

CSCI567 Machine Learning (Fall 2020)

Prof. Haipeng Luo

U of Southern California

Sep 3, 2020

1 / 55

Administration

Outline

- 1 Administration
- 2 Review of last lecture
- 3 Linear regression
- 4 Linear regression with nonlinear basis
- 5 Overfitting and preventing overfitting

3 / 55

Outline

- 1 Administration
- 2 Review of last lecture
- 3 Linear regression
- 4 Linear regression with nonlinear basis
- 5 Overfitting and preventing overfitting

2 / 55

Administration

Administrative stuff

Please enroll in Piazza (163/202 as of this morning).

HW1 to be released today.

Programming project:

- invitation to enroll is out
- all six tasks available now
- collaboration not allowed
- #submissions updated *from 10 to ∞*

4 / 55

Outline

- 1 Administration
- 2 Review of last lecture
- 3 Linear regression
- 4 Linear regression with nonlinear basis
- 5 Overfitting and preventing overfitting

5 / 55

Datasets

Training data

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$
- They are used to learn $f(\cdot)$

Test data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_M, y_M)\}$
- They are used to evaluate how well $f(\cdot)$ will do.

Development/Validation data

- L samples/instances: $\mathcal{D}^{\text{DEV}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_L, y_L)\}$
- They are used to optimize hyper-parameter(s).

These three sets should *not* overlap!

7 / 55

Multi-class classification

Training data (set)

- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)\}$
- Each $\mathbf{x}_n \in \mathbb{R}^D$ is called a feature vector.
- Each $y_n \in [C] = \{1, 2, \dots, C\}$ is called a label/class/category.
- They are used to learn $f: \mathbb{R}^D \rightarrow [C]$ for future prediction.

Special case: binary classification

- Number of classes: $C = 2$
- Conventional labels: $\{0, 1\}$ or $\{-1, +1\}$

K-NNC: predict the majority label within the K -nearest neighbor set

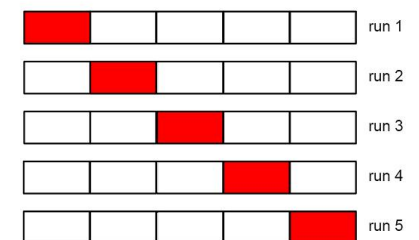
6 / 55

S-fold Cross-validation

What if we do not have a development set?

- Split the training data into S equal parts.
- Use each part *in turn* as a development dataset and use the others as a training dataset.
- Choose the hyper-parameter leading to best *average* performance.

S = 5: 5-fold cross validation



Special case: $S = N$, called leave-one-out.

8 / 55

High level picture

Typical steps of developing a machine learning system:

- Collect data, split into training, development, and test sets.
- *Train a model with a machine learning algorithm.* Most often we apply cross-validation to tune hyper-parameters.
- Evaluate using the test data and report performance.
- Use the model to predict future/make decisions.

How to do the *red part* exactly?

Outline

- 1 Administration
- 2 Review of last lecture
- 3 **Linear regression**
 - Motivation
 - Setup and Algorithm
 - Discussions
- 4 Linear regression with nonlinear basis
- 5 Overfitting and preventing overfitting

Regression

Predicting a continuous outcome variable using past observations

- Predicting future temperature (last lecture)
- Predicting the amount of rainfall
- Predicting the demand of a product
- Predicting the sale price of a house
- ...

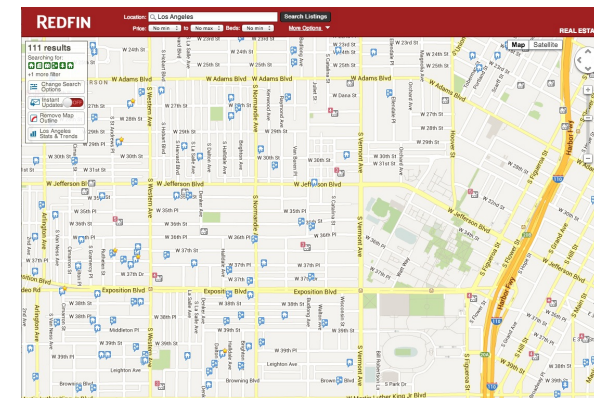
Key difference from classification

- continuous vs discrete
- measure *prediction errors* differently.
- lead to quite different learning algorithms.

