CSCI567 Machine Learning (Fall 2016)

Dr. Yan Liu

yanliu.cs@usc.edu

September 27, 2016

Outline

Computational complexity

Bottleneck of computing the solution

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

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

How many operations do we need?

- On the order of $O((\mathrm{D}+1)^3)$ (using Gauss-Jordan elimination) or $O((\mathrm{D}+1)^{2.373})$ (recent advances in computing)
- Impractical for very large D

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient (ignoring the constant factor) $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}$

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient (ignoring the constant factor) $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}$
  2. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \nabla RSS(\tilde{\boldsymbol{w}})$

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient (ignoring the constant factor) $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}$
  2. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \nabla RSS(\tilde{\boldsymbol{w}})$
  3. $t \leftarrow t + 1$

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient (ignoring the constant factor) $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}$
  2. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \nabla RSS(\tilde{\boldsymbol{w}})$
  3. $t \leftarrow t + 1$

What is the complexity here?

Why would this work?

If gradient descent converges, it will converge to the same solution using matrix inversion.

This is because $RSS(\tilde{\boldsymbol{w}})$ is a convex function in its parameters $\boldsymbol{w}$

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

Why would this work?

If gradient descent converges, it will converge to the same solution using matrix inversion.

This is because $RSS(\tilde{\boldsymbol{w}})$ is a convex function in its parameters $\boldsymbol{w}$

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

$$\Rightarrow \frac{\partial^2 RSS(\tilde{\boldsymbol{w}})}{\partial \tilde{\boldsymbol{w}} \tilde{\boldsymbol{w}}^{\mathrm{T}}} = 2 \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}$$

Why would this work?

If gradient descent converges, it will converge to the same solution using matrix inversion.

This is because $RSS(\tilde{\boldsymbol{w}})$ is a convex function in its parameters $\boldsymbol{w}$

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

$$\Rightarrow \frac{\partial^2 RSS(\tilde{\boldsymbol{w}})}{\partial \tilde{\boldsymbol{w}} \tilde{\boldsymbol{w}}^{\mathrm{T}}} = 2 \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}$$

as $\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}$ is positive semidefinite, because for any $\boldsymbol{v}$

$$\boldsymbol{v}^{\mathrm{T}} \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \boldsymbol{v} = \| \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{v} \|_2^2 \geq 0$$

Stochastic gradient descent

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$

Stochastic gradient descent

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_t$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_t = (\tilde{\boldsymbol{x}}_t^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_t)\tilde{\boldsymbol{x}}_t$$

Stochastic gradient descent

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_t$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_t = (\tilde{\boldsymbol{x}}_t^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_t)\tilde{\boldsymbol{x}}_t$$

  3. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \boldsymbol{g}_t$

Stochastic gradient descent

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_t$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_t = (\tilde{\boldsymbol{x}}_t^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_t)\tilde{\boldsymbol{x}}_t$$

  3. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \boldsymbol{g}_t$
  4. $t \leftarrow t + 1$

Stochastic gradient descent

Widrow-Hoff rule: update parameters using one example at a time

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (anything reasonable is fine); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. random choose a training a sample $\boldsymbol{x}_t$
  2. Compute its contribution to the gradient (ignoring the constant factor)

$$\boldsymbol{g}_t = (\tilde{\boldsymbol{x}}_t^{\mathrm{T}} \tilde{\boldsymbol{w}}^{(t)} - y_t)\tilde{\boldsymbol{x}}_t$$

  3. Update the parameters $\tilde{\boldsymbol{w}}^{(t+1)} = \tilde{\boldsymbol{w}}^{(t)} - \eta \boldsymbol{g}_t$
  4. $t \leftarrow t + 1$

What is the complexity here?

Mini-summary

- Batch gradient descent computes the exact gradient.
- Stochastic gradient descent computes the gradient pretending only one instance.
  Its expectation equals to the true gradient.
- Other forms can be used.
  Mini-batch: trade-off between accuracy of estimating gradient and computational cost
- Similar ideas extend to other optimization problems in machine learning.
  For large-scale problems, stochastic gradient descent often works well.

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

Can you think of any reasons why that could happen?

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

Can you think of any reasons why that could happen?

Answer 1: N $<$ D. Intuitively, not enough data to estimate all the parameters.

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

Can you think of any reasons why that could happen?

Answer 1: N $<$ D. Intuitively, not enough data to estimate all the parameters.

Answer 2: $\boldsymbol{X}$ columns are not linearly independent. Intuitively, there are two features that are perfectly correlated. In this case, solution is not unique.

Ridge regression

Intuition: what does a non-invertible $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ mean?

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \lambda_r & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \boldsymbol{U}$$

where $\lambda 1 \geq \lambda_2 \geq \cdots \lambda_r > 0$ and $r <$ D.

Ridge regression

Intuition: what does a non-invertible $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$ mean?

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} = \boldsymbol{U}^{\mathrm{T}} \begin{bmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & \cdots & \cdots & \cdots & 0 \\ 0 & \cdots & \cdots & \lambda_r & 0 \\ 0 & \cdots & \cdots & 0 & 0 \end{bmatrix} \boldsymbol{U}$$

where $\lambda 1 \geq \lambda_2 \geq \cdots \lambda_r > 0$ and $r <$ D.

