Linear Regression (continued)

Professor Ameet Talwalkar

Slide Credit: Professor Fei Sha

Outline

Homeworks / Project Proposal

- Graded HW1 will be available to pick up at the end of class
- HW3 and HW4 due next Thursday – start early!
- Guidelines for project proposal will be posted before class on Thursday

Outline

Perceptron Main idea

Consider a linear model for binary classification

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

We use this model to distinguish between two classes $\{-1, +1\}$.

One goal

$$\varepsilon = \sum_n \mathbb{I}[y_n \neq \mathsf{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]$$

i.e., to minimize errors on the training dataset.

Hard, but easy if we have only one training example

How can we change $\boldsymbol{w}$ such that

$$y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$$

Two cases

- If $y_n = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$, do nothing.
- If $y_n \neq \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$, $\boldsymbol{w}^{\mathrm{NEW}} \leftarrow \boldsymbol{w}^{\mathrm{OLD}} + y_n\boldsymbol{x}_n$
  - Gauranteed to make progress as $y_n{\boldsymbol{w}^{\mathrm{NEW}}}^{\mathrm{T}}\boldsymbol{x}_n > y_n{\boldsymbol{w}^{\mathrm{OLD}}}^{\mathrm{T}}\boldsymbol{x}_n$

Perceptron algorithm

Iteratively solving one case at a time

- REPEAT
- Pick a data point $\boldsymbol{x}_n$ (can be a fixed order of the training instances)
- Make a prediction $y = \text{sign}(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)$ using the *current* $\boldsymbol{w}$
- If $y = y_n$, do nothing. Else, $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n \boldsymbol{x}_n$
- UNTIL converged.

Perceptron algorithm

Iteratively solving one case at a time

- REPEAT
- Pick a data point $\boldsymbol{x}_n$ (can be a fixed order of the training instances)
- Make a prediction $y = \text{sign}(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)$ using the *current* $\boldsymbol{w}$
- If $y = y_n$, do nothing. Else, $\boldsymbol{w} \leftarrow \boldsymbol{w} + y_n\boldsymbol{x}_n$
- UNTIL converged.

Properties

- This is an online algorithm.
- If the training data is linearly separable, the algorithm stops in a finite number of steps.
- The parameter vector is always a linear combination of training instances.

Regression

Predicting a continuous outcome variable

- Predicting shoe size from height, weight and gender
- Predicting song year from audio features

Regression

Predicting a continuous outcome variable

- Predicting shoe size from height, weight and gender
- Predicting song year from audio features

Key difference from classification

Regression

Predicting a continuous outcome variable

- Predicting shoe size from height, weight and gender
- Predicting song year from audio features

Key difference from classification

- We can measure 'closeness' of prediction and labels
  - Predicting shoe size: better to be off by one size than by 5 sizes
  - Predicting song year: better to be off by one year than by 20 years
- As opposed to 0-1 classification error, we will focus on squared difference, i.e., $(\hat{y} - y)^2$

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: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$
- Model: $f : \boldsymbol{x} \rightarrow y$, with $f(\boldsymbol{x}) = w_0 + \sum_d w_d x_d = w_0 + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$

  We also sometimes call $\tilde{\boldsymbol{w}} = [w_0\ w_1\ w_2\ \cdots\ w_{\mathsf{D}}]^{\mathrm{T}}$ parameters too!

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: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$
- Model: $f : \boldsymbol{x} \rightarrow y$, with $f(\boldsymbol{x}) = w_0 + \sum_d w_d x_d = w_0 + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$

  We also sometimes call $\tilde{\boldsymbol{w}} = [w_0 \ w_1 \ w_2 \ \cdots \ w_{\mathsf{D}}]^{\mathrm{T}}$ parameters too!