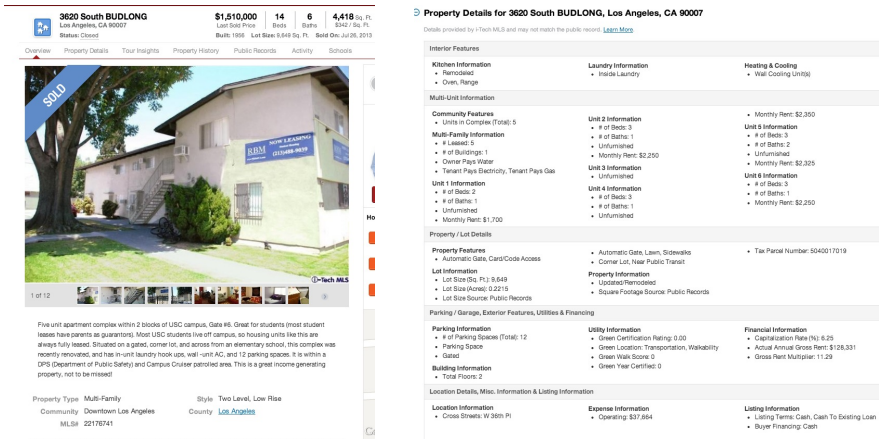
Linear Regression: regression with linear models

Ex: Predicting the sale price of a house

Retrieve historical sales records (training data)

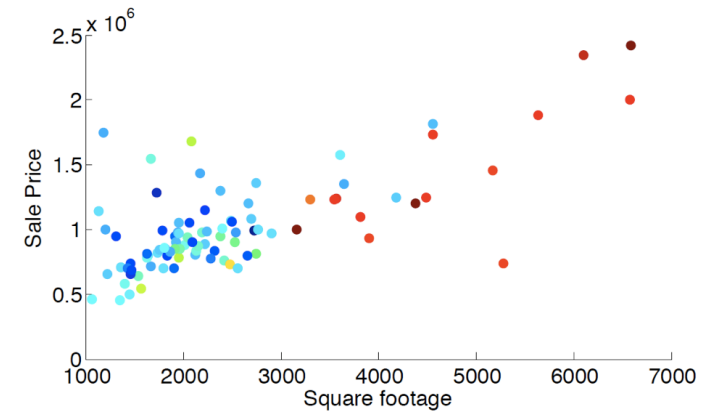


Features used to predict



13 / 55

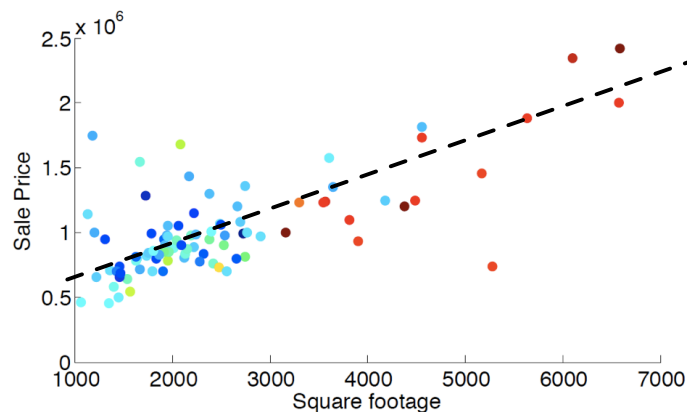
Correlation between square footage and sale price



14 / 55

Possibly linear relationship

Sale price \approx **price_per_sqft** \times square_footage + **fixed_expense**
(slope) (intercept)



15 / 55

How to learn the unknown parameters?

How to measure error for one prediction?

- The classification error (0-1 loss, i.e. *right* or *wrong*) is *inappropriate* for continuous outcomes.
- We can look at
 - *absolute* error: $|\text{prediction} - \text{sale price}|$
 - or *squared* error: $(\text{prediction} - \text{sale price})^2$ (**most common**)

Goal: pick the model (unknown parameters) that minimizes the average/total prediction error, but on *what set*?

