

**From:** FANG Changrui

**Subject:** Lecture Notes for Stanford CS229 Machine Learning

**Date:** from October 11, 2024 to January 5, 2026

---

## Contents

<b>1 Lecture 1: Introduction</b>	<b>4</b>
1.1 Supervised Learning . . . . .	4
1.2 Unsupervised Learning . . . . .	4
1.3 Reinforcement Learning . . . . .	5
<b>2 Lecture 2: Supervised learning setup</b>	<b>6</b>
2.1 Definitions . . . . .	6
2.2 Linear Regression . . . . .	6
<b>3 Lecture 3: Weighted Least Squares, Logistic regression, Newton's Method</b>	<b>10</b>
3.1 Introduction . . . . .	10
3.2 Probabilistic View of Linear Regression . . . . .	10
3.3 Classification . . . . .	11
3.4 Newton's Methods . . . . .	13
<b>4 Lecture 4: Exponential family, Generalized Linear Models</b>	<b>15</b>
4.1 Exponential Family . . . . .	15
4.2 Generalized Linear Models . . . . .	16
4.3 Multi Classification via Softmax . . . . .	18
<b>5 Lecture 5: Gaussian discriminant analysis, Naive Bayes</b>	<b>20</b>
5.1 Introduction . . . . .	20
5.2 Gaussian Discriminative analysis(GDA) . . . . .	21
5.3 Gaussian Discriminative analysis(GDA) – 2-Classification Case . . . . .	21
5.4 Summary . . . . .	24
<b>6 Lecture 6: Naive Bayes, Laplace Smoothing</b>	<b>25</b>
6.1 Introduction . . . . .	25
6.2 Naive Bayes . . . . .	25
6.3 Laplace Smoothing . . . . .	27
<b>7 Lecture 7: Kernels</b>	<b>28</b>
7.1 Feature Function . . . . .	28
7.2 Kernel Trick / Kernelized . . . . .	28
7.3 Design Kernel Function . . . . .	30
7.4 Extended content . . . . .	31
<b>8 Lecture 8: Neural Networks</b>	<b>32</b>
8.1 Outline . . . . .	32
8.2 Review . . . . .	32
8.3 Neural network . . . . .	32
8.4 Questions . . . . .	36

<b>9 Lecture 9: Backprop</b>	<b>37</b>
9.1 Outline . . . . .	37
9.2 Review . . . . .	37
9.3 Differentiable circuits / networks . . . . .	37
9.3.1 Preliminary – Chain rule . . . . .	38
9.3.2 Back propagation . . . . .	38
<b>10 Lecture 10: Bias - Variance, Regularization</b>	<b>41</b>
10.1 Introduction . . . . .	41
10.2 Bias-Variance Theory . . . . .	41
10.3 Double decent . . . . .	43
<b>11 Lecture 11: Feature / Model selection, ML Advice</b>	<b>44</b>
11.1 Complexity Measure . . . . .	44
11.2 Implicit Regularization . . . . .	45
11.3 Validation . . . . .	46
11.4 Machine Learning Advice . . . . .	46
<b>12 Lecture 12: K-Means, GMM(non EM), Expectation Maximization</b>	<b>47</b>
12.1 Introduction . . . . .	47
12.2 K-Means . . . . .	47
12.3 Mixture of Gaussian . . . . .	49
<b>13 Lecture 13: GMM (EM)</b>	<b>51</b>
13.1 Introduction . . . . .	51
13.2 Convexity and Jensen's inequality . . . . .	51
13.3 EM Algorithm as <i>MLE</i> . . . . .	52
13.4 EM for GMM . . . . .	54
<b>14 Lecture 14: Factor Analysis / PCA</b>	<b>56</b>
14.1 Introduction . . . . .	56
14.2 Factor Analysis . . . . .	56
14.3 Principle Component Analysis (PCA) . . . . .	59
<b>15 Lecture 15: PCA / ICA</b>	<b>61</b>
15.1 Introduction . . . . .	61
15.2 Principle Component Analysis . . . . .	61
15.3 Independent Component Analysis (ICA) . . . . .	62
<b>16 Lecture 16: Self-supervised learning</b>	<b>65</b>
16.1 Introduction . . . . .	65
16.2 Self-supervised Learning . . . . .	65
16.3 Contrastive Learning . . . . .	66
16.4 Large Language Model . . . . .	67
<b>A Generalization</b>	<b>69</b>
<b>B Gaussian Distribution</b>	<b>69</b>
<b>C Gradient of loss function</b>	<b>69</b>
<b>D Python codes for plotting Gaussian distribution</b>	<b>70</b>
<b>E Proof of Example 1</b>	<b>71</b>

<b>F Proof of Example 2</b>	<b>71</b>
<b>G Proof of Theorem 1</b>	<b>72</b>
<b>H Multivariate Gaussian Distribution</b>	<b>73</b>
<b>I Proof of the solutions of GDA(2-classification case)</b>	<b>74</b>
<b>J Proof of the decision boundary (38)</b>	<b>75</b>
<b>K Codes for Multivariate Gaussian Distribution</b>	<b>77</b>
<b>L Different optimize methods</b>	<b>78</b>
<b>M Codes for Lecture 10</b>	<b>79</b>
<b>N Weight Decay</b>	<b>80</b>
<b>O Proof of Eq. (75)</b>	<b>81</b>
<b>P Proof of Theorem 4</b>	<b>81</b>
<b>Q Proof of MLE of Gaussian</b>	<b>81</b>

# 1 Lecture 1: Introduction

Link on YouTube: Stanford CS229 Machine Learning, Introduction, 2022, Lecture 1

## History of Machine Learning

机器学习(Machine Learning)这个词语最早是由Arthur Samuel (1959)提出的: *Machine Learning is the field of study that gives the computer the ability to learn without being explicitly programmed*[1].

机器“学习”这个概念最早由Tom Mitchell (1998)定义: *a computer program is said to learn from experience  $E$  with respect to some class of tasks  $T$  and performance measure  $P$ , if its performance at tasks in  $T$ , as measured by  $P$ , improves with experience  $E$* [2].