Least Mean Squares (LMS) Objective: Minimize squared difference on training data (or residual sum of squares)

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - f(\boldsymbol{x}_n)]^2 = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2$$

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: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, \mathsf{N}\}$
- Model: $f : \boldsymbol{x} \rightarrow y$, with $f(\boldsymbol{x}) = w_0 + \sum_d w_d x_d = w_0 + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$

  We also sometimes call $\tilde{\boldsymbol{w}} = [w_0 \; w_1 \; w_2 \; \cdots \; w_{\mathsf{D}}]^{\mathrm{T}}$ parameters too!

Least Mean Squares (LMS) Objective: Minimize squared difference on training data (or residual sum of squares)

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - f(\boldsymbol{x}_n)]^2 = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2$$

1D Solution: Identify stationary points by taking derivative with respect to parameters and setting to zero, yielding 'normal equations'

Probabilistic interpretation

- Noisy observation model

$$Y = w_0 + w_1 X + \eta$$

where $\eta \sim N(0, \sigma^2)$ is a Gaussian random variable

Probabilistic interpretation

- Noisy observation model

$$Y = w_0 + w_1 X + \eta$$

where $\eta \sim N(0, \sigma^2)$ is a Gaussian random variable

- Likelihood of one training sample (x_n, y_n)

$$p(y_n|x_n) = N(w_0 + w_1 x_n, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{[y_n - (w_0 + w_1 x_n)]^2}{2\sigma^2}}$$

Maximum likelihood estimation

- Maximize over w_0 and w_1

$$\max \ \log P(\mathcal{D}) \Leftrightarrow \min \ \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

Maximum likelihood estimation

- Maximize over w_0 and w_1

$$\max \log P(\mathcal{D}) \Leftrightarrow \min \sum_n [y_n - (w_0 + w_1 x_n)]^2 \leftarrow \text{That is RSS}(\tilde{\boldsymbol{w}})!$$

- Maximize over $s = \sigma^2$

$$\frac{\partial \log P(\mathcal{D})}{\partial s} = -\frac{1}{2}\left\{ -\frac{1}{s^2}\sum_n [y_n - (w_0 + w_1 x_n)]^2 + \mathsf{N}\frac{1}{s} \right\} = 0$$

$$\rightarrow \sigma^{*2} = s^* = \frac{1}{\mathsf{N}} \sum_n [y_n - (w_0 + w_1 x_n)]^2$$

How does this probabilistic interpretation help us?

How does this probabilistic interpretation help us?

- It gives a solid footing to our intuition: minimizing $\text{RSS}(\tilde{\boldsymbol{w}})$ is a sensible thing based on reasonable modeling assumptions
- Estimating σ^* tells us how much noise there could be in our predictions. For example, it allows us to place confidence intervals around our predictions.

Outline

LMS when $\boldsymbol{x}$ is D-dimensional

$RSS(\tilde{\boldsymbol{w}})$ in matrix form

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2 = \sum_n [y_n - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n]^2$$

where we have redefined some variables (by augmenting)

$$\tilde{\boldsymbol{x}} \leftarrow [1\ x_1\ x_2\ \ldots\ x_{\mathsf{D}}]^{\mathrm{T}}, \quad \tilde{\boldsymbol{w}} \leftarrow [w_0\ w_1\ w_2\ \ldots\ w_{\mathsf{D}}]^{\mathrm{T}}$$

LMS when $\boldsymbol{x}$ is D-dimensional

$RSS(\tilde{\boldsymbol{w}})$ in matrix form

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2 = \sum_n [y_n - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n]^2$$

where we have redefined some variables (by augmenting)

$$\tilde{\boldsymbol{x}} \leftarrow [1 \ x_1 \ x_2 \ \ldots \ x_{\mathsf{D}}]^{\mathrm{T}}, \quad \tilde{\boldsymbol{w}} \leftarrow [w_0 \ w_1 \ w_2 \ \ldots \ w_{\mathsf{D}}]^{\mathrm{T}}$$

which leads to

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n (y_n - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n)(y_n - \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}})$$