- test set, ideal but we *cannot use test set while training*
- training set ✓

16 / 55

Example

Predicted price = **price_per_sqft** × square_footage + **fixed_expense**

one model: price_per_sqft = 0.3K, fixed_expense = 210K

sqft	sale price (K)	prediction (K)	squared error
2000	810	810	0
2100	907	840	67^2
1100	312	540	228^2
5500	2,600	1,860	740^2
...
Total			$0 + 67^2 + 228^2 + 740^2 + \dots$

Adjust price_per_sqft and fixed_expense such that the total squared error is minimized.

Formal setup for linear regression

Input: $\mathbf{x} \in \mathbb{R}^D$ (features, covariates, context, predictors, etc)

Output: $y \in \mathbb{R}$ (responses, targets, outcomes, etc)

Training data: $\mathcal{D} = \{(\mathbf{x}_n, y_n), n = 1, 2, \dots, N\}$

Linear model: $f: \mathbb{R}^D \rightarrow \mathbb{R}$, with $f(\mathbf{x}) = w_0 + \sum_{d=1}^D w_d x_d = w_0 + \mathbf{w}^T \mathbf{x}$ (superscript T stands for transpose), i.e. a *hyper-plane* parametrized by

- $\mathbf{w} = [w_1 \ w_2 \ \dots \ w_D]^T$ (weights, weight vector, parameter vector, etc)
- bias w_0

NOTE: for notation convenience, very often we

- append 1 to each \mathbf{x} as the first feature: $\tilde{\mathbf{x}} = [1 \ x_1 \ x_2 \ \dots \ x_D]^T$
- let $\tilde{\mathbf{w}} = [w_0 \ w_1 \ w_2 \ \dots \ w_D]^T$, a concise representation of all $D + 1$ parameters
- the model becomes simply $f(\mathbf{x}) = \tilde{\mathbf{w}}^T \tilde{\mathbf{x}}$
- sometimes just use $\mathbf{w}, \mathbf{x}, D$ for $\tilde{\mathbf{w}}, \tilde{\mathbf{x}}, D + 1$!

Goal

Minimize total squared error

- **Residual Sum of Squares** (RSS), a function of $\tilde{\mathbf{w}}$

$$\text{RSS}(\tilde{\mathbf{w}}) = \sum_n (f(\mathbf{x}_n) - y_n)^2 = \sum_n (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}} - y_n)^2$$

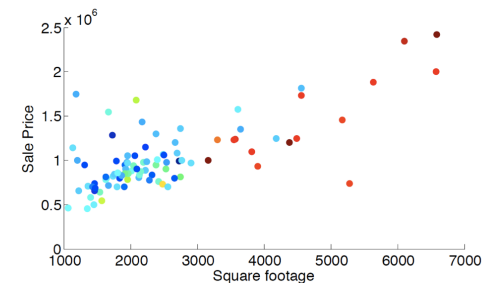
- find $\tilde{\mathbf{w}}^* = \underset{\tilde{\mathbf{w}} \in \mathbb{R}^{D+1}}{\text{argmin}} \text{RSS}(\tilde{\mathbf{w}})$, i.e. **least (mean) squares solution**
(more generally called **empirical risk minimizer**)

- *reduce machine learning to optimization*

- in principle can apply any optimization algorithm, but linear regression admits a *closed-form solution*

Warm-up: $D = 0$

Only one parameter w_0 : constant prediction $f(\mathbf{x}) = w_0$



f is a horizontal line, where should it be?

Warm-up: $D = 0$

Optimization objective becomes

$$\begin{aligned}
 \text{RSS}(w_0) &= \sum_n (w_0 - y_n)^2 \quad (\text{it's a *quadratic* } aw_0^2 + bw_0 + c) \\
 &= Nw_0^2 - 2 \left(\sum_n y_n \right) w_0 + \text{cnt.} \\
 &= N \left(w_0 - \frac{1}{N} \sum_n y_n \right)^2 + \text{cnt.}
 \end{aligned}$$

It is clear that $w_0^* = \frac{1}{N} \sum_n y_n$, i.e. the **average**

Exercise: what if we use absolute error instead of squared error?