Fix the problem by adding something positive

$$\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} + \lambda \boldsymbol{I} = \boldsymbol{U}^{\mathrm{T}}\mathrm{diag}(\lambda_1 + \lambda, \lambda_2 + \lambda, \cdots, \lambda)\boldsymbol{U}$$

where $\lambda > 0$ and $\boldsymbol{I}$ is the identity matrix

Regularized least square (ridge regression)

Solution

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

Regularized least square (ridge regression)

Solution

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

This is equivalent to adding an extra term to $RSS(\tilde{\boldsymbol{w}})$

$$\overbrace{\frac{1}{2}\left\{\tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - 2\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\tilde{\boldsymbol{w}}\right\}}^{RSS(\tilde{\boldsymbol{w}})} + \underbrace{\frac{1}{2}\lambda\|\tilde{w}\|_2^2}_{\text{regularization}}$$

Regularized least square (ridge regression)

Solution

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

This is equivalent to adding an extra term to $RSS(\tilde{\boldsymbol{w}})$

$$\overbrace{\frac{1}{2}\left\{\tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - 2\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\tilde{\boldsymbol{w}}\right\}}^{RSS(\tilde{\boldsymbol{w}})} + \underbrace{\frac{1}{2}\lambda\|\tilde{w}\|_2^2}_{\text{regularization}}$$

Benefits

- Numerically more stable, invertible matrix
- Prevent overfitting — more on this later

How to choose λ?

Again, λ is referred as *hyperparameter*, to be distinguished from $\boldsymbol{w}$.

- Use validation or cross-validation
- Other approaches such as Bayesian linear regression — we will describe them briefly later

linear regression versus nearest neighbors

Parametric versus non-parametric

- Parametric
  The size of the model does not grow with respect to the size of the training dataset.
  In linear regression, there are $D + 1$ parameters, irrelevant to how many training instances we have.
- Non-parametric
  The size of the model grows with respect to the size of the training dataset.
  In nearest neighbor classification, the training dataset itself needs to be kept in order to make prediction. Thus, the size of the model is the size of the training dataset.

Non-parametric does not mean *parameter-less*. It just means the number of parameters is a function of the training dataset.

linear regression versus nearest neighbors

Parametric versus non-parametric

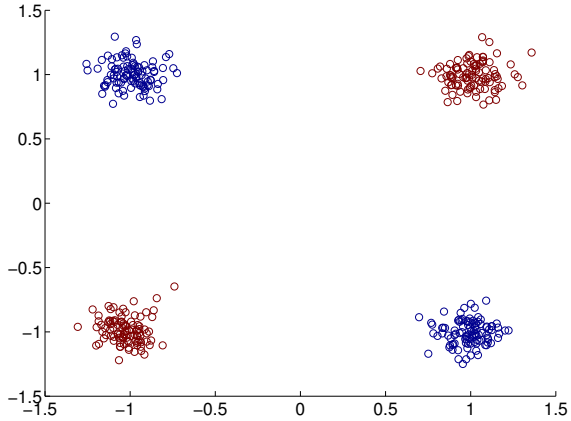
- Parametric
  The size of the model does not grow with respect to the size of the training dataset.
  In linear regression, there are $D + 1$ parameters, irrelevant to how many training instances we have.
- Non-parametric
  The size of the model grows with respect to the size of the training dataset.
  In nearest neighbor classification, the training dataset itself needs to be kept in order to make prediction. Thus, the size of the model is the size of the training dataset.

Non-parametric does not mean *parameter-less*. It just means the number of parameters is a function of the training dataset.

Outline

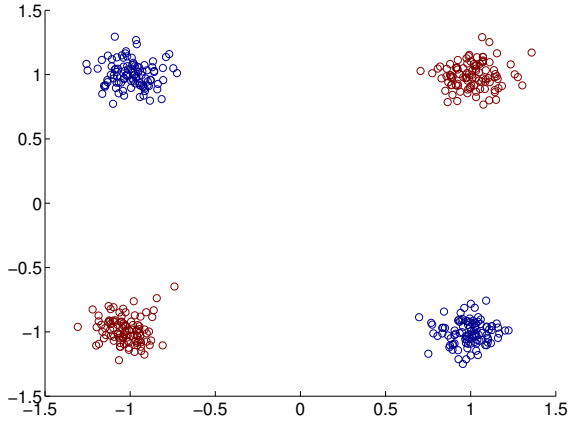
What if data is not linearly separable or fits to a line

Example of nonlinear classification

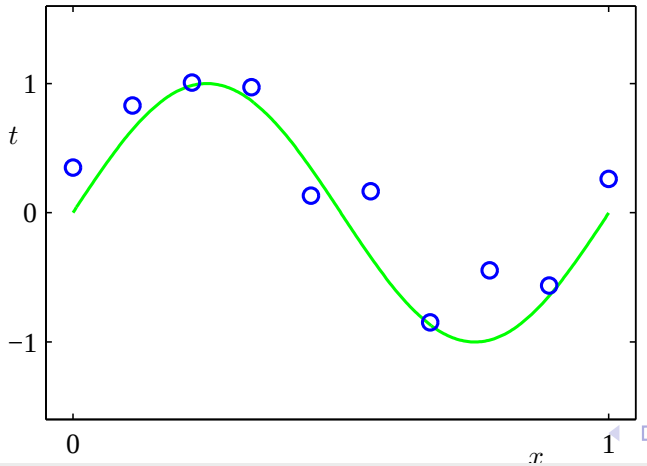


What if data is not linearly separable or fits to a line

Example of nonlinear classification



Example of nonlinear regression



Nonlinear basis for classification

Transform the input/feature

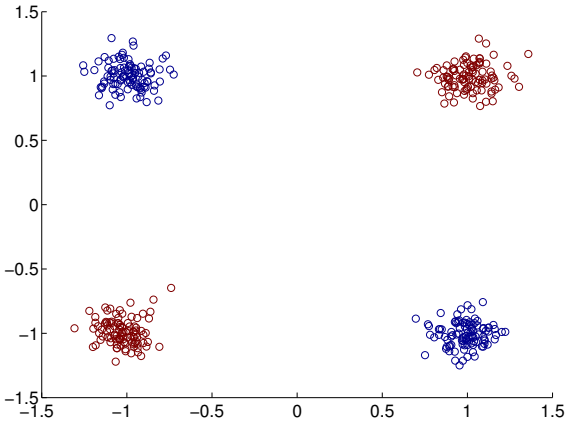
$$\phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^2 \rightarrow z = x_1 \cdot x_2$$