机器学习根据所解决的任务可以简单地分类为有监督学习(Supervised Learning), 无监督学习(Unsupervised Learning)和强化学习(Reinforcement Learning), 但是他们之间并非完全分开, 现在已经更把他们当作解决问题的工具和方法, 他们之间互有交叉。

## 1.1 Supervised Learning

### Example of house price prediction

以房价预测任务为例, 给定数据集<sup>a</sup>(data set)  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$ , 该集合中的元素 $(x^{(k)}, y^{(k)})$ 被称为样本(samples), 样本由数据-标签对构成, 其中 $x^{(k)}$ 为房子面积,  $y^{(k)}$ 为对应的房价。我们的任务就是根据数据集学习如何根据房子面积预测房价。一种解决方式是使用一次(线性, linear)/二次(quadratic)函数去拟合数据集。

继续考虑更复杂的情况, 例如我们的数据中不仅仅包含房屋面积, 还包括例如生活区面积(living size)等等特征, 此时就希望根据更复杂的数据集 $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}$ 进行预测<sup>b</sup>, 其中 $\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}) \in \mathbb{R}^2$ 。数学形式上, 我们就是希望找到这样的映射:

$$\mathbb{R}^2 \rightarrow \mathbb{R}, \underbrace{(\text{size, lot size})}_{\text{features/input}} \rightarrow \underbrace{\text{price}}_{\text{label/output/supervision}}$$

同样地还可以考虑更高维度的 $\mathbf{x} \in \mathbb{R}^d$ 。后续会讨论数据特征是无限维的情况, 也会讨论如何从数据的多种特征中筛选出“好”的特征。

<sup>a</sup>用Tom Mitchell的话来说数据集就是experience

<sup>b</sup>实际上是更高维度的数据

### Classification of supervised learning

根据标签(label)的类型, 有监督学习可以分为回归(regression)和分类(classification)两类:

	variable	example
Regression	Continuous	price prediction)
Classification	Discrete	predicting types of residence

Table 1: Regression vs Classification

## 1.2 Unsupervised Learning

## Examples

**Clustering:** 仍然以房价预测为例，无监督学习与有监督学习最大的不同就是在无监督学习中数据集没有标签(label)，即 $y^{(k)}$ 。一种做法是将使用算法将相似的数据进行归类，如果新的数据与某一类最“相似”，那么我们就可以将他们归为一类，例如后面的k-means聚类(Clustering)。

**Clustering Genes:** 聚类也可以用于生物信息学，例如基因聚类(Clustering Genes)[3]，将不同的个体的基因进行向量化数值表示后进行使用聚类，就可以将个体分组。

**Latent Semantic Analysis (LSA):** 潜在语义分析，在有很多的文件(documents)组成的语料库中，可以针对每个文件中出现的某些代表性词语(words)对文件进行聚类，从而能够发现主题相似/相同的文件。更多细节可以参见wikipeidia——潜在语义学, Topic detection in a document-word matrix.gif

**Word Embeddings:** 词嵌入是一种将词汇编码为向量的机器学习算法，在自然语言处理中非常重要，代表算法有Word2vec[4], GloVe[5]。在进行词嵌入后，词语编码为向量，词与词之间的关系编码为向量空间中的方向：

$$\text{word} \xrightarrow{\text{encode}} \text{vector}, \text{relation} \xrightarrow{\text{encode}} \text{direction}$$

例如考虑每个国家的首都：在嵌入后的向量空间中，意大利指向其首都罗马的方向与法国指向其首都巴黎的方向基本为一个方向，因此如果想要寻找中国的首都，就在此方向上对城市进行搜索，大概率能够找到北京。更多细节可以参见wikipeida——词嵌入, Word2vec

**Large Language Models, LLM:** 暂略，重要些不言而喻。由于本人数学专业，分享一篇近期Apple研究人员的结果：GSM-Symbolic[6]。他们的结果表明伴随着训练，大模型的数学推理能力并不能被证明得到了提高。

## 1.3 Reinforcement Learning

### Learning to make sequential decisions

无论是在监督学习还是无监督学习中，我们的目的都很清晰，大体上都是在对 $y$ 进行预测，但是在强化学习中，我们仅仅只能从环境中获得反馈(甚至这种反馈不能是及时的反馈)，根据反馈做出决策(这个决策也可能不是立即可以在获得反馈后做出的)以达到一个长期的目的。

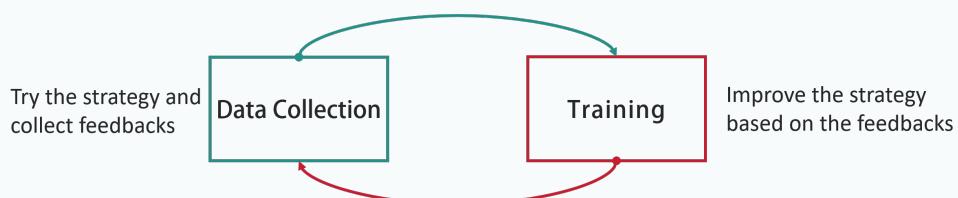


Figure 1: Reinforcement Learning

例如，著名的AlphaGo[7]，在围棋比赛中，最终目的是赢下比赛，这是一个长远的目标，中途需要一步一步的落子以达成目的，并且每一步落子后有一步的反馈，根据这个反馈能够指导下一步的落子。

## 2 Lecture 2: Supervised learning setup

YouTube: Stanford CS229 Machine Learning, Supervised learning setup, 2022, Lecture 2

### 2.1 Definitions

#### Supervised Learning

监督学习的一个重要特征是数据集中含有标签数据。以预测任务(prediction)为例, 我们希望找到一个映射  $h: x \rightarrow y$ , 将特征  $\mathbf{x}$  尽量准确地对应于  $y$ , 例如判断一张图片( $\mathbf{x}$ )中是否是猫( $y$ ), 一段文本( $\mathbf{x}$ )是否含有仇恨语言( $y$ ), 通过一系列特征( $\mathbf{x}$ )预测房价( $y$ )等。

监督学习的数学表达如下:

**given:** 给定训练数据集(training set)

$$D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}, \quad \text{其中 } \mathbf{x}^{(i)} \in X \subseteq \mathbb{R}^m, y^{(i)} \in Y$$

**do:** 找到一个“好”<sup>a</sup>的映射<sup>b</sup>  $h: x \rightarrow y$ .

**Note 1.** 我们想要的并不完全是  $h$  能够在训练集  $D$  上表现好, 我们更希望在  $D$  之外的数据上  $h$  也能够有很好的表现, 即关注所谓的泛化能力(**generalization**)。并且这里有一个隐含的前提假设是  $D$  满足的分布与真实世界的分布相同, 详见附录A。

如果我们的标签  $y$  是离散(discrete)的, 那么就称该监督学习任务是分类任务(**classification**), 例如判断一张图片中的东西是否是猫咪, 输出0表示不是, 输出1表示是。如果  $y$  是连续的(**continuous**), 那么称之为回归任务(**regression**), 例如著名的波士顿房价预测<sup>c</sup>。

<sup>a</sup>如何称为“好”以及怎么构建“好”的映射是一个重要的问题, 将在后续课程讲解

<sup>b</sup>这里使用字母  $h$  的原因是也可以称之为一个假设(**hypothesis**)