LMS when $\boldsymbol{x}$ is D-dimensional

$RSS(\tilde{\boldsymbol{w}})$ in matrix form

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2 = \sum_n [y_n - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n]^2$$

where we have redefined some variables (by augmenting)

$$\tilde{\boldsymbol{x}} \leftarrow [1 \ x_1 \ x_2 \ \ldots \ x_{\mathsf{D}}]^{\mathrm{T}}, \quad \tilde{\boldsymbol{w}} \leftarrow [w_0 \ w_1 \ w_2 \ \ldots \ w_{\mathsf{D}}]^{\mathrm{T}}$$

which leads to

$$\begin{aligned} RSS(\tilde{\boldsymbol{w}}) &= \sum_n (y_n - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n)(y_n - \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}}) \\ &= \sum_n \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - 2 y_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} + \text{const.} \end{aligned}$$

LMS when $\boldsymbol{x}$ is D-dimensional

$RSS(\tilde{\boldsymbol{w}})$ in matrix form

$$RSS(\tilde{\boldsymbol{w}}) = \sum_n [y_n - (w_0 + \sum_d w_d x_{nd})]^2 = \sum_n [y_n - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n]^2$$

where we have redefined some variables (by augmenting)

$$\tilde{\boldsymbol{x}} \leftarrow [1 \ x_1 \ x_2 \ \ldots \ x_{\mathsf{D}}]^{\mathrm{T}}, \quad \tilde{\boldsymbol{w}} \leftarrow [w_0 \ w_1 \ w_2 \ \ldots \ w_{\mathsf{D}}]^{\mathrm{T}}$$

which leads to

$$\begin{aligned} RSS(\tilde{\boldsymbol{w}}) &= \sum_n (y_n - \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n)(y_n - \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}}) \\ &= \sum_n \tilde{\boldsymbol{w}}^{\mathrm{T}} \tilde{\boldsymbol{x}}_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} - 2 y_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \tilde{\boldsymbol{w}} + \text{const.} \\ &= \left\{ \tilde{\boldsymbol{w}}^{\mathrm{T}} \left(\sum_n \tilde{\boldsymbol{x}}_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \right) \tilde{\boldsymbol{w}} - 2 \left(\sum_n y_n \tilde{\boldsymbol{x}}_n^{\mathrm{T}} \right) \tilde{\boldsymbol{w}} \right\} + \text{const.} \end{aligned}$$

Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} 9 & 3 & 5 \\ 4 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} 9 & 3 & 5 \\ 4 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} \boxed{9 \quad 3 \quad 5} \\ 4 \quad 1 \quad 2 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} 1 \\ 3 \\ 2 \end{matrix}} & \begin{matrix} 2 \\ -5 \\ 3 \end{matrix} \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

$$9 \times 1 + 3 \times 3 + 5 \times 2 = 28$$

Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} \boxed{9 \quad 3 \quad 5} \\ 4 \quad 1 \quad 2 \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} 1 \\ 3 \\ 2 \end{matrix}} & \begin{matrix} 2 \\ -5 \\ 3 \end{matrix} \end{bmatrix} = \begin{bmatrix} 28 & \\ & \end{bmatrix}$$

$$9 \times 1 + 3 \times 3 + 5 \times 2 = 28$$

Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} \boxed{9 \quad 3 \quad 5} \\ 4 \quad 1 \quad 2 \end{bmatrix} \begin{bmatrix} 1 & \boxed{\begin{matrix} 2 \\ -5 \\ 3 \end{matrix}} \\ 3 & \\ 2 & \end{bmatrix} = \begin{bmatrix} 28 & \\ & \end{bmatrix}$$

Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} \boxed{9 & 3 & 5} \\ 4 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & \boxed{\begin{matrix} 2 \\ -5 \\ 3 \end{matrix}} \\ 3 & \\ 2 & \end{bmatrix} = \begin{bmatrix} 28 & 18 \\ & \end{bmatrix}$$