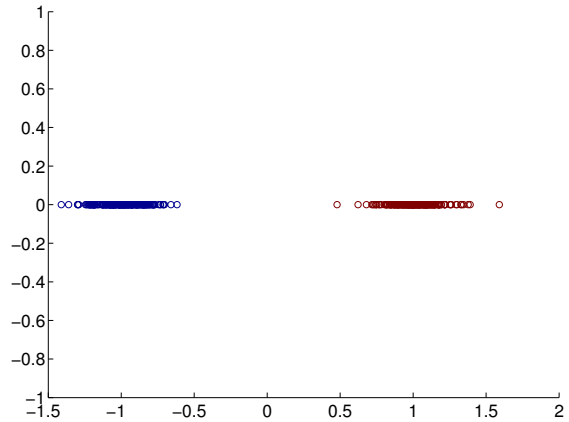
Nonlinear basis for classification

Transform the input/feature

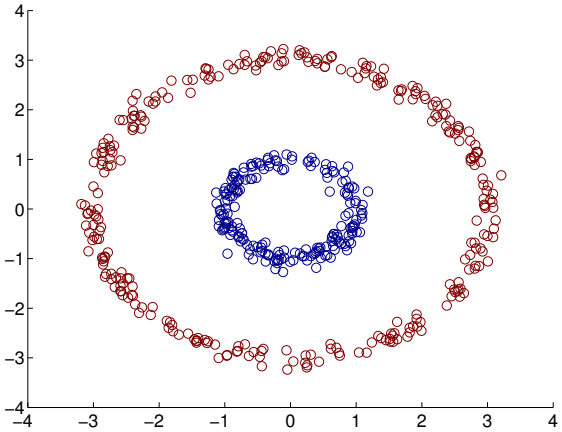
$$\phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^2 \rightarrow z = x_1 \cdot x_2$$

Transformed training data: linearly separable!



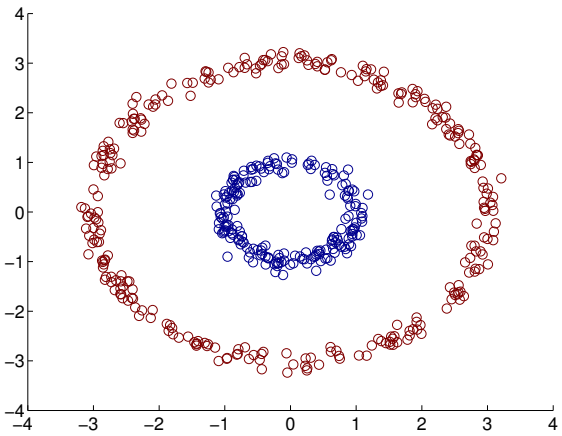


Another example



How to transform the input/feature?

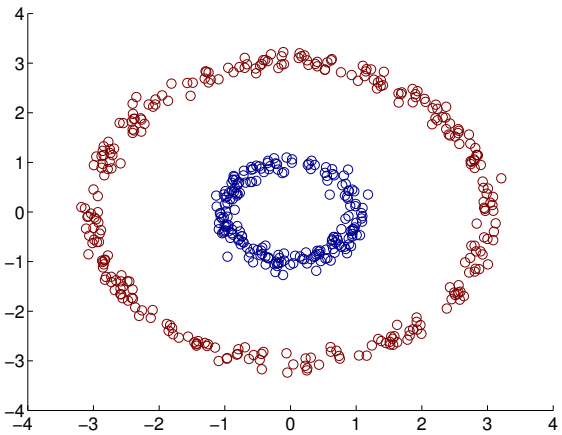
Another example



How to transform the input/feature?

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^2 \rightarrow \boldsymbol{z} = \begin{bmatrix} x_1^2 \\ x_1 \cdot x_2 \\ x_2^2 \end{bmatrix} \in \mathbb{R}^3$$

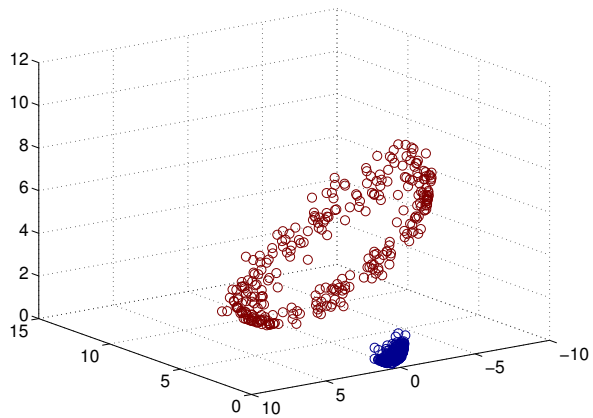
Another example



How to transform the input/feature?

$$\boldsymbol{\phi}(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^2 \rightarrow \boldsymbol{z} = \begin{bmatrix} x_1^2 \\ x_1 \cdot x_2 \\ x_2^2 \end{bmatrix} \in \mathbb{R}^3$$

Transformed training data: linearly separable



Intuition:
$\boldsymbol{w} = [1\ 0\ 1]^{\mathrm{T}}$ then,
$\boldsymbol{w}^{\mathrm{T}}\boldsymbol{z} = \|\boldsymbol{x}\|_2^2$, i.e., the distance to the origin!

