



中国科学技术大学  
University of Science and Technology of China

# 人工智能讲义

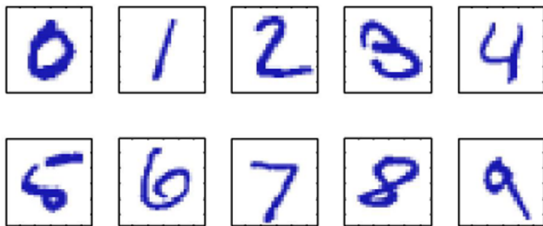
## 线性回归

April 7, 2022

# Outline

- ① 从例子开始
- ② 损失函数
- ③ 最小化损失

## $h$ 如何获得



### 手写体数字识别：问题描述

- 每个手写体字符  $x$  是  $100 \times 100$  的像素点方阵，每个像素点取值 0 或 1，也就是说每个输入变量是长度为  $100 \times 100$  的 0-1 方阵（二维矩阵）
- 每个手写体字符的输出离散化为 0,1,2,3,4,5,6,7,8,9。
- 我们也可以将输出定义为任意实数  $y$ ，然后视  $y$  和这 10 个数字字符的远近做离散化近似。如  $y = 2.4$  时， $y$  可以离散化为 2； $y = -20$  时， $y$  可以离散化为 0 等

## $h$ 如何获得

### 手写体数字识别问题分析 1: $f$ 是什么?

- 人眼看到输入, 人脑对输出做出判决;
- 该  $f$  目前无法用数学函数来显示表达;
- 该  $f$  由视觉/光学/神经信息处理等等一系列物理过程构成, 非常复杂

### 手写体数字识别问题分析 2: $h$ 是什么?

- 用最简的线性函数  $h(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + b$  来近似  $f$
- 公式中的点表示两个矩阵的点积: 两个矩阵对应位置的元素相乘然后全部加起来
- 其中  $\mathbf{W}$  是  $100 \times 100$  的实数矩阵, 对图片  $x$  的每个像素点给一个权值, 描述其在识别过程中的作用。每个像素点对每个字符的贡献可能会不同/有冲突, 权值该如何设置?

## $h$ 如何获得

$$h(\mathbf{x}) = \mathbf{W} \cdot \mathbf{x} + b \sim f$$

手写体数字识别：解决问题 搜索最优的  $\mathbf{W}$  和  $b$

- 线性函数  $h(\mathbf{x})$  由其系数矩阵  $\mathbf{W}$  和常数项  $b$  确定，这二者称为线性函数  $h(\mathbf{x})$  的参数
- 不同参数得到的不同  $h(\mathbf{x})$ ，最后识别字符时，准确率会有差异，找到使得差异最小的那组最优参数
- 找最优参数的方法：最小二乘法

## 输入变量 $\mathbf{x}$ 的预处理

$$h(x) = \mathbf{W} \cdot \mathbf{x} + b \Rightarrow h(x) = \mathbf{W} \cdot \phi(x)$$

- 完整的线性函数应该包括常数项  $b$
- 但是，我们把输入变量  $\mathbf{x}$  做一个简单的变换， $\mathbf{x} = (x_1, x_2, \dots, x_n)^t \longrightarrow \mathbf{x} = (x_1, x_2, \dots, x_n, 1)^t$ ，就可以消去线性函数中的常数项  $b$
- 更一般地，我们将输入变量  $x$ ，做一次变换，即  $x \longrightarrow \phi(x) = (\phi_1(x), \phi_2(x), \dots, \phi_n(x))^t$ ，不管  $x$  是几维变量，经过一次变换，成为  $n$  维向量，我们称变换后的结果为“输入特征向量”，该变换称为“特征提取函数”。
- 故线性  $h$  的表达式  $h(x) = \mathbf{W} \cdot \mathbf{x}$  就变成  $h(x) = \mathbf{W} \cdot \phi(x)$ ，以之作为我们讨论线性  $h$  的标准表达式