Warm-up: $D = 1$

Optimization objective becomes

$$\text{RSS}(\tilde{\mathbf{w}}) = \sum_n (w_0 + w_1 x_n - y_n)^2$$

General approach: find **stationary points**, i.e., points with **zero gradient**

$$\begin{aligned}
 \begin{cases} \frac{\partial \text{RSS}(\tilde{\mathbf{w}})}{\partial w_0} = 0 \\ \frac{\partial \text{RSS}(\tilde{\mathbf{w}})}{\partial w_1} = 0 \end{cases} &\Rightarrow \begin{cases} \sum_n (w_0 + w_1 x_n - y_n) = 0 \\ \sum_n (w_0 + w_1 x_n - y_n) x_n = 0 \end{cases} \\
 \Rightarrow \begin{cases} Nw_0 + w_1 \sum_n x_n = \sum_n y_n \\ w_0 \sum_n x_n + w_1 \sum_n x_n^2 = \sum_n y_n x_n \end{cases} & \quad (\text{a linear system}) \\
 \Rightarrow \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix} \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} &= \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}
 \end{aligned}$$

Least square solution for $D = 1$

$$\Rightarrow \begin{pmatrix} w_0^* \\ w_1^* \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}^{-1} \begin{pmatrix} \sum_n y_n \\ \sum_n x_n y_n \end{pmatrix}$$

(assuming the matrix is invertible)

Are stationary points minimizers?

- yes for **convex** objectives (RSS is convex in $\tilde{\mathbf{w}}$)
- not true in general

General least square solution

Objective

$$\text{RSS}(\tilde{\mathbf{w}}) = \sum_n (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}} - y_n)^2$$

Again, find stationary points (**multivariate calculus**)

$$\begin{aligned}
 \nabla \text{RSS}(\tilde{\mathbf{w}}) &= 2 \sum_n \tilde{\mathbf{x}}_n (\tilde{\mathbf{x}}_n^T \tilde{\mathbf{w}} - y_n) \propto \left(\sum_n \tilde{\mathbf{x}}_n \tilde{\mathbf{x}}_n^T \right) \tilde{\mathbf{w}} - \sum_n \tilde{\mathbf{x}}_n y_n \\
 &= (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}) \tilde{\mathbf{w}} - \tilde{\mathbf{X}}^T \mathbf{y} = \mathbf{0}
 \end{aligned}$$

where

$$\tilde{\mathbf{X}} = \begin{pmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \vdots \\ \tilde{\mathbf{x}}_N^T \end{pmatrix} \in \mathbb{R}^{N \times (D+1)}, \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{pmatrix} \in \mathbb{R}^N$$

General least square solution

$$(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}) \tilde{\mathbf{w}} - \tilde{\mathbf{X}}^T \mathbf{y} = \mathbf{0} \Rightarrow \tilde{\mathbf{w}}^* = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{y}$$

assuming $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ (**covariance matrix**) is invertible for now.

Again by convexity $\tilde{\mathbf{w}}^*$ is the minimizer of RSS.

Verify the solution when D = 1:

$$\tilde{\mathbf{X}}^T \tilde{\mathbf{X}} = \begin{pmatrix} 1 & 1 & \cdots & 1 \\ x_1 & x_2 & \cdots & x_N \end{pmatrix} \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \cdots & \cdots \\ 1 & x_N \end{pmatrix} = \begin{pmatrix} N & \sum_n x_n \\ \sum_n x_n & \sum_n x_n^2 \end{pmatrix}$$

when D = 0: $(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} = \frac{1}{N}$, $\tilde{\mathbf{X}}^T \mathbf{y} = \sum_n y_n$

Another approach

RSS is a quadratic:

$$\begin{aligned} \text{RSS}(\tilde{\mathbf{w}}) &= \sum_n (\tilde{\mathbf{w}}^T \tilde{\mathbf{x}}_n - y_n)^2 = \|\tilde{\mathbf{X}} \tilde{\mathbf{w}} - \mathbf{y}\|_2^2 \\ &= (\tilde{\mathbf{X}} \tilde{\mathbf{w}} - \mathbf{y})^T (\tilde{\mathbf{X}} \tilde{\mathbf{w}} - \mathbf{y}) \\ &= \tilde{\mathbf{w}}^T \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \tilde{\mathbf{w}} - \mathbf{y}^T \tilde{\mathbf{X}} \tilde{\mathbf{w}} - \tilde{\mathbf{w}}^T \tilde{\mathbf{X}}^T \mathbf{y} + \text{cnt.} \\ &= \left(\tilde{\mathbf{w}} - (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{y} \right)^T (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}) \left(\tilde{\mathbf{w}} - (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{y} \right) + \text{cnt.} \end{aligned}$$