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.

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.

With the new features, we can apply our learning techniques

- linear methods: prediction is based on $\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\phi}(\boldsymbol{x})$
- 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

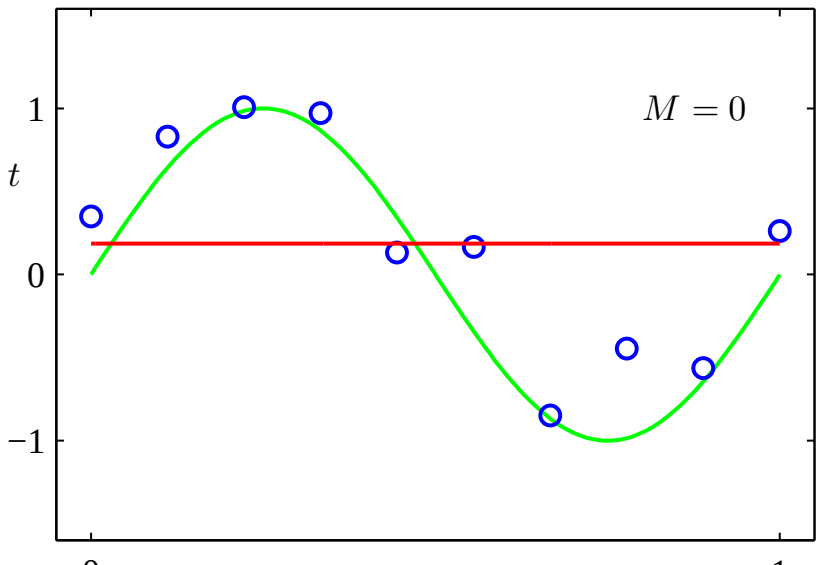
$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

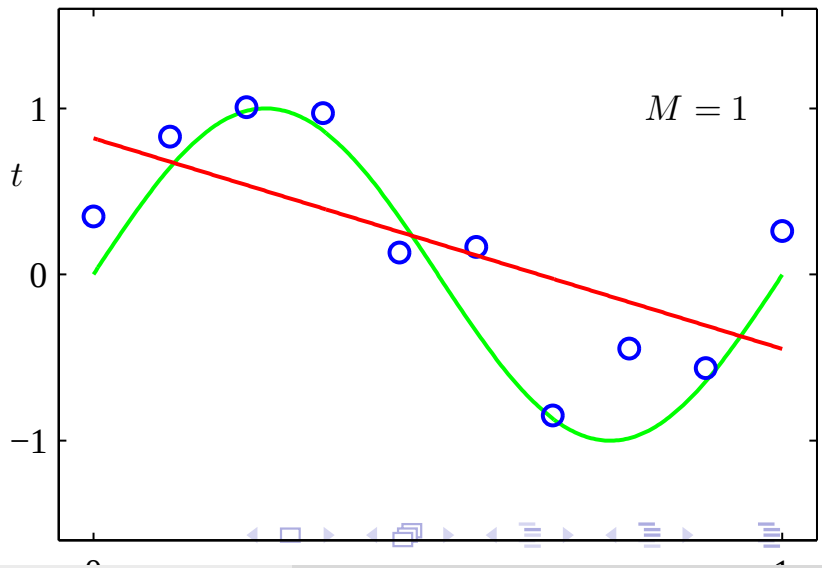
Example with regression

Polynomial basis functions

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

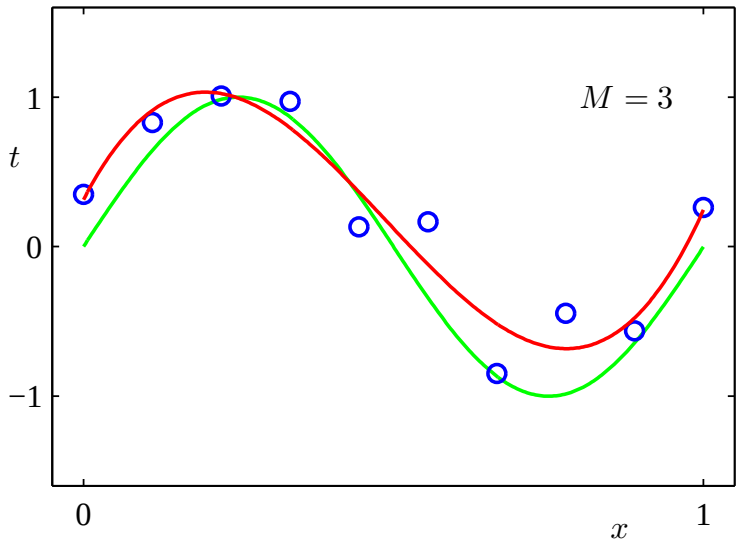
Fitting samples from a sine function: *underrfitting* as $f(x)$ is too simple





