has not been seen!

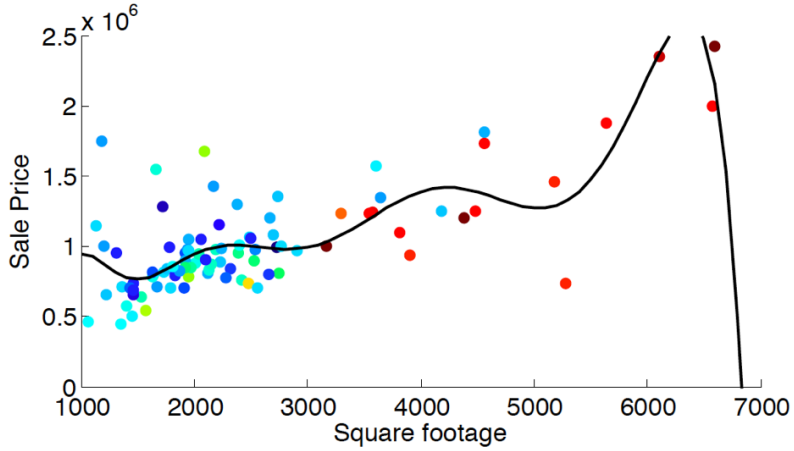
Overfiting

Parameters for higher-order polynomials are very large

	$M=0$	$M=1$	$M=3$	$M=9$
w_0	0.19	0.82	0.31	0.35
w_1		-1.27	7.99	232.37
w_2			-25.43	-5321.83
w_3			17.37	48568.31
w_4				-231639.30
w_5				640042.26
w_6				-1061800.52
w_7				1042400.18
w_8				-557682.99
w_9				125201.43

Overfitting can be quite disastrous

Fitting the housing price data with $M = 3$

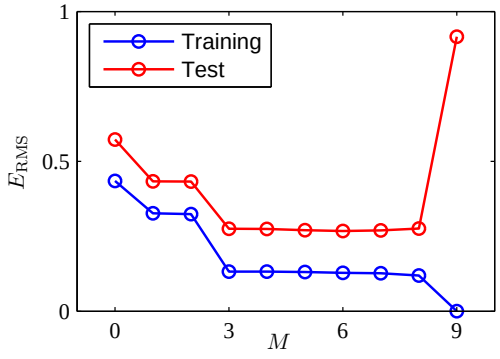


Note that the price would goes to zero (or negative) if you buy bigger ones!
This is called poor generalization/overfitting.

Detecting overfitting

Plot model complexity versus objective function

As model becomes more complex, performance on training keeps improving while on test data improve first and deteriorate later.

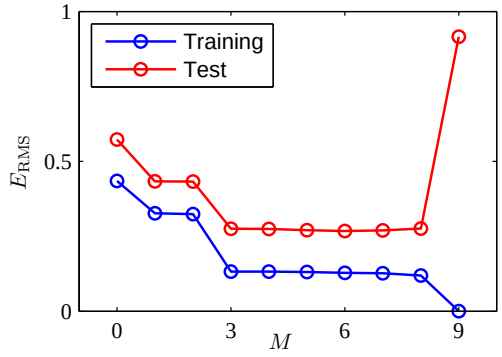


- Horizontal axis: *measure of model complexity*
  In this example, we use the maximum order of the polynomial basis functions.

Detecting overfitting

Plot model complexity versus objective function

As model becomes more complex, performance on training keeps improving while on test data improve first and deteriorate later.