<sup>c</sup>Kaggle 数据集, Kaggle 上的实践代码

### 2.2 Linear Regression

#### Linear Regression Model

线性回归是最简单的回归模型, 其是指将  $h$  表示为所有选定特征的线性加和, 即:

$$h(\mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_m x_m = \sum_{i=0}^m \theta_i x_i \quad (1)$$

其中, 数据特征向量  $\mathbf{x} = (x_1, x_2, \dots, x_m)$ , 通常为了方便我们引入一个偏置项  $x_0 = 1$ , 从而可以将模型简洁地表示为  $h(\mathbf{x}) = \sum_{i=0}^m \theta_i x_i$ 。(注意此时  $\mathbf{x}$  已经增加一维)

在模型训练时, 我们所做的就是将  $h$  作用于各数据  $\mathbf{x}^{(i)}$ , 将之对应于相应的标签  $y^{(i)}$ , 如果将所有的数据集统一表达, 就是下面的矩阵形式:

$$h(X) = \theta X \approx Y, \quad X = \begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1^{(1)} & x_1^{(2)} & \dots & x_1^{(m)} \\ x_2^{(1)} & x_2^{(2)} & \dots & x_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x_n^{(1)} & x_n^{(2)} & \dots & x_n^{(m)} \end{bmatrix}, Y^T = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}, \theta^T = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_n \end{bmatrix}$$

一个二维的简单情形如图2所示，其中线性模型为 $h(x) = \theta_0 + \theta_1 x$ ， $\theta_0$ 反映了该直线的截距， $\theta_1$ 反映了斜率。

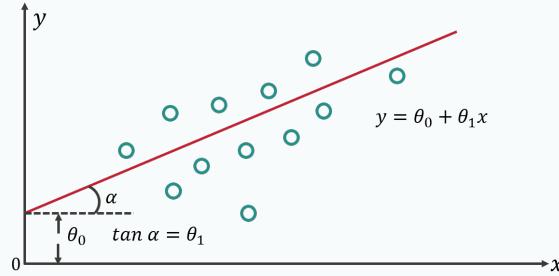


Figure 2: supervised learning in 2 dimansion case

### Loss Function

现在我们已有具体模型的抽象数学表达式(1)，但参数 $\boldsymbol{\theta} = (\theta_0, \dots, \theta_n)$ 现在都是未知的。为获得这些参数可以从优化的视角去看：如果我们有了一种量化一组参数 $\{\theta_i\}_{i=0}^n$ 好坏的方式(或称度量) $L(\boldsymbol{\theta})$ ，并且 $L(\boldsymbol{\theta})$ 越小代表训练集上 $h_{\boldsymbol{\theta}}(\mathbf{x})$ 与 $y$ 差距越小，那么在众多的参数选择中，我们需要找到一组最好(或者一定程度上较好)的参数 $\{\theta_i^*\}_{i=0}^n$ 使得 $L(\boldsymbol{\theta}^*)$ 最小(或较小)。因此我们现在需要一种度量参数好坏的方式，这就是损失函数(**loss function**)

最简单的损失函数是均方误差(**Mean Squared Error, MSE**)：

$$L(\boldsymbol{\theta}) = \frac{1}{2} \sum_{i=1}^n \left( h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \quad (2)$$

其中 $\frac{1}{2}$ 的作用是在对 $L(\boldsymbol{\theta})$ 后与2约去，更简洁。并且从MSE的定义也可以看出， $L(\boldsymbol{\theta})$ 越小代表在总体平均的意义下，在训练集上 $L(\boldsymbol{\theta})$ 能够使得 $h_{\boldsymbol{\theta}}(\mathbf{x})$ 越接近于 $y$ 。

### Gradient Decent

我们知道一个函数的负梯度方向是使其函数值下降的方向，因此我们在寻找 $\boldsymbol{\theta}^* = \arg \min L(\boldsymbol{\theta})$ 时可以首先随机选择一个 $\boldsymbol{\theta}^{(0)}$ ，然后沿着 $L(\boldsymbol{\theta}^{(0)})$ 的负梯度方向“走一步”得到 $\boldsymbol{\theta}^{(1)}$ ，依此类推慢慢向 $\boldsymbol{\theta}^*$ 靠近，示意图如图3所示，其中黄色为 $L(\boldsymbol{\theta}^{(0)})$ 的负梯度方向。

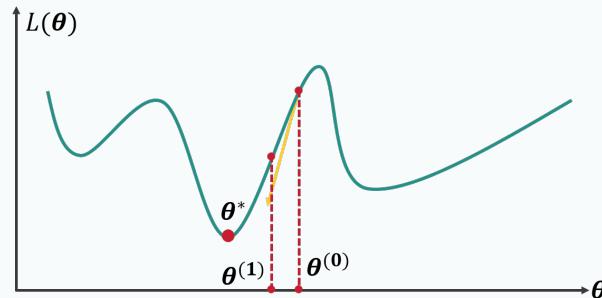


Figure 3: gradient decent

用数学形式写下来就是：

$$\begin{aligned}\theta^{(0)} &= \theta^{(0)} \\ \theta_j^{(t+1)} &= \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} L(\theta_j^{(t)}), \quad j = 1, \dots, m\end{aligned}\tag{3}$$

其中 $\alpha$ 就是上文所说的“走”一步的“步长”。

上面的算法被称为梯度下降法(**Gradient Decent, GD**)，实际上GD会遇到几个问题：

1. 初始值 $\theta^{(0)}$ 选取的不够好很可能无法得到全局极小点，而陷入局部极小，即鞍点(saddle point)，如图4区域II所示，使用GD只能靠近鞍点 $\tilde{\theta}$ 。一种解决方案是引入随机性，使得下一步的方向有可能能够逃出鞍点。
2. 由于步长 $\alpha$ 是固定的，因此有可能出现区域I中“反复横跳”的情形，造成算法的收敛非常困难。一种解决的方案是将步长 $\alpha$ 随着迭代步数 $t$ 的增加不断减小。

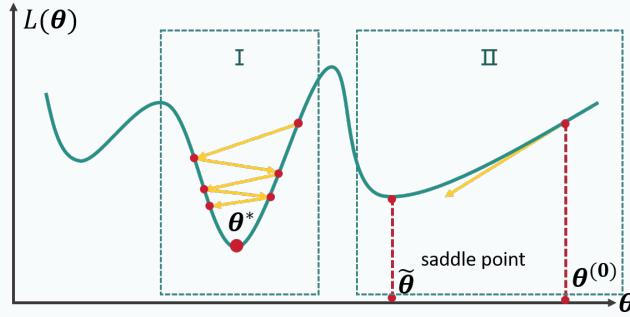


Figure 4: GD problems

对于式(3)，我们完整写出更新过程就是：

$$\begin{aligned}\theta_j^{(t+1)} &= \theta_j^{(t)} - \alpha \frac{\partial}{\partial \theta_j} L(\theta^{(t)}) \\ &= \theta_j^{(t)} - \alpha \sum_{i=1}^n \frac{1}{2} \frac{\partial}{\partial \theta_j} \left( h_{\theta^{(t)}}(\mathbf{x}^{(i)}) - y^{(i)} \right)^2 \\ &= \theta_j^{(t)} - \alpha \sum_{i=1}^n \left( h_{\theta^{(t)}}(\mathbf{x}^{(i)}) - y^{(i)} \right) \frac{\partial}{\partial \theta_j} h_{\theta^{(t)}}(\mathbf{x}^{(i)}) \\ &= \theta_j^{(t)} - \alpha \sum_{i=1}^n \left( h_{\theta^{(t)}}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}, \quad j = 1, \dots, m\end{aligned}\tag{4}$$