Matrix Multiplication via Inner Products

Each entry of output matrix is result of inner product of inputs matrices

$$\begin{bmatrix} \boxed{9 \quad 3 \quad 5} \\ 4 \quad 1 \quad 2 \end{bmatrix} \begin{bmatrix} 1 & \boxed{\begin{matrix} 2 \\ -5 \\ 3 \end{matrix}} \\ 3 & \\ 2 & \end{bmatrix} = \begin{bmatrix} 28 & 18 \\ 11 & 9 \end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & 3 & 5 \\ 4 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} \boxed{\begin{matrix} 9 \\ 4 \end{matrix}} & \begin{matrix} 3 \\ 1 \end{matrix} & \begin{matrix} 5 \\ 2 \end{matrix} \end{bmatrix} \begin{bmatrix} \boxed{\begin{matrix} 1 & 2 \end{matrix}} \\ \begin{matrix} 3 & -5 \end{matrix} \\ \begin{matrix} 2 & 3 \end{matrix} \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} \boxed{\begin{matrix} 9 \\ 4 \end{matrix}} & \begin{matrix} 3 \\ 1 \end{matrix} & \begin{matrix} 5 \\ 2 \end{matrix} \end{bmatrix} \begin{bmatrix} \boxed{1 \quad 2} \\ 3 \quad -5 \\ 2 \quad 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & \boxed{\begin{matrix} 3 \\ 1 \end{matrix}} & 5 \\ 4 & & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ \boxed{3 \quad -5} \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & \boxed{\begin{matrix} 3 \\ 1 \end{matrix}} & 5 \\ 4 & & 2 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ \boxed{3 \quad -5} & \\ 2 & 3 \end{bmatrix} = \begin{bmatrix} \quad & \quad \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix} + \begin{bmatrix} 9 & -15 \\ 3 & -5 \end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\left[\begin{matrix}9 & 3\\ 4 & 1\end{matrix}\;\boxed{\begin{matrix}5\\ 2\end{matrix}}\right]\left[\begin{array}{c}\begin{matrix}1 & 2\\ 3 & -5\end{matrix}\\ \boxed{\begin{matrix}2 & 3\end{matrix}}\end{array}\right]=\begin{bmatrix} & \\ & \end{bmatrix}$$

$$\begin{bmatrix}9 & 18\\ 4 & 8\end{bmatrix}+\begin{bmatrix}9 & -15\\ 3 & -5\end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & 3 & \boxed{\begin{matrix} 5 \\ 2 \end{matrix}} \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ \boxed{2 \quad 3} \end{bmatrix} = \begin{bmatrix} & \\ & \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix} + \begin{bmatrix} 9 & -15 \\ 3 & -5 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 4 & 6 \end{bmatrix}$$

Matrix Multiplication via Outer Products

Output matrix is **sum of outer products** between corresponding rows and columns of input matrices

$$\begin{bmatrix} 9 & 3 & \boxed{\begin{matrix} 5 \\ 2 \end{matrix}} \\ 4 & 1 & \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 3 & -5 \\ \boxed{2 \quad 3} \end{bmatrix} = \begin{bmatrix} 28 & 18 \\ 11 & 9 \end{bmatrix}$$

$$\begin{bmatrix} 9 & 18 \\ 4 & 8 \end{bmatrix} + \begin{bmatrix} 9 & -15 \\ 3 & -5 \end{bmatrix} + \begin{bmatrix} 10 & 15 \\ 4 & 6 \end{bmatrix}$$

$RSS(\tilde{\boldsymbol{w}})$ in new notations

Design matrix and target vector

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

$RSS(\tilde{\boldsymbol{w}})$ in new notations

Design matrix and target vector

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

Compact expression

$$RSS(\tilde{\boldsymbol{w}}) = ||\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}||_2^2 = \left\{ \tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - 2\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\tilde{\boldsymbol{w}} \right\} + \text{const}$$

Solution in matrix form

Compact expression

$$RSS(\tilde{\boldsymbol{w}}) = ||\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}||_2^2 = \left\{\tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - 2\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\tilde{\boldsymbol{w}}\right\} + \text{const}$$

Gradients of Linear and Quadratic Functions

- $\nabla_{\boldsymbol{x}}\boldsymbol{b}^\top\boldsymbol{x} = \boldsymbol{b}$
- $\nabla_{\boldsymbol{x}}\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} = 2\boldsymbol{A}\boldsymbol{x}$ (symmetric $\boldsymbol{A}$)