Note: $\mathbf{u}^T (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}) \mathbf{u} = (\tilde{\mathbf{X}} \mathbf{u})^T \tilde{\mathbf{X}} \mathbf{u} = \|\tilde{\mathbf{X}} \mathbf{u}\|_2^2 \geq 0$ and is 0 if $\mathbf{u} = \mathbf{0}$.
So $\tilde{\mathbf{w}}^* = (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} \tilde{\mathbf{X}}^T \mathbf{y}$ is the minimizer.

Computational complexity

Bottleneck of computing

$$\tilde{\mathbf{w}}^* = \left(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \right)^{-1} \tilde{\mathbf{X}}^T \mathbf{y}$$

is to invert the matrix $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}} \in \mathbb{R}^{(D+1) \times (D+1)}$

- naively need $O(D^3)$ time
- there are many faster approaches (such as conjugate gradient)

What if $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ is not invertible

What does that imply?

Recall $(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}) \mathbf{w}^* = \tilde{\mathbf{X}}^T \mathbf{y}$. If $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ not invertible, this equation has

- no solution (\Rightarrow RSS has no minimizer? **X**)
- or infinitely many solutions (\Rightarrow infinitely many minimizers **✓**)

What if $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ is not invertible

Why would that happen?

One situation: $N < D + 1$, i.e. not enough data to estimate all parameters.

Example: $D = N = 1$

sqft	sale price
1000	500K

Any line passing this single point is a minimizer of RSS.

How about the following?

$D = 1, N = 2$

sqft	sale price
1000	500K
1000	600K

Any line passing **the average** is a minimizer of RSS.

$D = 2, N = 3?$

sqft	#bedroom	sale price
1000	2	500K
1500	3	700K
2000	4	800K

Again *infinitely many minimizers*.

How to resolve this issue?

Intuition: what does inverting $\tilde{\mathbf{X}}^T \tilde{\mathbf{X}}$ do?

$$\text{eigendecomposition: } \tilde{\mathbf{X}}^T \tilde{\mathbf{X}} = \mathbf{U}^T \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_D & 0 \\ 0 & \cdots & 0 & \lambda_{D+1} \end{bmatrix} \mathbf{U}$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_{D+1} \geq 0$ are **eigenvalues**.

$$\text{inverse: } (\tilde{\mathbf{X}}^T \tilde{\mathbf{X}})^{-1} = \mathbf{U}^T \begin{bmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_D} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{D+1}} \end{bmatrix} \mathbf{U}$$

i.e. just inverse the eigenvalues

How to solve this problem?

Non-invertible \Rightarrow some eigenvalues are 0.

One natural fix: add something positive

$$\tilde{\mathbf{X}}^T \tilde{\mathbf{X}} + \lambda \mathbf{I} = \mathbf{U}^T \begin{bmatrix} \lambda_1 + \lambda & 0 & \cdots & 0 \\ 0 & \lambda_2 + \lambda & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \lambda_D + \lambda & 0 \\ 0 & \cdots & 0 & \lambda_{D+1} + \lambda \end{bmatrix} \mathbf{U}$$

where $\lambda > 0$ and \mathbf{I} is the identity matrix. Now it is invertible:

$$(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}} + \lambda \mathbf{I})^{-1} = \mathbf{U}^T \begin{bmatrix} \frac{1}{\lambda_1 + \lambda} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2 + \lambda} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & \frac{1}{\lambda_D + \lambda} & 0 \\ 0 & \cdots & 0 & \frac{1}{\lambda_{D+1} + \lambda} \end{bmatrix} \mathbf{U}$$

Fix the problem

The solution becomes

$$\tilde{\mathbf{w}}^* = \left(\tilde{\mathbf{X}}^T \tilde{\mathbf{X}} + \lambda \mathbf{I} \right)^{-1} \tilde{\mathbf{X}}^T \mathbf{y}$$

- not a minimizer of the original RSS

λ is a *hyper-parameter*, can be tuned by cross-validation.