其中最后一步利用了 $h(\mathbf{x})$ 的定义式(1)。

### Full Batch v.s. Stochastic Mini Batch

式(4)的一个问题是每一步的更新都需要对训练集 $D$ 上的 $n$ 个数据全部计算一次，这一计算量有时是非常大的，也就是说梯度下降每更新一步都耗时较长。因此下面介绍小批次的梯度下降算法。

**Batch:** 在机器学习中，“batch”指的是在一次迭代中用于训练模型的一组样本。一次训练使用全部训练数据就是使用**Full Batch**，如果将一份训练集随机分为若干等分，每次使用一份进行训练，就是使用**Mini Batch**。其重要优势是可以在GPU上实现并行

化(parallelism)。下面是其数学表示。

我们将训练集 $D$ 均分为 $d$ 份，分别记为 $D_1, D_2, \dots, D_d$ ，其中 $D_i \cap D_j = \emptyset (i \neq j)$ ， $\bigcup_{k=1}^d D_k = D$ 。这样在每一个mini batch  $D_k$ 上，我们都有：

$$\frac{\partial}{\partial \theta_j} L_k(\boldsymbol{\theta}^{(t)}) = \sum_{i \in D_k} \left( h_{\boldsymbol{\theta}^{(t)}}(\mathbf{x}^{(i)}) - y^{(i)} \right) x_j^{(i)}$$

显然对于式(4)中的 $\frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta}^{(t)})$ 有：

$$\frac{\partial}{\partial \theta_j} L(\boldsymbol{\theta}^{(t)}) = \sum_{k=1}^d \frac{\partial}{\partial \theta_j} L_k(\boldsymbol{\theta}^{(t)})$$

由于各个mini batch 中的梯度计算是可以同时进行的(即并行)，因此较full batch 更高效。

---

from October 30, 2024 to October 31, 2024

Updated: October 31, 2024

Last updated: November 15, 2024

### 3 Lecture 3: Weighted Least Squares, Logistic regression, Newton's Method

YouTube: Stanford CS229 Machine Learning, Weighted Least Squares, Logistic regression, Newton's Method, 2022, Lecture 3

#### 3.1 Introduction

##### Recall of Linear Regression

given: 给定训练数据集(training set)<sup>a</sup>

$$D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}, \quad \text{其中 } \mathbf{x}^{(i)} \in \mathbb{R}^{d+1}, y^{(i)} \in \mathbb{R}$$

do: 找到  $\boldsymbol{\theta} \in \mathbb{R}^{d+1}$  使得  $\boldsymbol{\theta} = \arg \min_{\boldsymbol{\theta}} \sum_{i=1}^n (y^{(i)} - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}))^2$ , 其中  $h(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$

<sup>a</sup>特征  $\mathbf{x}^{(i)} \in \mathbb{R}^{d+1}$  是因为包含了常数项1, 这只是一个为了方便的记号

#### 3.2 Probabilistic View of Linear Regression

##### Noise / Error

使用概率的视角去看待线性回归是因为这样我们可以很自然地将其应用于更丰富的模型类别。

现在考虑一个有噪声的线性回归:

$$y^{(i)} = \boldsymbol{\theta}_*^T \mathbf{x}^{(i)} + \epsilon^{(i)} \quad (5)$$

其中对于每一个  $i$  来说,  $\epsilon^{(i)}$  是一个随机噪声(error/noise).<sup>a</sup>

由于噪声  $\epsilon$  是随机的, 那么对于所有的噪声  $\{\epsilon^{(1)}, \epsilon^{(2)}, \dots, \epsilon^{(n)}\}$ , 我们可以关注其统计性质, 一般来说其统计性质的设定有:

1.  $\mathbb{E}[\epsilon^{(i)}] = 0$ , 即无偏性<sup>b</sup>
2.  $\mathbb{E}[\epsilon^{(i)}\epsilon^{(j)}] = \mathbb{E}[\epsilon^{(i)}]\mathbb{E}[\epsilon^{(j)}] \quad \text{for } i \neq j$ , 即误差是相互独立的(一个error不会提供另一个不同的error的任何信息)
3.  $\mathbb{E}[(\epsilon^{(i)})^2] = \sigma^2$ , 即噪声的方差 =  $\mathbb{E}[(\epsilon^{(i)})^2] - \mathbb{E}[(\epsilon^{(i)})]^2 = \sigma^2$ , 这是描述噪声的另一个统计量

虽然真实情况可能并非如此, 但是基本上我们会假设  $\epsilon \sim \mathcal{N}(0, \sigma^2)$ , 关于高斯分布可以见附录B.

**Note 2.** 模型(5)实际上是所谓的前向模型(Forward Model), 就是说即使不知道  $\boldsymbol{\theta}$  是什么, 但是知道标签  $y$  是通过何种方式生成的。或者说这是一种生成模型(Generative Model), 即给了特征  $\mathbf{x}^{(i)}$ , 也给了相应的噪声  $\epsilon^{(i)}$  的构造方式, 那么就可以表示  $y^{(i)}$ .

<sup>a</sup>模型可以解释的部分就是  $\boldsymbol{\theta}$ , 但模型一般不能完全做到  $y^{(i)} = \boldsymbol{\theta}^T \mathbf{x}^{(i)}$ , 这部分差异即不能解释的因素就可以归入噪声中; 在物理中相似地可以理解为每一次进行测量时带来的测量误差(measurement error).

<sup>b</sup>无偏性说明了即使噪声在某些地方对  $y$  的预测产生了影响, 但是总体上而言是并无影响的, 即总体上“没有偏差”; 此外如果噪声期望不为0, 实际上可以对参数  $\boldsymbol{\theta}$  进行简单改变就能得到无偏.