Solution in matrix form

Compact expression

$$RSS(\tilde{\boldsymbol{w}}) = ||\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}||_2^2 = \left\{\tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - 2\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\tilde{\boldsymbol{w}}\right\} + \text{const}$$

Gradients of Linear and Quadratic Functions

- $\nabla_{\boldsymbol{x}}\boldsymbol{b}^\top\boldsymbol{x} = \boldsymbol{b}$
- $\nabla_{\boldsymbol{x}}\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} = 2\boldsymbol{A}\boldsymbol{x}$ (symmetric $\boldsymbol{A}$)

Normal equation

$$\nabla_{\tilde{\boldsymbol{w}}}RSS(\tilde{\boldsymbol{w}}) \propto \tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\boldsymbol{w} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = 0$$

Solution in matrix form

Compact expression

$$RSS(\tilde{\boldsymbol{w}}) = ||\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - \boldsymbol{y}||_2^2 = \left\{\tilde{\boldsymbol{w}}^{\mathrm{T}}\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}} - 2\left(\tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}\right)^{\mathrm{T}}\tilde{\boldsymbol{w}}\right\} + \text{const}$$

Gradients of Linear and Quadratic Functions

- $\nabla_{\boldsymbol{x}}\boldsymbol{b}^{\top}\boldsymbol{x} = \boldsymbol{b}$
- $\nabla_{\boldsymbol{x}}\boldsymbol{x}^{\top}\boldsymbol{A}\boldsymbol{x} = 2\boldsymbol{A}\boldsymbol{x}$ (symmetric $\boldsymbol{A}$)

Normal equation

$$\nabla_{\tilde{\boldsymbol{w}}} RSS(\tilde{\boldsymbol{w}}) \propto \tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\boldsymbol{w} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y} = 0$$

This leads to the least-mean-square (LMS) solution

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

Mini-Summary

- Linear regression is the linear combination of features $f : \boldsymbol{x} \rightarrow y$, with $f(\boldsymbol{x}) = w_0 + \sum_d w_d x_d = w_0 + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$
- If we minimize residual sum of squares as our learning objective, we get a closed-form solution of parameters
- Probabilistic interpretation: maximum likelihood if assuming residual is Gaussian distributed

Computational complexity

Bottleneck of computing the solution?

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

Computational complexity

Bottleneck of computing the solution?

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

Matrix multiply of $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} \in \mathbb{R}^{(\mathsf{D}+1)\times(\mathsf{D}+1)}$
Inverting the matrix $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$

How many operations do we need?

Computational complexity

Bottleneck of computing the solution?

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

Matrix multiply of $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}} \in \mathbb{R}^{(\mathsf{D}+1)\times(\mathsf{D}+1)}$
Inverting the matrix $\tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}$

How many operations do we need?

- $O(\mathsf{ND}^2)$ for matrix multiplication
- $O(\mathsf{D}^3)$ (e.g., using Gauss-Jordan elimination) or $O(\mathsf{D}^{2.373})$ (recent theoretical advances) for matrix inversion
- Impractical for very large D or N

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (e.g., randomly); set $t = 0$; choose $\eta > 0$

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (e.g., randomly); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}} \tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}} \boldsymbol{y}$

Alternative method: an example of using numerical optimization

(Batch) Gradient descent

- Initialize $\tilde{\boldsymbol{w}}$ to $\tilde{\boldsymbol{w}}^{(0)}$ (e.g., randomly); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient
     $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$
  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 $\tilde{\boldsymbol{w}}^{(0)}$ (e.g., randomly); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$
  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 $\tilde{\boldsymbol{w}}^{(0)}$ (e.g., randomly); set $t = 0$; choose $\eta > 0$
- Loop *until convergence*
  1. Compute the gradient $\nabla RSS(\tilde{\boldsymbol{w}}) = \tilde{\boldsymbol{X}}^{\mathrm{T}}\tilde{\boldsymbol{X}}\tilde{\boldsymbol{w}}^{(t)} - \tilde{\boldsymbol{X}}^{\mathrm{T}}\boldsymbol{y}$
  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 of each iteration?