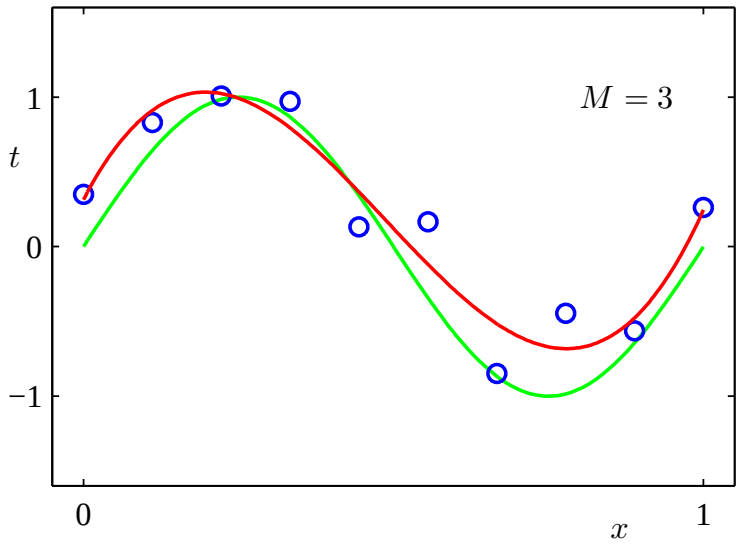
Adding more high-order basis

M=3

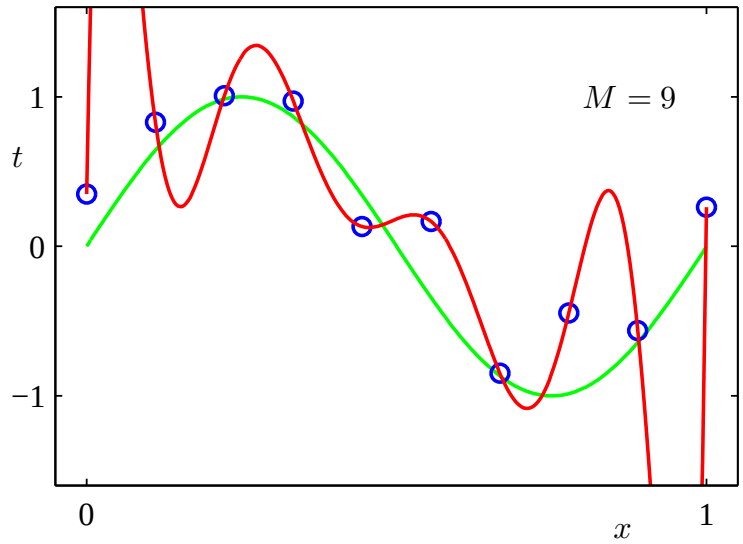


Adding more high-order basis

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!

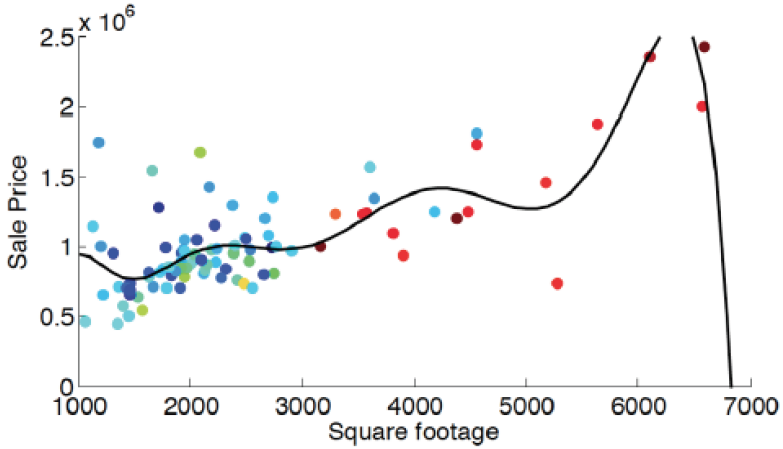
Overfiting

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 $M = 3$

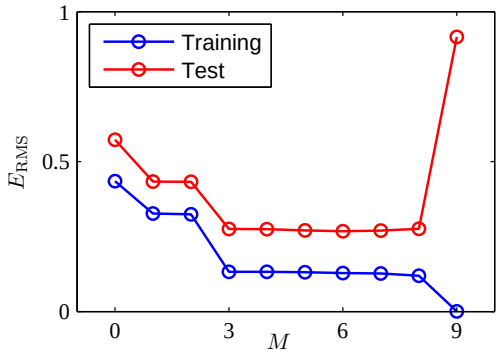


Note that the price would goes to zero (or negative) if you buy bigger ones!
This is called poor generalization/overfitting.

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.

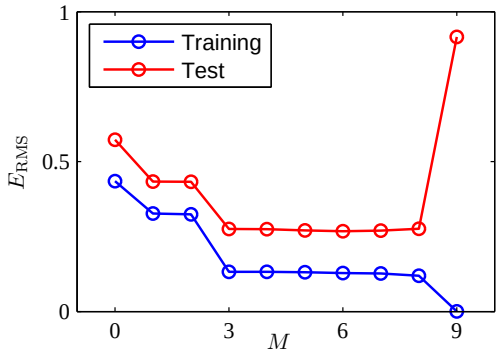


- Horizontal axis: *measure of model complexity*
  In this example, we use the maximum order of the polynomial basis functions.

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.