### Probabilistic View of Linear Regression – Why Least Square Method

由式(5)可知 $\epsilon^{(i)} = y^{(i)} - \boldsymbol{\theta}_*^T \mathbf{x}^{(i)}$ ，因此若 $\epsilon \sim \mathcal{N}(0, \sigma^2)$ ，那么 $y - \boldsymbol{\theta}_*^T \mathbf{x} \sim \mathcal{N}(0, \sigma^2)$ ， $y \sim \mathcal{N}(\boldsymbol{\theta}_*^T \mathbf{x}, \sigma^2)$ ，因此有：

$$P(y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}_*) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y^{(i)} - \boldsymbol{\theta}_*^T \mathbf{x}^{(i)})^2}{2\sigma^2}\right) \quad (6)$$

这也就是说，确定模型的参数 $\boldsymbol{\theta}$ 就意味着确定相应分布，即特征 $\mathbf{x}^{(i)}$ 对应的标签 $y^{(i)}$ 满足概率分布(6)，那么实际上模型的Loss就可以理解为在此概率分布下的期望 $\mathbb{E}[y^{(i)} \neq \boldsymbol{\theta}^T \mathbf{x}^{(i)}]$ 。

**Likelihood<sup>a</sup>**: 似然性是指在已知特征 $\mathbf{x}^{(i)}$ 时 $y^{(i)}$ 取值的可能性，我们当然是要选取“最可能”的 $y^{(i)}$ 作为预测标签(最大化对应的似然)。我们需要估计的是模型的似然，即参数 $\boldsymbol{\theta}$ 的似然：

$$\mathcal{L}(\boldsymbol{\theta}) = P(\mathbf{y} | \mathbf{x}; \boldsymbol{\theta}) \stackrel{i.i.d.}{=} \prod_{i=1}^n P(y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}) = \prod_{i=1}^n \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(y^{(i)} - \boldsymbol{\theta}_*^T \mathbf{x}^{(i)})^2}{2\sigma^2}\right) \quad (7)$$

可以看到当前式(7)中的结果是将所有样本的似然性相乘，这是因为我们假设样本是独立同分布的(i.i.d.)。但是为方便进行梯度下降，我们更倾向于使用一种累和而非累加的形式，而取对数就可以不改变单调性地将累乘转化为累加：

$$l(\boldsymbol{\theta}) = \log \mathcal{L}(\boldsymbol{\theta}) = \sum_{i=1}^n \log P(y^{(i)} | \mathbf{x}^{(i)}; \boldsymbol{\theta}) = \sum_{i=1}^n \log \frac{1}{\sigma\sqrt{2\pi}} - \sum_{i=1}^n \frac{(y^{(i)} - \boldsymbol{\theta}_*^T \mathbf{x}^{(i)})^2}{2\sigma^2} \quad (8)$$

我们的目的是使得 $\mathbf{x}^{(i)}$ 对应的标签 $y^{(i)}$ 的概率最大，即最大化似然 $\mathcal{L}(\boldsymbol{\theta})$ ，而对数不改变单调性，因此最终目的就是：

$$\max_{\boldsymbol{\theta}} l(\boldsymbol{\theta}) = \min_{\boldsymbol{\theta}} \sum_{i=1}^n \frac{(y^{(i)} - \boldsymbol{\theta}_*^T \mathbf{x}^{(i)})^2}{2} \triangleq J(\boldsymbol{\theta}) \quad (9)$$

显然，式(9)就是我们熟知的最小二乘法。

对 $J(\boldsymbol{\theta})$ 求导可得：

$$\frac{\partial}{\partial \boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum_{i=1}^n \left( h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) - y^{(i)} \right) \mathbf{x}^{(i)} \quad (10)$$

<sup>a</sup>可见[https://en.wikipedia.org/wiki/Likelihood\\_function](https://en.wikipedia.org/wiki/Likelihood_function)

### 3.3 Classification

#### Introduction

分类问题也是机器学习中的一大类重要问题，且在生活中非常常见。一个最简单的分类问题是二分类：

**given**: 给定训练数据集(training set)

$$D = \{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(n)}, y^{(n)})\}, \quad \text{其中 } \mathbf{x}^{(i)} \in \mathbb{R}^{d+1}, y^{(i)} \in \{-1, 1\}$$

**do**: 找到合适的方式将 $y = 1$ 的正类(positive class)与 $y = -1$ 的负类(negative class)尽可能区分开<sup>a</sup>。

当数据非常简单时，例如只有一维、数据量不大，并且类别相同的数据很好地聚在一

起时，我们可以使用简单的线性回归就将数据很好的分开，如图5左图所示。但是当同类数据不是很集中时，就很难使用简单的线性回归进行分类，如图5右图所示，直线会逐渐变平导致系数逐渐趋于零使的分类器几乎与特征无关（不能提取特征）。

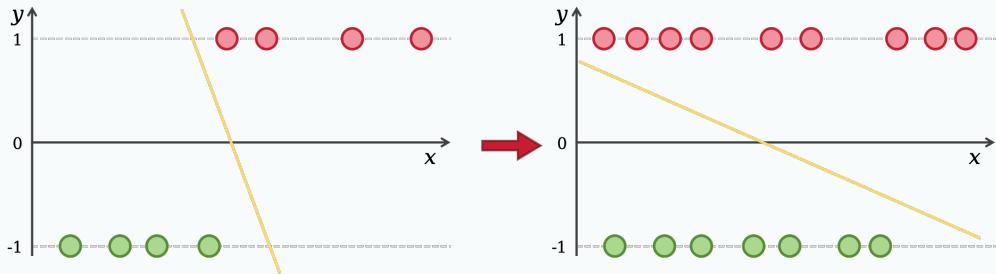


Figure 5: example of classification

<sup>a</sup>正类与负类只是名称，并无特定含义，同时取任何能代表类别不同的指示如{0,1}都可以

### Logistic Regression-Introduction

可以看到我们使用直线或超平面并不足以对数据点进行很好的分类，如果我们的分类器是一个更复杂的曲线(如图6使用所谓的logistic回归对{0, 1}标签进行回归)，那么会对提升分类成功率也许会有帮助。

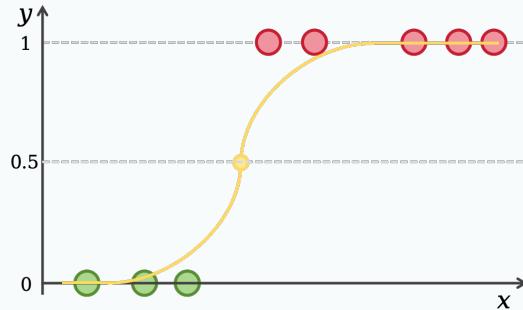


Figure 6: logistic regression

让我们的线性模型变成曲线/曲面的自然做法是添加非线性，即对模型 $h(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}$ 中的 $\boldsymbol{\theta}^T \mathbf{x}$ 添加非线性作用 $g(\cdot)$ ，这种非线性作用被称为链接函数(link function)<sup>a</sup>。一个常用的link function是sigmoid函数：

$$g(x) = \frac{1}{1 + e^{-x}} \quad (11)$$

显然有 $0 < g(x) < 1$ ，因此自然可以作为对{0, 1}标签的分类器，并且由于此函数非常“光滑”，因此计算导数时性质较好，也便于进行梯度下降。

这样就得到了logistic regression模型：

$$h(\mathbf{x}) = g(\boldsymbol{\theta}^T \mathbf{x}) = (1 + \exp(-\boldsymbol{\theta}^T \mathbf{x}))^{-1} \quad (12)$$