Why would this work?

Why would this work?

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

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

Hessian of RSS

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

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

Why would this work?

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

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

Hessian of RSS

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

$\tilde{\boldsymbol{X}}^{\mathrm{T}} \tilde{\boldsymbol{X}}$ 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

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

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

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

  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

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

  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

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

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

How does the complexity per iteration compare with gradient descent?

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

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

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

How does the complexity per iteration compare with gradient descent?

- O(ND) for gradient descent versus O(D) for SGD

Mini-summary

- Batch gradient descent computes the exact gradient.
- Stochastic gradient descent approximates the gradient with a single data point; Its expectation equals the true gradient.
- Mini-batch variant: trade-off between accuracy of estimating gradient and computational cost
- Similar ideas extend to other ML optimization problems.
    - 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 $\tilde{X}^{\mathrm{T}}\tilde{X}$ is not invertible

Can you think of any reasons why that could happen?

Answer 1: $\mathrm{N} < \mathrm{D}$. 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? Consider the SVD of this matrix:

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

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? Consider the SVD of this matrix:

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

where $\lambda_1 \geq \lambda_2 \geq \cdots \lambda_r > 0$ and $r <$ D.

Fix the problem by ensuring all singular values are non-zero

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

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?

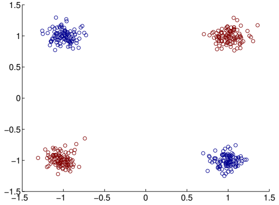
λ is referred as *hyperparameter*

- In contrast $\boldsymbol{w}$ is the parameter vector
- Use validation or cross-validation to find good choice of λ

Outline

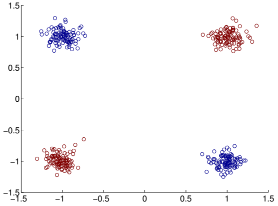
What if data is not linearly separable or fits to a line

Example of nonlinear classification

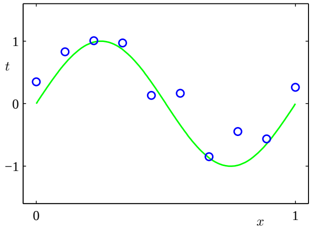


What if data is not linearly separable or fits to a line

Example of nonlinear classification



Example of nonlinear regression



Nonlinear basis for classification

Transform the input/feature

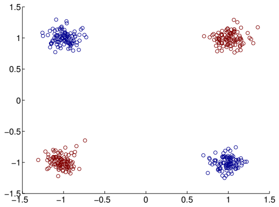
$$\phi(\boldsymbol{x}) : \boldsymbol{x} \in \mathbb{R}^2 \rightarrow z = x_1 \cdot x_2$$

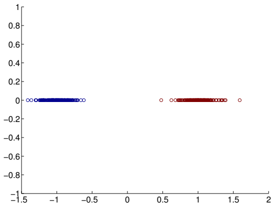
Nonlinear basis for classification

Transform the input/feature

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

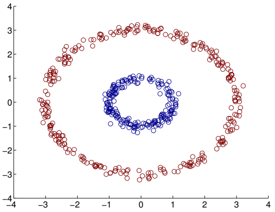
Transformed training data: linearly separable!



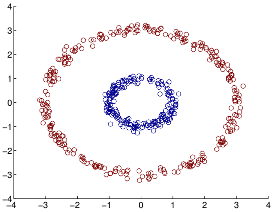


Another example

How to transform the input/feature?



Another example

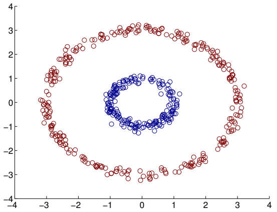


How to transform the input/feature?