- Horizontal axis: *measure of model complexity*
  In this example, we use the maximum order of the polynomial basis functions.
- Vertical axis:
  1. For regression, the vertical axis would be RSS or RMS (squared root of RSS)
  2. For classification, the vertical axis would be classification error rate or cross-entropy error function (more on the latter later)

Outline

1 Administration

2 Review of Last Lecture

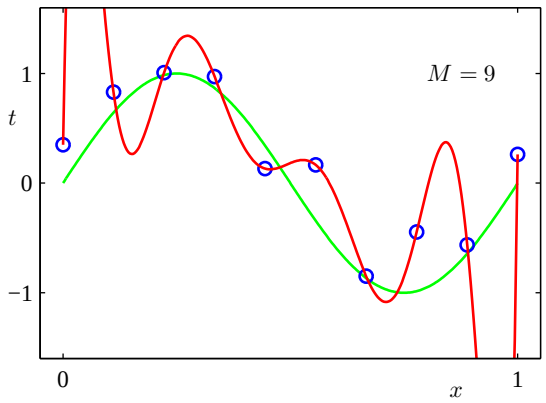
3 Nonlinear basis functions

4 Basic ideas of overcome overfitting
- Use more training data
- Regularization methods
- Cross-validation

5 Bias/Variance Analysis

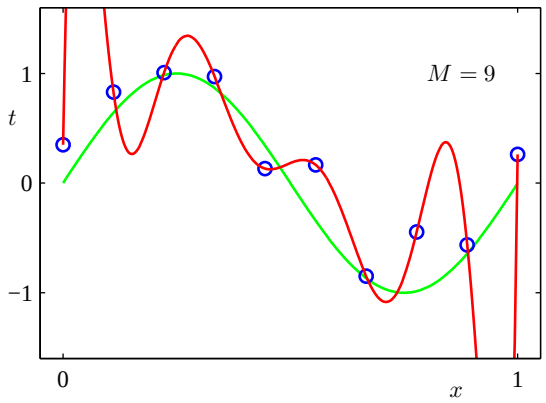
Use more training data to prevent over fitting