## $h$ 的获得

### 从评价标准：准确性出发

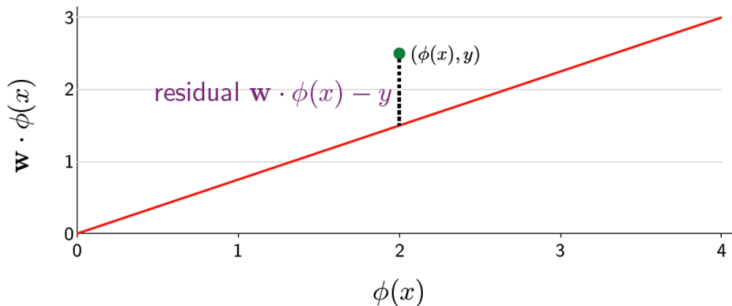
- 从分析事情要达到的目标出发，反向思考，我们应怎么去找  $h$
- 我们称  $h(x)$  和  $f(x)$  在某个输入  $x$  上值不一样，为“误差”或“损失”
  - 绝对损失： $loss_1(h, f, x) = |f(x) - h(x)|$ ，适用于连续  $y$  值情形
  - 平方损失： $loss_2(h, f, x) = (f(x) - h(x))^2$ ，适用于连续  $y$  值情形
  - 计数损失： $loss_0(h, f, x) = 1[f(x) \neq h(x)]$ ，适用于离散  $y$  值情形

### 线性 $h$ 的“一行”损失函数

- 所谓“一行”是指用表格描述  $h$ ，一行代表  $x$  的某一个取值；有时也称一个数据的损失。
- $h$  为线性函数时，损失函数如下：
  - 绝对损失： $loss_1(\mathbf{W}, f, x) = |f(x) - h(x)| = |\mathbf{W} \cdot \phi(x) - f(x)|$
  - 平方损失： $loss_2(\mathbf{W}, f, x) = (f(x) - h(x))^2 = (\mathbf{W} \cdot \phi(x) - f(x))^2$
  - 计数损失： $loss_0(\mathbf{W}, f, x) = 1[f(x) \neq \mathbf{W} \cdot \phi(x)]$

以下我们仅考虑线性  $h$  的情形。

## 残差

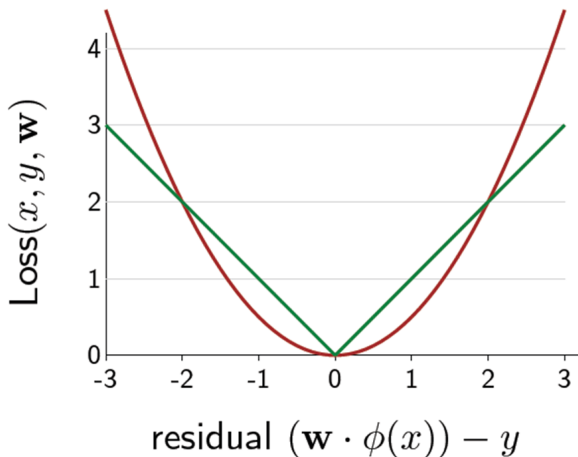


若令  $y = f(x)$ ,

- 我们定义:  $\mathbf{W} \cdot \phi(x) - y$  为残差/residual
- 例子如上图所示, 绿点为真实数据, 红线为我们找到的近似线性  $h$ , 黑色虚线长度表示了残差的大小,
- 残差有正负



## 基于残差定义损失函数



### 解释说明

- 红色曲线是平方残差损失函数,  $\text{loss}_2(\mathbf{W}, f, x) = (f(x) - h(x))^2 = (\mathbf{W} \cdot \phi(x) - f(x))^2$
- 绿色曲线是绝对值残差损失函数,  $\text{loss}_1(\mathbf{W}, f, x) = |f(x) - h(x)| = |\mathbf{W} \cdot \phi(x) - f(x)|$

## $x$ 定义域上所有损失之和

或者说完整映射表  $h$  中，所有行的损失之和

- 以平方损失为例：

$$\sum_x \text{loss}_2(\mathbf{W}, f, x) = (f(x) - h(x))^2 = (\mathbf{W} \cdot \phi(x) - f(x))^2$$

- 仅具备理论价值，是我们评价  $h$  近似  $f$  的准确性的最理想标准
- 然后计算的时空代价使得其不实用。

为什么要把各个不同  $x$  的损失加起来？

- 通常  $x$  的取值数目是很多的，超过了  $W$  中的未知参数个数，也就是若每个  $x$ （称为样本）都要求是损失为 0，那么每个样本数据就是一个线性方程，整个方程组很有可能无解，因此必然有部分/全部样本损失不为 0，所以用求和来代替，看整体效应。