<sup>a</sup>link function的作用就是增加非线性，在深度学习中，实际上就是激活函数(activation function)的作用

### Logistic Regression–Loss Function

与式(7)中一样，我们同样以概率和统计的视角考虑logistic regression的极大似然。首先令：

$$P(y=1|x; \theta) = h_\theta(x), \quad P(y=0|x; \theta) = 1 - h_\theta(x) \quad (13)$$

此时将式(14)合二为一即为：

$$P(y|x; \theta) = h_\theta(x)^y \times (1 - h_\theta(x))^{1-y}, \quad y \in \{0, 1\} \quad (14)$$

因此可以将似然函数定义如下：

$$\mathcal{L}(\theta) = P(\mathbf{y}|\mathbf{x}; \theta) \stackrel{i.i.d.}{=} \prod_{i=1}^n P(y^{(i)}|\mathbf{x}^{(i)}; \theta) = \prod_{i=1}^n h_\theta(x^{(i)})^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \quad (15)$$

分析式(15)的定义形式可以看出，如果标签  $y^{(i)} = 1$ ，那么只剩下  $h_\theta(x^{(i)})^{y^{(i)}}$ ，此时我们期望的当然是  $h_\theta(x^{(i)})^{y^{(i)}}$  更加接近1，即使得  $\mathcal{L}(\theta)$  更大；如果标签  $y^{(i)} = 0$ ，那么只剩下  $1 - h_\theta(x^{(i)})^{y^{(i)}}$ ，此时我们期望的是  $h_\theta(x^{(i)})^{y^{(i)}}$  更加接近0，同样使得  $\mathcal{L}(\theta)$  更大。

与式(8)类似，对式(15)取对数，得到：

$$l(\theta) = \log \mathcal{L}(\theta) = \sum_{i=1}^n \log P(y^{(i)}|\mathbf{x}^{(i)}; \theta) = \sum_{i=1}^n \left( y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right) \quad (16)$$

因此我们的损失函数定义为：

$$J(\theta) \triangleq - \sum_{i=1}^n \left( y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log (1 - h_\theta(x^{(i)})) \right) \quad (17)$$

此时再进行梯度下降：

$$\theta^{i+1} = \theta^i - \alpha \frac{\partial}{\partial \theta} J(\theta) \quad (18)$$

其中

$$\frac{\partial}{\partial \theta} J(\theta) = \sum_{i=1}^n \left( h_\theta(x^{(i)}) - y^{(i)} \right) x^{(i)} \quad (19)$$

巧合的是，这里损失函数的导数竟然与线性回归中的式(10)一模一样，事实上这也四一个普遍的规律，将会在后续课程中介绍。此处式(19)的详细推导详见附录C。

### 3.4 Newton's Methods

#### Newton's Methods

给定函数  $f: \mathbb{R}^d \rightarrow \mathbb{R}$ ，我们想要找到  $f$  的零点，即  $f(\theta) = 0$ ，这实际上对应于我们在寻找最值过程中的目标  $l'(\theta) = 0$ 。我们的迭代过程如图7所示：

1. 初始点为猜测点（随机/一定方式框定的）记为  $\theta^{(0)}$
2. 沿该点切线方向寻找其与  $x$  轴交点，记为  $\theta^{(1)}$
3. 依照2的模式迭代直至满足停机准则

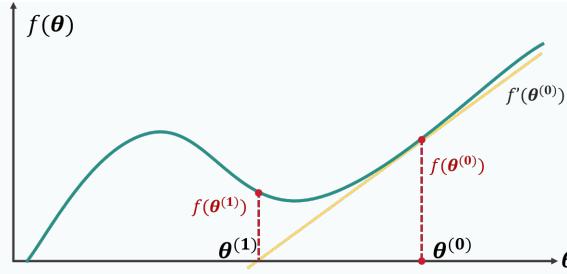


Figure 7: Newton method

这种迭代格式的数学形式就是：

$$\theta^{(t+1)} = \theta^{(t)} - \Delta$$

其中根据三角形 $\triangle_{f(\theta^{(t)})\theta^{(t)}\theta^{(t+1)}}$ 的结构就可以知道

$$\Delta = f(\theta^{(t)})/f'(\theta^{(t)})$$

自然地，当函数值是高维情形 $\theta \in \mathbb{R}^{d+1}$ 时：

$$\theta^{(t+1)} = \theta^{(t)} - H^{-1} \cdot \nabla_{\theta} l(\theta)$$

其中 $H \in \mathbb{R}^{(d+1) \times (d+1)}$ 是Hessian矩阵， $H_{ij} = \frac{\partial^2}{\partial \theta_i \partial \theta_j} l(\theta)$ 。

最后值得一提的是，牛顿法在收敛速度方面一般来说是非常快的，但是可以看到计算Hessian矩阵所需要的计算耗时和存储都非常巨大。

### Comparison of different optimization algorithms

Comparison of different optimization algorithms			
Methods	Per iteration	Compute <sup>1</sup>	Steps to error $\epsilon$
SGD	1 data point	$\Theta(d)$	$\epsilon^{-2}$
Batch GD	$N$ data points	$\Theta(nd)$	$\approx \epsilon^{-1}$
Newton Method	$N$ data points	$\Omega(nd^2)$	$\approx \log(\epsilon^{-1})$

<sup>1</sup>  $d$ 为数据维度， $n$ 为训练数据量（并不等价于数据集大小，因为可能没训练完）

<sup>2</sup>  $\Theta(x)$ 表示渐近紧界，即在 $x$ 的常数范围 $(C_1x, C_2x)$ 内变动； $\Omega(x)$ 表示渐近下界，表示至少是 $x$

Table 2: Representative Algorithms by Category

from November 3, 2024 to November 22, 2024

First updated: November 22, 2024

Last updated: May 31, 2025, modified equation(12), thanks to Yifu Zheng!