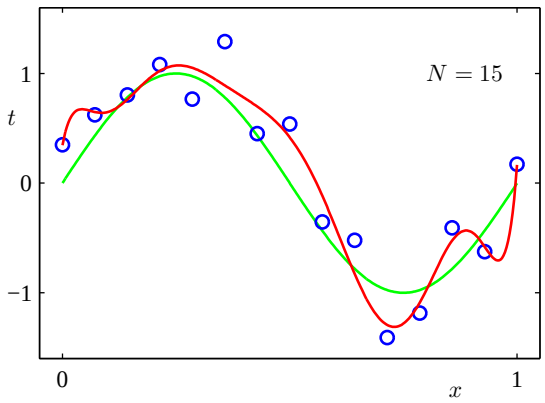
The more, the merrier



Use more training data to prevent over fitting

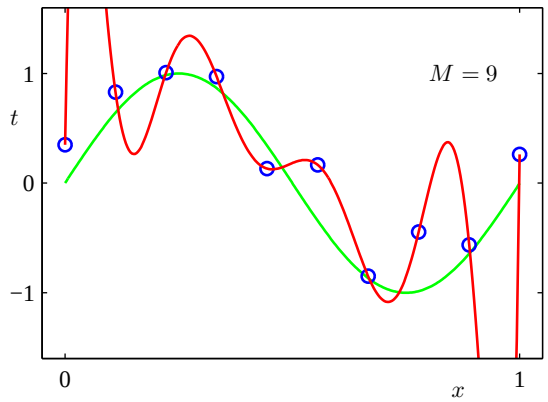
The more, the merrier

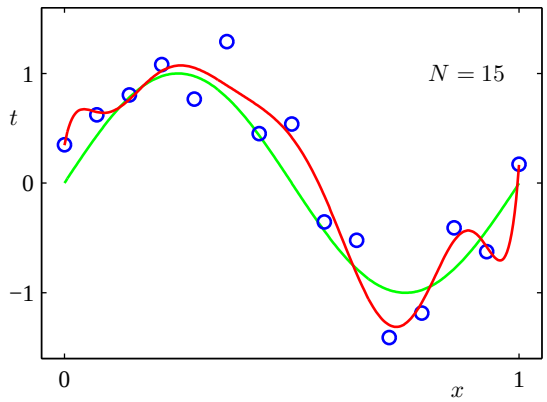


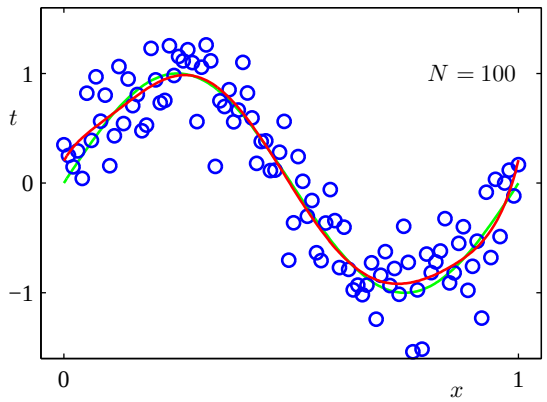


Use more training data to prevent over fitting

The more, the merrier







What if we do not have a lot of data?

What is a simple model?

For a linear model for regression

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$

what do we mean by *being simple*?

What is a simple model?

For a linear model for regression

$$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$

what do we mean by *being simple*?

Intuition

- $w_1x_1 + w_2x_2$ is more complex than either w_1x_1 or w_2x_2.
- The smaller the w_i, the simpler the model is
- The simplest model probably has a lot of $w_i = 0$ or w_i is being small.

Example: fitting data with polynomials

Our regression model

$$y = \sum_{m=1}^{M} w_m x^m$$

Thus, smaller w_m will likely lead to a smaller oder of polynomial, thus potentially preventing overfitting.

How to make $\boldsymbol{w}$ small?

Regularized linear regression: a new error to minimize

$$\min \mathcal{E}(\boldsymbol{w}) = \min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2 + \lambda \|\boldsymbol{w}\|_2^2$$

where $\lambda > 0$. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

How to make $\boldsymbol{w}$ small?

Regularized linear regression: a new error to minimize

$$\min \mathcal{E}(\boldsymbol{w}) = \min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2 + \lambda \|\boldsymbol{w}\|_2^2$$

where $\lambda > 0$. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

Intuitions

- If $\lambda \to +\infty$, then $\boldsymbol{w}$ approaches $\mathbf{0}$.

How to make $\boldsymbol{w}$ small?

Regularized linear regression: a new error to minimize

$$\min \mathcal{E}(\boldsymbol{w}) = \min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2 + \lambda \|\boldsymbol{w}\|_2^2$$

where $\lambda > 0$. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

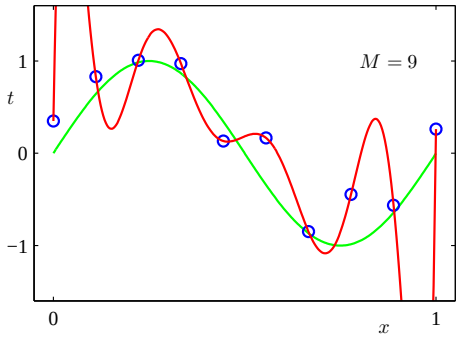
Intuitions

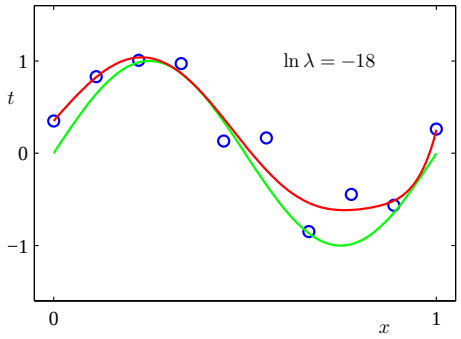
- If $\lambda \to +\infty$, then $\boldsymbol{w}$ approaches $\mathbf{0}$.
- If $\lambda \to 0$, then we approach the standard LMS solution

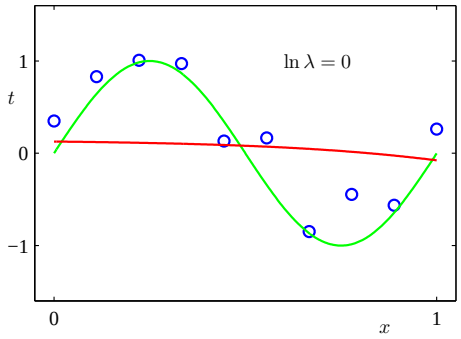
$$\arg\min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2$$