33 / 55

Outline

- 1 Administration
- 2 Review of last lecture
- 3 Linear regression
- 4 Linear regression with nonlinear basis
- 5 Overfitting and preventing overfitting

35 / 55

Comparison to NNC

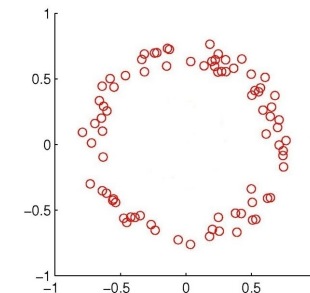
Parametric versus non-parametric

- **Parametric methods:** the size of the model does *not grow* with the size of the training set N .
 - e.g. linear regression, $D + 1$ parameters, independent of N .
- **Non-parametric methods:** the size of the model *grows* with the size of the training set.
 - e.g. NNC, the training set itself needs to be kept in order to predict. Thus, the size of the model is the size of the training set.

34 / 55

What if linear model is not a good fit?

Example: a straight line is a bad fit for the following data



36 / 55

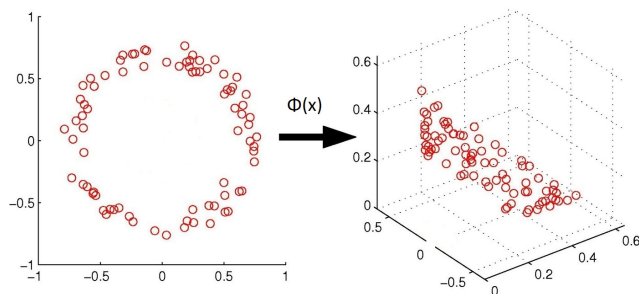
Solution: nonlinearly transformed features

1. Use a nonlinear mapping

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

to transform the data to a more complicated feature space

2. Then apply linear regression (hope: linear model is a better fit for the new feature space).



37 / 55

Regression with nonlinear basis

Model: $f(x) = w^T \phi(x)$ where $w \in \mathbb{R}^M$

Objective:

$$\text{RSS}(w) = \sum_n (w^T \phi(x_n) - y_n)^2$$

Similar least square solution:

$$w^* = (\Phi^T \Phi)^{-1} \Phi^T y \quad \text{where} \quad \Phi = \begin{pmatrix} \phi(x_1)^T \\ \phi(x_2)^T \\ \vdots \\ \phi(x_N)^T \end{pmatrix} \in \mathbb{R}^{N \times M}$$

38 / 55

Example

Polynomial basis functions for $D = 1$

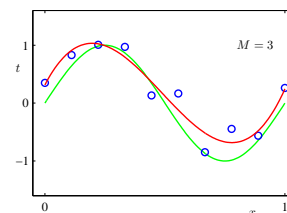
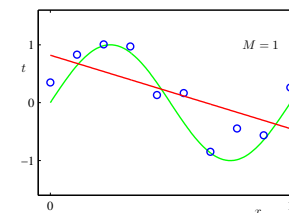
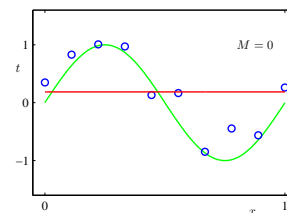
$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^M w_m x^m$$

Learning a linear model in the new space
= learning an *M-degree polynomial model* in the original space

39 / 55

Example

Fitting a noisy sine function with a polynomial ($M = 0, 1$, or 3):



40 / 55

Why nonlinear?