## 4 Lecture 4: Exponential family, Generalized Linear Models

YouTube: Stanford CS229 Machine Learning, Exponential family, Generalized Linear Models, 2022, Lecture 4

### 4.1 Exponential Family

#### Introduction

Exponential Family这一概念在历史上曾经很重要，虽然这并不是目前最先进的模型，但其中的思想依然值得学习。而其思想是将概率密度函数(probability density function, p.d.f.)写成一个统一的形式，从而将多种常见分布的密度函数直接作为其特殊情形下的推论。

Exponential Family模型是将概率密度函数写为如下数学形式：

$$\mathbb{P}(y; \eta) = b(y) \exp [\eta^T T(y) - a(\eta)] \quad (20)$$

其中 $y$ 为数据(data),  $\eta$ 为自然参数(natural parameters),

1.  $b(y)$ 称为基本度量(base measure), 其与 $\eta$ 无关, 为标量(scalar)
2.  $T(y)$ 称为充分统计量(sufficient statistics), 其在捕捉与数据 $y$ 有关的信息, 事实上后续会设定 $T(y) = y$ ,  $T(y)$ 是与 $\eta$ 的维数相同的向量
3.  $a(\eta)$ 被称为对数分配函数(log partition function), 其与数据 $y$ 无关<sup>a</sup>, 为标量

<sup>a</sup>尽管我们对 $P(y; \eta)$ 要求作用于 $y$ 后概率和为1, 但是实际上与 $y$ 无关的 $a(\eta)$ 却是包含了整个 $P$ 信息的核心, 可见Theorem 1.

#### Example 1 – Bernoulli distribution

对于一个Bernoulli distribution, 即0-1分布

$$\mathbb{P}(y; \phi) = \phi^y (1 - \phi)^{1-y}, \quad y \in \{0, 1\}$$

其可以写为(推导详见E):

$$\mathbb{P}(y; \phi) = \exp [\eta y - \log(e^{\eta+1})] \quad (21)$$

其中 $\eta = \log\left(\frac{\phi}{1-\phi}\right)$ ,  $T(y) = y$ . 并且神奇的是计算可得

$$a'(\eta) = \frac{e^\eta}{1 + e^\eta} = \frac{\phi}{1 - \phi} \times (1 - \phi) = \phi = \mathbb{E}[T(y)]$$
$$a''(\eta) = \left(\frac{e^\eta}{1 + e^\eta}\right)' = \frac{e^\eta}{((1 + e^\eta)^2)} = \frac{\phi}{1 - \phi} \times (1 - \phi)^2 = \phi(1 - \phi) = \text{Var}(T(y))$$

### Example 2 – Gaussian distribution

对于一个高斯分布  $y \sim \mathcal{N}(\mu, \sigma^2)$ , 其中均值  $\mu$  需要预测, 方差  $\sigma^2$  固定

$$\mathbb{P}(y; \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y - \mu)^2}{2\sigma^2}\right]$$

其可以写为(推导详见F):

$$\mathbb{P}(y; \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{y^2}{2}\right) \exp\left[\eta \cdot \left(\frac{1}{\sigma^2}y\right) - \frac{1}{2\sigma^2}\eta^2\right] \quad (22)$$

其中  $\eta = \mu$ ,  $T(y) = \frac{1}{\sigma^2}y$ .

同样地计算可得

$$a'(\eta) = \frac{1}{\sigma^2}\mu = \mathbb{E}[T(y)], \quad a''(\eta) = \frac{1}{\sigma^2} = \text{Var}(T(y))$$

### Why exponential family?

在Lecture 3中我们已经将回归、分类等问题使用概率的视角进行转化, 实际上我们最后做的工作就是对某一个假设的概率的决定性参数进行估计, 例如高斯分布我们需要估计均值(也可以同时估计方差)、0-1分布我们需要估计事件成功的概率, 而将这些概率模型写成式(20)的形式后, 我们事实上有:

**Theorem 1.** 将  $\mathbb{P}(y; \eta)$  写成 *exponential family* 形式  $\mathbb{P}(y; \eta) = b(y) \exp[\eta^T T(y) - a(\eta)]$  后, 有<sup>a</sup>

$$a'(\eta) = \mathbb{E}[T(y)], \quad a''(\eta) = \text{Var}(T(y))$$

将式(20)对  $\eta$  求二阶导发现  $\mathbb{P}''(y; \eta) = -a'' = -\text{Var}(y) \leq 0$ , 因此使用此式定义损失函数后得到的是一个完全凹的优化问题(添上负号后当然等价于凸优化), 求解是非常高效的。

<sup>a</sup> 证明见附录G, 且计算后发现结论与课程中的不匹配, 课程中写的是  $\mathbb{E}[y]$  和  $\text{Var}(y)$

## 4.2 Generalized Linear Models

### Assumptions of Generalized Linear Models

广义线性模型是将自然参数表示为特征的线性组合, 即  $\eta = \theta^T x$ , 再将自然参数代入 Exponential Family 计算概率分布, 因此需要作出如下假设:

假设一: 模型分布满足 Exponential Family, 即  $y|x; \theta \sim \text{Exponential Family}$ .

不同的任务会对应 Exponential Family 中不同的分布, 如 Table 3 所示:

Task-Distribution Correspondence	
Task	Distribution
Binary(分类)	Bernoulli Distribution
Real Value(回归)	Gaussian Distribution
Counts(计数)	Poisson Distribution
$R_x$ (正实数线)	Gamma Distribution

Table 3: Task-distribution correspondence

**Note 3.** 我们已经看到指数族模型已经包括了很多种分布，因此这个假设比一般的如假设数据满足高斯分布等更具有普遍性，但是由于指数族模型也并非囊括了所有的分布，所以也会引入模型误差。

假设二: Exponential Family 的自然参数是特征的线性组合, 即  $\eta = \theta^T x, \theta \in \mathbb{R}^{d+1}, x \in \mathbb{R}^{d+1}$ .

**Note 4.** 在低维情况下可能数据点并非是线性可分的, 即可能并不存在超平面将两组数据点分开, 但是在高维情形下, 这一般是可以做到的。

假设三: 在测试即推理(inference)阶段, 输出为  $\mathbb{E}[y|x; \theta]$ , 即

$$\eta_\theta(x) = \mathbb{E}[y|x; \theta] \quad (23)$$

### Generalized Linear Model

Generalized Linear Model 首先将 Exponential Family 的自然参数  $\eta$  表示为特征的线性组合:

$$\eta = \theta^T x \quad (24)$$

随后根据式(20)输出  $y$  的概率分布。在学习(Learning)阶段, 以最大化正确标签的概率<sup>a</sup>

$$\max_{\theta} \log \mathbb{P}(y | x; \theta) \quad (25)$$

其中参数优化的过程与前面几个Lecture中得到的一致, 为:

$$\theta_j := \theta_j + \alpha \sum_{i=1}^N (y^{(i)} - h_\theta x^{(i)}) x^{(i)} \quad (26)$$