Overfitting in terms of λ

Overfitting is reduced from complex model to simpler one with the help of increasing regularizers

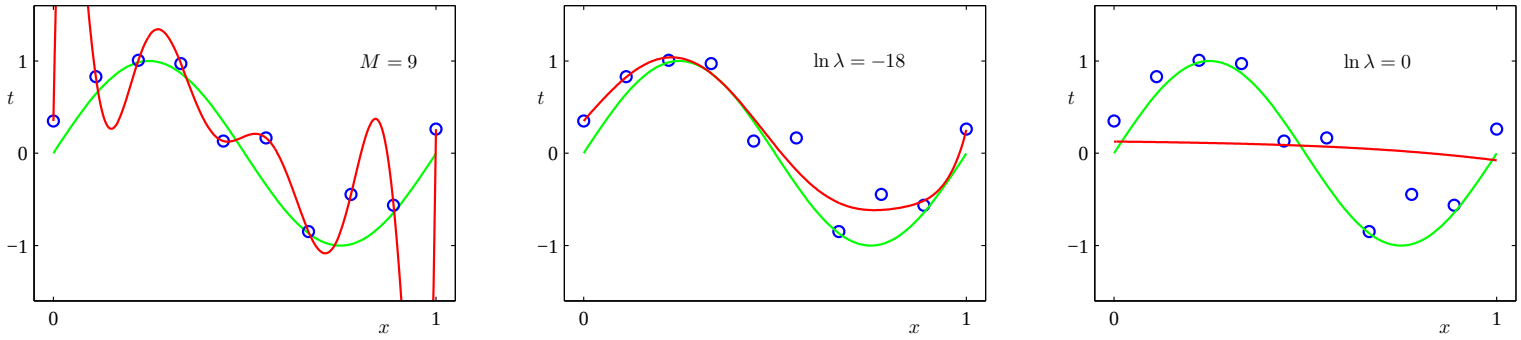




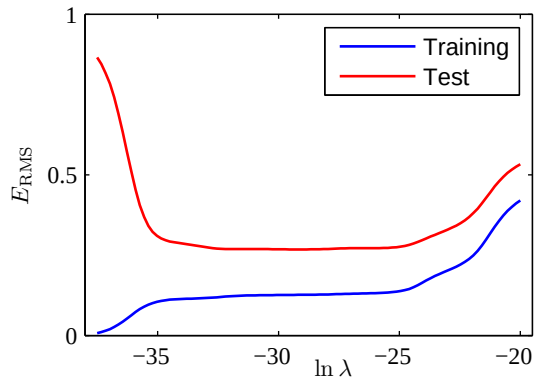


Overfitting in terms of λ

Overfitting is reduced from complex model to simpler one with the help of increasing regularizers



λ **vs. residual error** shows the difference of the model performance on training and testing dataset



Closed-form solution

For regularized linear regression (RLS): the solution changes very little (in form) from the LMS solution

$$\arg\min \sum_n (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - y_n)^2 + \lambda \|\boldsymbol{w}\|_2^2 \Rightarrow \boldsymbol{w}^{\mathrm{RLS}} = \left(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{I}\right)^{-1} \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y}$$

and reduces to the LMS solution when $\lambda = 0$, as expected.

Note that this form is the same as the proposed solution when the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is not vertible

The effect of λ

Large λ attenuating parameters towards 0

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

How to choose the right amount of regularization?

Can we tune λ on the training dataset?

No: as this will set λ to zero, i.e., without regularization, defeating our intention to use it to control model complexity and to gain better generalization.

λ is thus a hyperparmeter. To tune it,

- We can use a development/holdout dataset independent of training and testing dataset.
- We can do leave-one-out (LOO)

The procedure is similar to choose K in the nearest neighbor classifiers.

How to choose the right amount of regularization?

Can we tune λ on the training dataset?

No: as this will set λ to zero, i.e., without regularization, defeating our intention to use it to control model complexity and to gain better generalization.