Can I use a fancy **linear feature map**?

$$\phi(x) = \begin{bmatrix} x_1 - x_2 \\ 3x_4 - x_3 \\ 2x_1 + x_4 + x_5 \\ \vdots \end{bmatrix} = \mathbf{A}x \quad \text{for some } \mathbf{A} \in \mathbb{R}^{M \times D}$$

No, it basically *does nothing* since

$$\min_{w \in \mathbb{R}^M} \sum_n (w^T \mathbf{A}x_n - y_n)^2 = \min_{w' \in \text{Im}(\mathbf{A}^T) \subset \mathbb{R}^D} \sum_n (w'^T x_n - y_n)^2$$

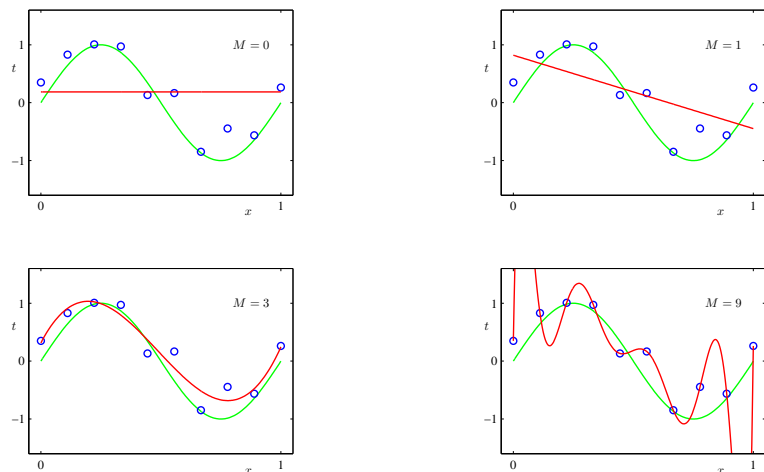
We will see more nonlinear mappings soon.

Outline

- 1 Administration
- 2 Review of last lecture
- 3 Linear regression
- 4 Linear regression with nonlinear basis
- 5 Overfitting and preventing overfitting

Should we use a very complicated mapping?

Ex: fitting a noisy sine function with a polynomial:



Underfitting and Overfitting

$M \leq 2$ is *underfitting* the data

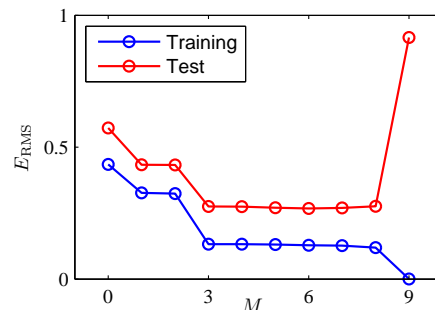
- large training error
- large test error

$M \geq 9$ is *overfitting* the data

- small training error
- **large test error**

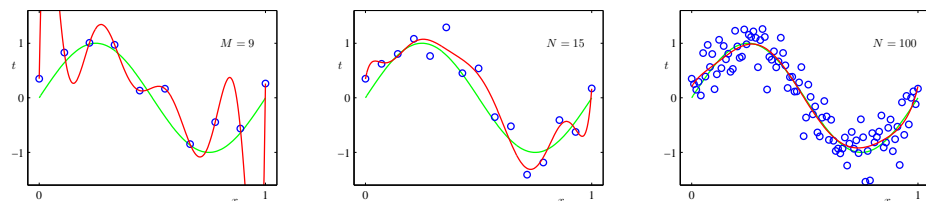
More complicated models \Rightarrow larger gap between training and test error

How to prevent overfitting?



Method 1: use more training data

The more, the merrier



More data \Rightarrow smaller gap between training and test error

45 / 55

Method 2: control the model complexity

For polynomial basis, the **degree M** clearly controls the complexity

- use cross-validation to pick hyper-parameter M

When M or in general Φ is fixed, are there still other ways to control complexity?

46 / 55

Magnitude of weights

Least square solution for the polynomial example:

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
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

Intuitively, **large weights** \Rightarrow **more complex model**

47 / 55

How to make w small?