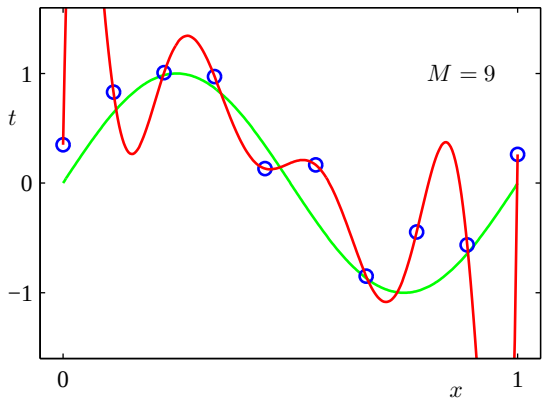


- Horizontal axis: *measure of model complexity*
  In this example, we use the maximum order of the polynomial basis functions.
- Vertical axis:
  1. For regression, the vertical axis would be RSS or RMS (squared root of RSS)
  2. For classification, the vertical axis would be classification error rate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

Outline

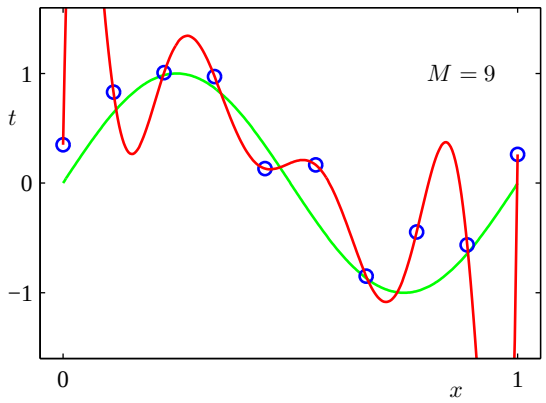
Use more training data to prevent over fitting

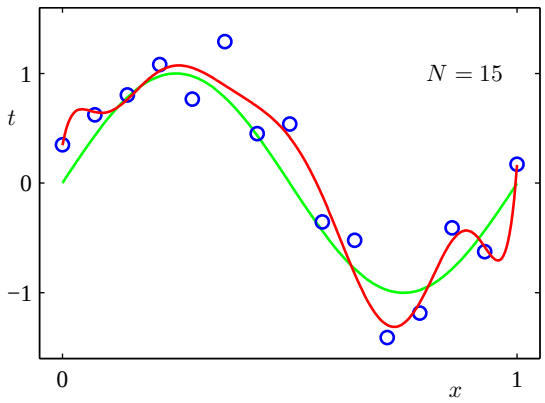
The more, the merrier



Use more training data to prevent over fitting

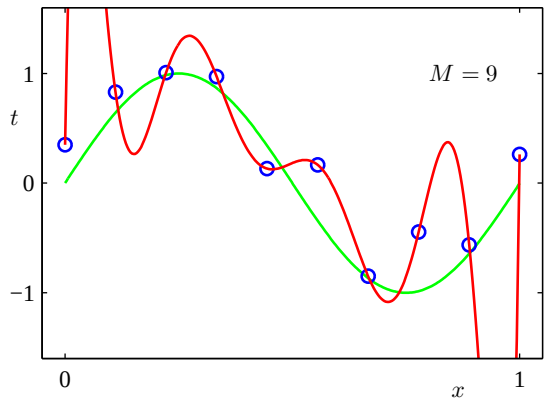
The more, the merrier

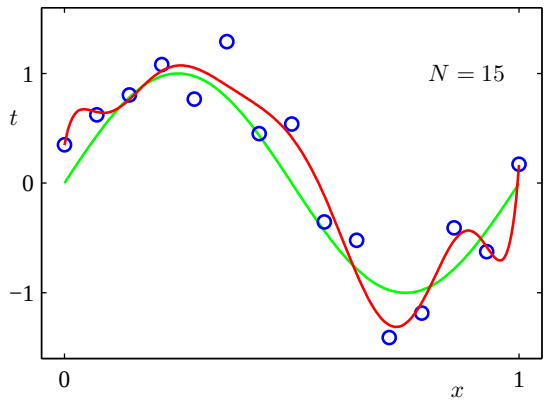


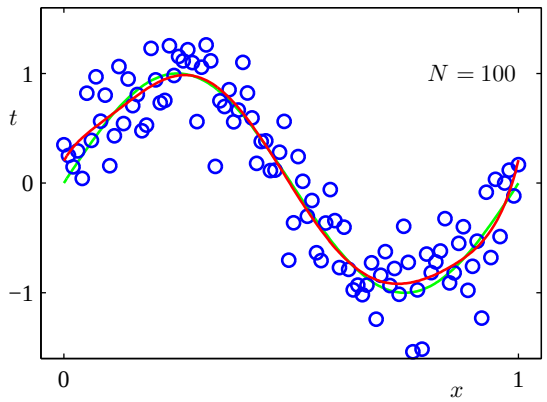


Use more training data to prevent over fitting

The more, the merrier







What if we do not have a lot of data?

Regularization methods

Intuition: for a linear model for regression

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

what do we mean by *being simple*?

Regularization methods

Intuition: for a linear model for regression

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

what do we mean by *being simple*?

Assumptions

$$p(w_d) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{w_d^2}{2\sigma^2}}$$

Namely, *a prior*, we believe w_d is around zero, i.e., resulting in a simple model for prediction.

Note that this line of thinking is to regard $\boldsymbol{w}$ as a random variable and we will use the observed data $\mathcal{D}$ to update our prior belief on $\boldsymbol{w}$

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.

Setup for regularized linear regression

Linear regression

$$y = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} + \eta$$

where η is a Gaussian noise, distributed according to $N(0, \sigma_0^2)$.