λ is thus a hyperparmeter. To tune it,

- We can use a development/holdout dataset independent of training and testing dataset.
- We can do leave-one-out (LOO)

The procedure is similar to choose K in the nearest neighbor classifiers.

For different λ, we get $\boldsymbol{w}^{\mathrm{RLS}}$ and evaluate the model on the development/holdout dataset (or, the samples being left in LOO).

How to choose the right amount of regularization?

Can we tune λ on the training dataset?

No: as this will set λ to zero, i.e., without regularization, defeating our intention to use it to control model complexity and to gain better generalization.

λ is thus a hyperparmeter. To tune it,

- We can use a development/holdout dataset independent of training and testing dataset.
- We can do leave-one-out (LOO)

The procedure is similar to choose K in the nearest neighbor classifiers.

For different λ, we get $\boldsymbol{w}^{\mathrm{RLS}}$ and evaluate the model on the development/holdout dataset (or, the samples being left in LOO).

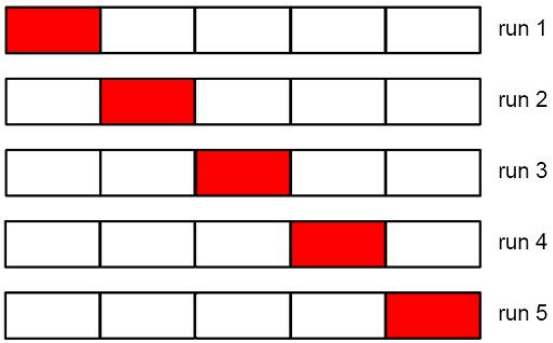
We then plot the curve λ versus prediction error (accuracy, classification error) and find the place that the performance on the holdout/LOO is the best.

Use cross-validation to choose λ

Procedure

- Randomly partition training data into K *disjoint* parts
  Normally, K is chosen to be 10, 5, etc.
- For each possible value of λ
  1. Use one part as holdout; use other $(K-1)$ parts as training
  2. Evaluate the model on the holdout
  3. Do this K times, and average the performance on the holdouts
- Choose the λ with the best performance

When $K = N$ (the number of training examples), this becomes LOO.



Outline

1 Administration

2 Review of Last Lecture

3 Nonlinear basis functions

4 Basic ideas of overcome overfitting

5 Bias/Variance Analysis

Basic and important machine learning concepts

Supervised learning

We aim to build a function $f(\boldsymbol{x})$ to predict the true value y associated with $\boldsymbol{x}$. If we make a mistake, we incur a *loss*

$$L(f(\boldsymbol{x}), y)$$

Basic and important machine learning concepts

Supervised learning

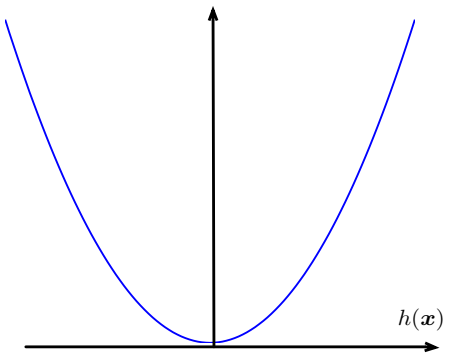
We aim to build a function $f(\boldsymbol{x})$ to predict the true value y associated with $\boldsymbol{x}$. If we make a mistake, we incur a *loss*

$$L(f(\boldsymbol{x}), y)$$

Example: quadratic loss function for regression when y is continuous

$$L(f(\boldsymbol{x}), y) = [f(\boldsymbol{x}) - y]^2$$

Ex: when $y = 0$

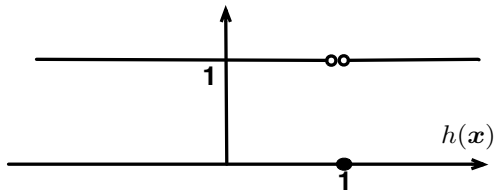


Other types of loss functions

For classification: $0/1$ loss

$$L(f(\boldsymbol{x}), y) = \mathbb{I}[f(\boldsymbol{x}) \neq y]$$

Ex: when $y = 1$



Measure how good our predictor is

Risk: assume we know the true distribution of data $p(\boldsymbol{x}, y)$, the *risk* is

$$R[f(\boldsymbol{x})] = \mathbb{E}_{(\boldsymbol{x},y)\tilde{p}(\boldsymbol{x},y)} L(f(\boldsymbol{x}), y) = \int L(f(\boldsymbol{x}), y) p(\boldsymbol{x}, y) d\boldsymbol{x} d\, y$$