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

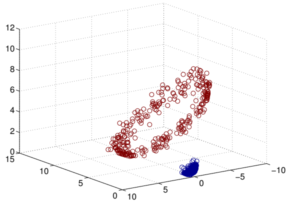
Another example

How to transform the input/feature?



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

Transformed training data: linearly separable



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

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

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 to minimize our errors on the transformed training data

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

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

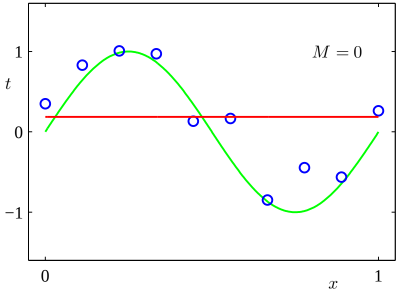
$$\boldsymbol{\phi}(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^M \end{bmatrix} \Rightarrow f(x) = w_0 + \sum_{m=1}^{M} w_m x^m$$

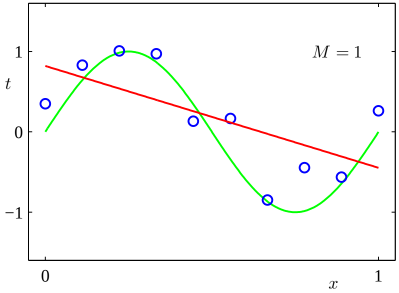
Example with regression

Polynomial basis functions

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

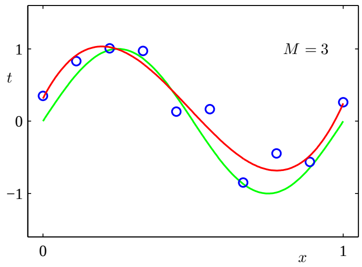
Fitting samples from a sine function: *underrfitting* as $f(x)$ is too simple





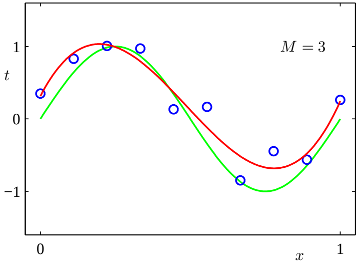
Adding high-order terms

M=3

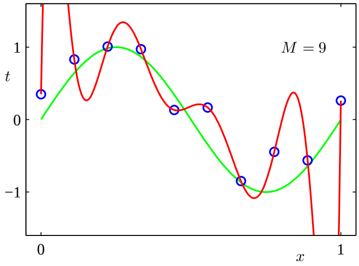


Adding high-order terms

M=3



M=9: *overfitting*



More complex features lead to better results on the training data, but potentially worse results on new data, e.g., test data!

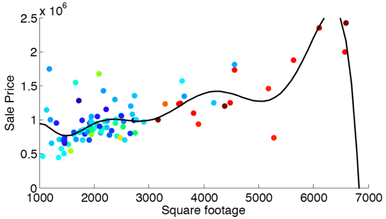
Overfiting

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 $M = 3$

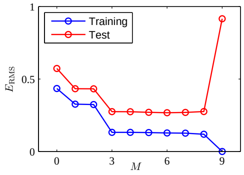


Note that the price would goes to zero (or negative) if you buy bigger ones!
This is called poor generalization/overfitting.

Detecting overfitting

Plot model complexity versus objective function

As a model increases in complexity, performance on training data keeps improving while performance on test data may first improve but eventually deteriorate.

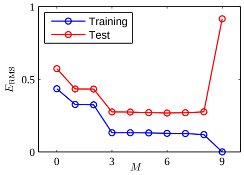


- Horizontal axis: *measure of model complexity*; in this example complexity defined by order of the polynomial basis functions.

Detecting overfitting

Plot model complexity versus objective function

As a model increases in complexity, performance on training data keeps improving while performance on test data may first improve but eventually deteriorate.



- Horizontal axis: *measure of model complexity*; in this example complexity defined by order of the polynomial basis functions.
- Vertical axis:
  1. For regression, RSS or RMS (squared root of RSS)
  2. For classification, classification error rate or cross-entropy error function