# 训练集上的所有损失之和

## 实用的评价标准

- $TrainLoss(D_{train}, \mathbf{W}) = \frac{\sum_{(x,y) \in D_{train}} loss(x,y,\mathbf{W})}{|D_{train}|}$
- 也称为 “训练误差”

## 训练集上为什么会有损失？

- 模型/假设  $h$  不准确，例如线性  $h$  太简单了，几乎不能描述世上绝大多数的事物。
- 训练数据集有噪声（如测量不准确，错误数据等）。

## 测试集上的所有损失之和

也称“经验风险”

- $TestLoss(D_{test}, \mathbf{W}) = \frac{\sum_{(x,y) \in D_{test}} loss(x,y,\mathbf{W})}{|D_{test}|}$
- 通常用测试集上的所有损失来作为  $h$  的评价标准。

如何降低训练集和测试集上的损失？

- 如何最小化损失之和？
- 测试集不能被直接使用，只能用在最后的评估过程中，所以我们只能尽可能最小化训练集上的误差。

# 最小化训练集上的损失之和

## 涉及的主要工作

- 选择恰当的损失函数，即确定一个数据/一行/一个样本的损失函数的定义
- 求解最小化  $TrainLoss(D_{train}, \mathbf{W}) = \frac{\sum_{(x,y) \in D_{train}} loss(x,y,\mathbf{W})}{|D_{train}|}$  问题

## 最小化问题分析

- 已知量：  $D_{train}$ ，残缺的表格，样本数据集，训练数据集，每一个数据可表示为  $(x_i, y_i)$
- 未知量：  $\mathbf{W}$ ，线性  $h$  的系数，也是我们最小化  $TrainLoss(\cdot)$  所需要调整的、搜索的变量。
- 寻找最优的  $\mathbf{W}$ ，使得  $TrainLoss(\cdot)$  最小。寻找一个  $\mathbf{W}$ ，减少某个样本的残差，容易；寻找一个  $\mathbf{W}$ ，减少所有样本上的残差，困难，可能不同的样本对  $\mathbf{W}$  的要求有冲突。

# 损失函数的选择

## 考量的方面

- 平方损失函数：近似寻找  $y$  的均值，综合所有样本的影响
- 绝对损失函数：近似寻找  $y$  的中值，抵抗离群点/样本的影响

## 当采用平方损失函数时，

- $TrainLoss(\cdot) = \sum loss_2(\mathbf{W}, f, x) = (f(x) - h(x))^2 = \sum (\mathbf{W} \cdot \phi(x) - f(x))^2$
- 即为著名的 **最小二乘法**

# 最小化问题求解算法

## 梯度下降法

- 算法略。(参考“优化问题求解”一章)

## 算法时间复杂度

- 算法的每一步，即每一次循环，更新一次  $W$ ，都需要完全扫描一次训练数据集，即：
$$W_{s+1} = W_s - \lambda_s \nabla_W \left( \frac{\sum_{(x,y) \in D_{train}} (W_s \cdot \phi(x) - y)^2}{|D_{train}|} \right) = W_s - \lambda_s \frac{\sum_{(x,y) \in D_{train}} 2(W_s \cdot \phi(x) - y)\phi(x)}{|D_{train}|}$$
- 当训练数据集很大时，包括千万/上亿的行时，算法的时间开销巨大。
- 数据挖掘工程实践中常考虑这类问题。此时训练数据集太大，无法一次载入内存，涉及内外存数据交换，时间开销更大，因此需要对算法进行改进。

# 最小化问题求解算法

## 随机梯度下降法

- 计算梯度方向时，只用了一个随机样本的信息，替代经典梯度下降算法要计算所有训练数据集对梯度方向的贡献。
- 即： 
$$W_{s+1} = W_s - \lambda_s \frac{\sum_{(x,y) \in D_{train}} 2(W_s \cdot \phi(x) - y)\phi(x)}{|D_{train}|} \implies W_{s+1} = W_s - \lambda_s \nabla_W ((W_s \cdot \phi(x_i) - y_i)^2)$$

时间性能得到提升，不需要每次循环都载入一次训练数据集；付出的代价是更多的循环次数。