Measure how good our predictor is

Risk: assume we know the true distribution of data $p(\boldsymbol{x}, y)$, the *risk* is

$$R[f(\boldsymbol{x})] = \mathbb{E}_{(\boldsymbol{x},y)\tilde{p}(\boldsymbol{x},y)} L(f(\boldsymbol{x}), y) = \int L(f(\boldsymbol{x}), y) p(\boldsymbol{x}, y) d\boldsymbol{x} d\, y$$

However, we cannot compute $R[f(\boldsymbol{x})]$, so we use *empirical risk*, given a training dataset $\mathcal{D}$

$$R^{\mathrm{EMP}}[f(\boldsymbol{x})] = \frac{1}{N} \sum_n L(f(\boldsymbol{x}_n), y_n)$$

Measure how good our predictor is

Risk: assume we know the true distribution of data $p(\boldsymbol{x}, y)$, the *risk* is

$$R[f(\boldsymbol{x})] = \mathbb{E}_{(\boldsymbol{x},y)\tilde{p}(\boldsymbol{x},y)} L(f(\boldsymbol{x}), y) = \int L(f(\boldsymbol{x}), y) p(\boldsymbol{x}, y) d\boldsymbol{x} d\, y$$

However, we cannot compute $R[f(\boldsymbol{x})]$, so we use *empirical risk*, given a training dataset $\mathcal{D}$

$$R^{\text{EMP}}[f(\boldsymbol{x})] = \frac{1}{N} \sum_n L(f(\boldsymbol{x}_n), y_n)$$

Intuitively, as $N \to +\infty$,

$$R^{\text{EMP}}[f(\boldsymbol{x})] \to R[f(\boldsymbol{x})]$$

How this relates to what we have learnt?

So far, we have been doing empirical risk minimization (ERM)

- For linear regression, $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$, and we use squared loss, which leads to *residual sum squares*

How this relates to what we have learnt?

So far, we have been doing empirical risk minimization (ERM)

- For linear regression, $f(\boldsymbol{x}) = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$, and we use squared loss, which leads to *residual sum squares*

ERM might be problematic

- If $f(\boldsymbol{x})$ is complicated enough,

$$R^{\mathrm{EMP}}[f(\boldsymbol{x})] \to 0$$

- But then $f(\boldsymbol{x})$ is unlikely to do well in predicting things out of the training dataset $\mathcal{D}$
  This is called *poor generalization* or *overfitting*. We have seen how to fix that problem.

The root of overfitting

Intuition

- Given a specific dataset $\mathcal{D}$, the learned function $f_{\mathcal{D}}(x)$ has two types of errors
  - $f_{\mathcal{D}}(x)$ fluctuates around the best possible $f(x)$ if $\mathcal{D}$ is infinitely large. This error is called *variance*
  - $f_{\mathcal{D}}(x)$ or $f(x)$ is a specific type of function (eg. linear), thus, it might not be able to model complex relations. This error is called *bias*
- The total error is the *sum of variance and bias*
- Simpler models (functions $f(x)$) has a smaller variance but a larger bias
- Complex models (functions $f(x)$) has a larger variance but a smaller bias

Thus, one needs to balance bias and variance.

Regularized models reduces variance (because they lead to simpler models) but then increase the bias.

Summary

- We can extend linear regression to nonlinear regression by using nonlinear basis functions to compose features.
- However, *we have never suggested how to choose the right nonlinear basis functions to use*
- Furthermore, with complex nonlinear basis functions, we increase the risk of *overfitting*
- Overfitting leads to poor generalization error, which means we have a bad tradeoff between variance and bias.