Regularized linear regression: new objective

$$\mathcal{E}(w) = \text{RSS}(w) + \lambda R(w)$$

Goal: find $w^* = \text{argmin}_w \mathcal{E}(w)$

- $R : \mathbb{R}^D \rightarrow \mathbb{R}^+$ is the **regularizer**
 - measure how complex the model w is
 - common choices: $\|w\|_2^2$, $\|w\|_1$, etc.
- $\lambda > 0$ is the **regularization coefficient**
 - $\lambda = 0$, no regularization
 - $\lambda \rightarrow +\infty$, $w \rightarrow \text{argmin}_w R(w)$
 - i.e. control **trade-off** between training error and complexity

48 / 55

The effect of λ

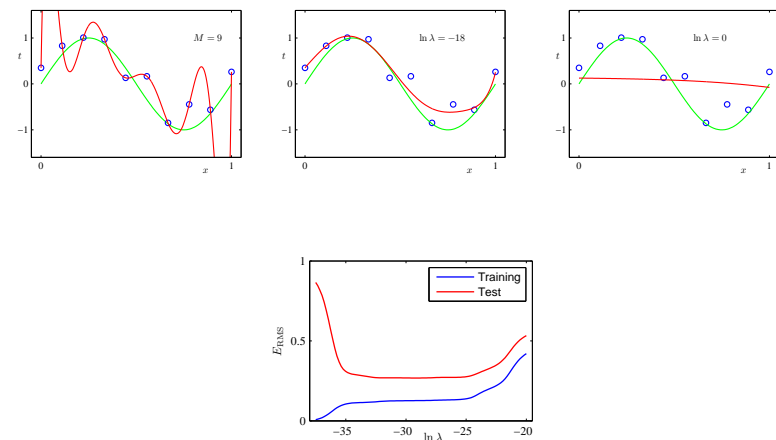
when we increase regularization coefficient λ

	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
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

49 / 55

The trade-off

when we increase regularization coefficient λ



50 / 55

How to solve the new objective?

Simple for $R(w) = \|w\|_2^2$:

$$\mathcal{E}(w) = \text{RSS}(w) + \lambda \|w\|_2^2 = \|\Phi w - y\|_2^2 + \lambda \|w\|_2^2$$

$$\nabla \mathcal{E}(w) = 2(\Phi^T \Phi w - \Phi^T y) + 2\lambda w = 0$$

$$\Rightarrow (\Phi^T \Phi + \lambda I) w = \Phi^T y$$

$$\Rightarrow w^* = (\Phi^T \Phi + \lambda I)^{-1} \Phi^T y$$

Note the same form as in the fix when $X^T X$ is not invertible!

For other regularizers, as long as it's **convex**, standard optimization algorithms can be applied.

51 / 55

Equivalent form

Regularization is also sometimes formulated as

$$\underset{w}{\operatorname{argmin}} \text{RSS}(w) \quad \text{subject to } R(w) \leq \beta$$

where β is some hyper-parameter.

Finding the solution becomes a **constrained optimization problem**.

Choosing either λ or β can be done by cross-validation.

52 / 55

Summary

$$\mathbf{w}^* = (\Phi^T \Phi + \lambda \mathbf{I})^{-1} \Phi^T \mathbf{y}$$

Important to understand the derivation than remembering the formula

Overfitting: small training error but large test error

Preventing Overfitting: more data + regularization

Recall the question

Typical steps of developing a machine learning system:

- Collect data, split into training, development, and test sets.
- *Train a model with a machine learning algorithm.* Most often we apply cross-validation to tune hyper-parameters.
- Evaluate using the test data and report performance.
- Use the model to predict future/make decisions.

How to do the *red part* exactly?

General idea to derive ML algorithms

1. Pick a set of **models** \mathcal{F}
 - e.g. $\mathcal{F} = \{f(x) = \mathbf{w}^T \mathbf{x} \mid \mathbf{w} \in \mathbb{R}^D\}$
 - e.g. $\mathcal{F} = \{f(x) = \mathbf{w}^T \Phi(x) \mid \mathbf{w} \in \mathbb{R}^M\}$
2. Define **error/loss** $L(y', y)$
3. Find **empirical risk minimizer (ERM)**:

$$\mathbf{f}^* = \operatorname{argmin}_{f \in \mathcal{F}} \sum_{n=1}^N L(f(x_n), y_n)$$

or **regularized empirical risk minimizer**:

$$\mathbf{f}^* = \operatorname{argmin}_{f \in \mathcal{F}} \sum_{n=1}^N L(f(x_n), y_n) + \lambda R(f)$$

ML becomes optimization