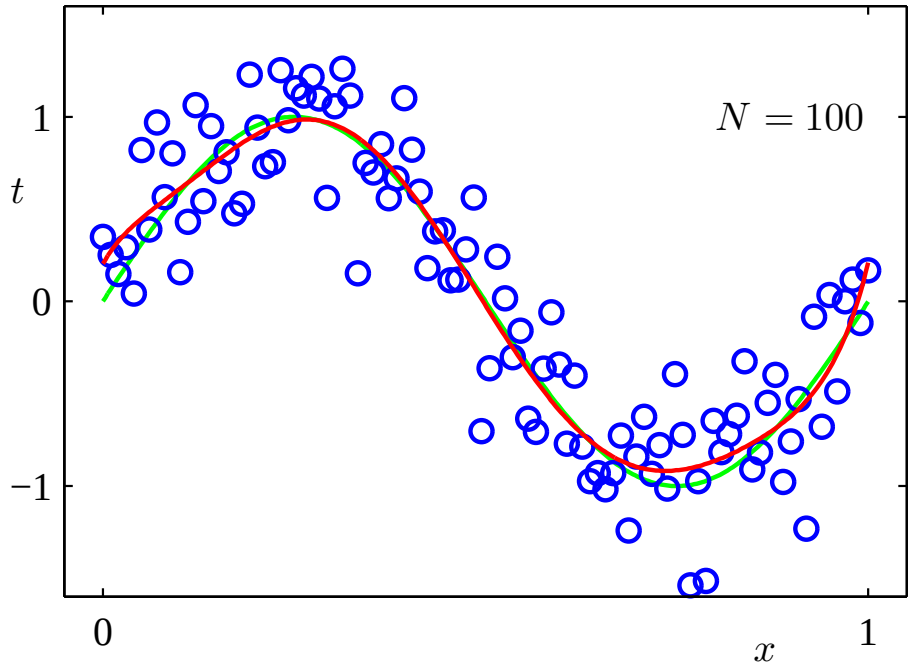
Prior distribution on the parameter

$$p(\boldsymbol{w}) = \prod_{d=1}^{M} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{w_d^2}{2\sigma^2}}$$

Note that all the dimensions share the same variance σ^2.

How to prevent overfitting?

Use more training data



Regularization: adding a term to the objective function

$$\lambda \|\boldsymbol{w}\|_2^2$$

that favors a small parameter vector $\boldsymbol{w}$.

Outline

Estimate $\boldsymbol{w}$

Joint likelihood of both training data and parameter

$$\log p(\mathcal{D}, \boldsymbol{w}) = \sum_n \log p(y_n | \boldsymbol{x}_n, \boldsymbol{w}) + \log p(\boldsymbol{w})$$

$$= -\frac{\sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2}{2\sigma_0^2} - \sum_d \frac{1}{2\sigma^2} w_d^2 + \text{const}$$

where σ_0^2 is the variance of the noise.

Estimate $\boldsymbol{w}$

Joint likelihood of both training data and parameter

$$\log p(\mathcal{D}, \boldsymbol{w}) = \sum_n \log p(y_n|\boldsymbol{x}_n, \boldsymbol{w}) + \log p(\boldsymbol{w})$$
$$= -\frac{\sum_n (\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n - y_n)^2}{2\sigma_0^2} - \sum_d \frac{1}{2\sigma^2} w_d^2 + \text{const}$$

where σ_0^2 is the variance of the noise.

Maximum a posterior (MAP) estimate: we seek to maximize

$$\boldsymbol{w}^{\mathrm{MAP}} = \arg\max_{\boldsymbol{w}} \log p(\boldsymbol{w}|\mathcal{D}) = \arg\max_{\boldsymbol{w}} \log p(\mathcal{D}, \boldsymbol{w})$$

that is, the most likely $\boldsymbol{w}$ *conditioning* on observed training data $\mathcal{D}$.

It is also possible to consider the distribution $p(\boldsymbol{w}|\mathcal{D})$ instead of just its mode. This results in a full Bayesian treatment. More on this later.

Maximum a posterior (MAP) estimate

Regularized linear regression: a new error to minimize

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

where $\lambda > 0$ is used to denote σ_0^2/σ^2. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

Maximum a posterior (MAP) estimate

Regularized linear regression: a new error to minimize

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

where $\lambda > 0$ is used to denote σ_0^2/σ^2. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

Intuitions

- If $\lambda \to +\infty$, then $\sigma_0^2 \gg \sigma^2$. That is, the variance of noise is far greater than what our prior model can allow for $\boldsymbol{w}$. In this case, our prior model on $\boldsymbol{w}$ would be more accurate than what data can tell us. Thus, we are getting a *simple* model. Numerically,

$$\boldsymbol{w}^{\mathrm{MAP}} \to \mathbf{0}$$

Maximum a posterior (MAP) estimate

Regularized linear regression: a new error to minimize

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

where $\lambda > 0$ is used to denote σ_0^2/σ^2. This extra term $\|\boldsymbol{w}\|_2^2$ is called regularization/regularizer and controls the model complexity.

Intuitions

- If $\lambda \to +\infty$, then $\sigma_0^2 \gg \sigma^2$. That is, the variance of noise is far greater than what our prior model can allow for $\boldsymbol{w}$. In this case, our prior model on $\boldsymbol{w}$ would be more accurate than what data can tell us. Thus, we are getting a *simple* model. Numerically,

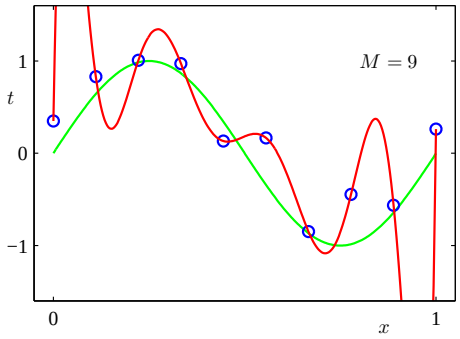
$$\boldsymbol{w}^{\mathrm{MAP}} \to \mathbf{0}$$

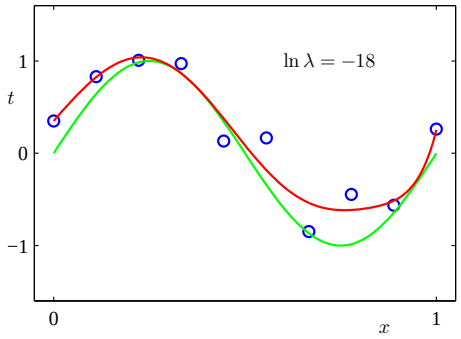
- If $\lambda \to 0$, then we trust our data more. Numerically,

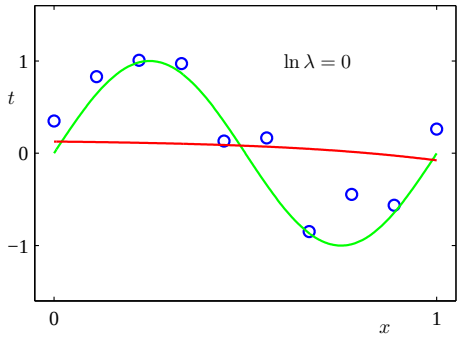
$$\boldsymbol{w}^{\mathrm{MAP}} \to \boldsymbol{w}^{\mathrm{LMS}} = \arg\min \sum_n (\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n - y_n)^2$$

Overfitting in terms of λ

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

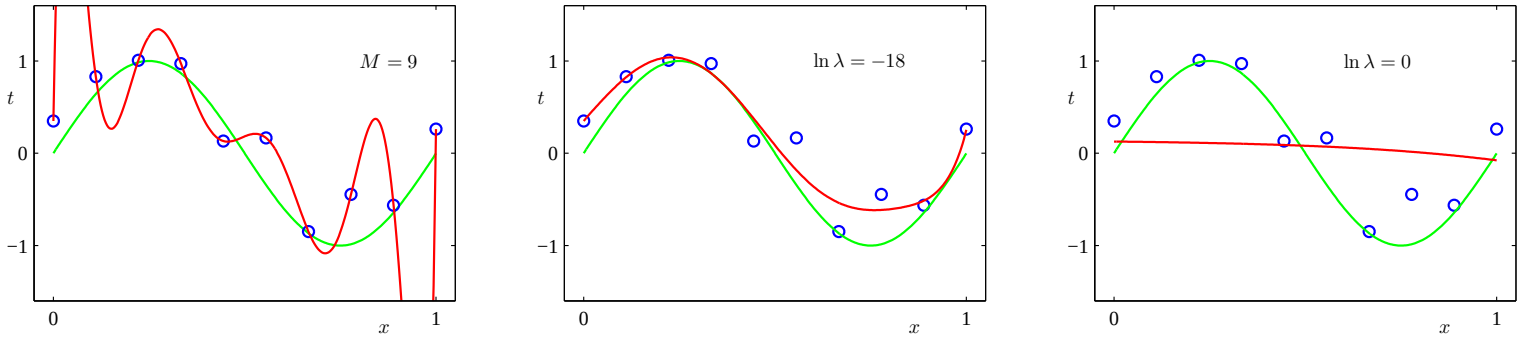




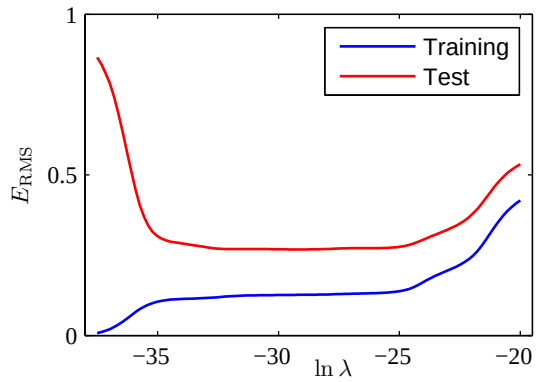


Overfitting in terms of λ

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: the solution changes very little (in form) from the LMS solution

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

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

Closed-form solution

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

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

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

If we have to use numerical procedure, the gradients and the Hessian matrix would change nominally too,

$$\nabla \mathcal{E}(\boldsymbol{w}) = 2(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}\boldsymbol{w} - \boldsymbol{X}^{\mathrm{T}}\boldsymbol{y} + \lambda \boldsymbol{w}), \qquad \boldsymbol{H} = 2(\boldsymbol{X}^{\mathrm{T}}\boldsymbol{X} + \lambda \boldsymbol{I})$$

As long as $\lambda \geq 0$, the optimization is convex.

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

Regularized methods for classification

Adding regularizer to the cross-entropy functions used for binary and multinomial logistic regression

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1-y_n)\log[1-\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\} + \lambda\|\boldsymbol{w}\|_2^2$$

$$\mathcal{E}(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) = -\sum_n \sum_k \log y_{nk} P(C_k|\boldsymbol{x}_n) + \lambda \sum_k \|\boldsymbol{w}_k\|_2^2$$

Regularized methods for classification

Adding regularizer to the cross-entropy functions used for binary and multinomial logistic regression

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1-y_n)\log[1-\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\} + \lambda\|\boldsymbol{w}\|_2^2$$

$$\mathcal{E}(\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_K) = -\sum_n \sum_k \log y_{nk} P(C_k|\boldsymbol{x}_n) + \lambda \sum_k \|\boldsymbol{w}_k\|_2^2$$

Numerical optimization

- Objective functions remain to be convex as long as $\lambda \geq 0$.
- Gradients and Hessians are changed marginally and can be easily derived.

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 $\boldsymbol{w}^{\mathrm{MAP}}$ 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 $\boldsymbol{w}^{\text{MAP}}$ 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.