因此Generalized Linear Model 的示意图如图8所示

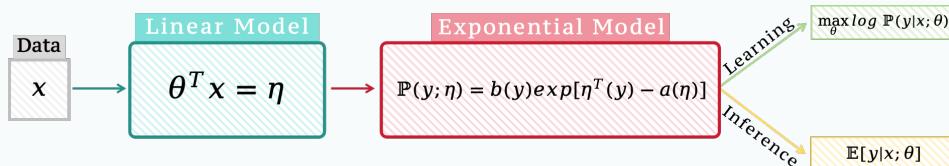


Figure 8: Generalized Linear Model

我们在 Lecture 3 中介绍概率模型时是先对分布进行假设后再对其中的参数进行估计和优化, 例如二分类中使用 Bernoulli 分布, 需要估计参数  $\phi$ , 在连续值回归预测中使用高斯分布, 固定方差  $\sigma^2$  后估计均值  $\mu$ 。我们称之前所假设的这些分布中的参数为 canonical

parameters, 与 Exponential Family 中的 natural parameter  $\eta$  相对应。事实上我们可以通过映射  $g$  将  $\eta$  映射为 canonical parameters, 同理也有逆映射  $g^{-1}$  将 canonical parameters 映射为  $\eta$ , 并将映射  $g$  称为 canonical function / canonical response / link function.

在 Logistic Regression 中(对应于 **Example 1 – Bernoulli distribution**),

$$h_\theta(x) = \mathbb{E}[y|x; \theta], \quad \phi = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-\theta^T x}}$$

在一般连续值回归中(对应于 **Example 2 – Gaussian distribution**),

$$h_\theta(x) = \mathbb{E}[y|x; \theta] = \mu = \theta^T x$$

<sup>a</sup>注意由于  $\eta$  由  $\theta$  表示, 因此  $\mathbb{P}(y | x; \eta)$  就是  $\mathbb{P}(y | x; \theta)$

### 4.3 Multi Classification via Softmax

#### Multi Classification-Encoding

在 Lecture 3 中我们考虑了二分类任务, 但是实际中多分类任务也是常见的。例如考虑一个四分类任务: 区分 {cat, dog, car, bus} 这四类事物的图片。

为了将他们区分开, 我们首先要为不同的类别配备可以完全区分的数值化表示——**one hot vector** —— 对于  $K$  个类别, 编码每个类别为一个  $K$  维向量

$$\mathbf{y} \in \{0, 1\}^K, \quad s.t. \sum_{i=1}^K y_i = 1$$

因此这里我们可以将四种类别编码为:

$$\text{cat} : [1 \ 0 \ 0 \ 0]^T; \quad \text{dog} : [0 \ 1 \ 0 \ 0]^T; \quad \text{car} : [0 \ 0 \ 1 \ 0]^T; \quad \text{bus} : [0 \ 0 \ 0 \ 1]^T$$

在进行了向量化编码之后, 我们就可以在向量空间中考虑分类问题, 如图(9)所示, 对于类别  $i$ , 我们想要找到的是一个超平面  $\theta_i x = 0$  使得该类别与其他类别尽量分开<sup>a</sup>:

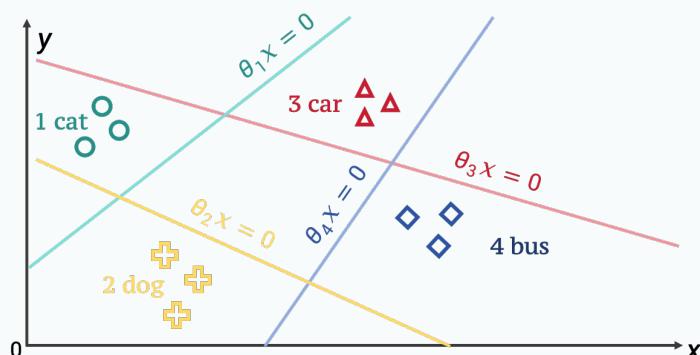


Figure 9: Multi Classification

<sup>a</sup>此示意图是二维空间, 但实际是四维; 在真实情况下数据的分布很可能也没有图中这么聚集; 我们目的是使得只有第  $i$  类的点满足  $\theta_i x > 0$

## Multi Classification–Probability

正如one-hot vector 所示真实情况下四个维度只能由一个类别的概率为1(如图10)，这个概率为1对应的类别当然就是其真实类别，但是这样的编码是一个离散型的，在我们在模型中常常不能实现只有一个类别概率为1其余为0，而是每个类别都有可能，但是有概率大小区别(如图11)。

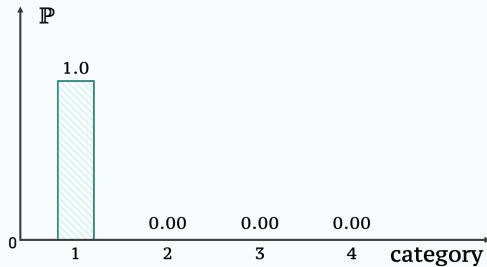


Figure 10: one-hot vector

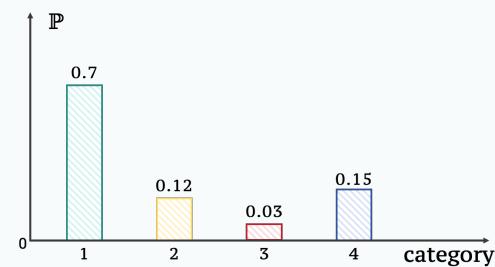


Figure 11: model output

在训练模型时我们实际得到的是参数 $\{\theta_1, \dots, \theta_K\}$ ，因此对于数据 $x$ ，可以得到一系列的值 $\{\theta_1 x, \dots, \theta_K x\}$ ，为了将其转化为概率，我们只需要做一步softmax:

$$\mathbb{P}(y = k|x; \theta) = \frac{\exp \theta_k x}{\sum_{i=1}^K \exp \theta_i x} \quad (27)$$

因此我们需要最大化正确类别的预测概率，即等价于:

$$\min - \sum_{i=1}^K \mathbb{P}(y = i) \log \hat{\mathbb{P}}(y) = \min - \log(\hat{\mathbb{P}}(y_k)) = - \min \log \frac{\exp \theta_k x}{\sum_{i=1}^K \exp \theta_i x} \quad (28)$$

**Note 5.** 在式(27)中我们只使用了 $x$ 作为特征进行计算，但是有时候仅使用他们是不够的。神经网络效果表现良好很大程度上的原因是其能够作为一个高效特征提取器，例如可以提取出 $x^2, \sqrt{x}$ 这样的非线性特征。

from November 30, 2024 to December 8, 2024

First updated: December 8, 2024

Last updated: August 11, 2025, change the eq.(22)

## 5 Lecture 5: Gaussian discriminant analysis, Naive Bayes

YouTube: Stanford CS229 Machine Learning, Gaussian discriminant analysis, Naive Bayes, 2022, Lecture 5

### 5.1 Introduction

#### Generative learning algorithms – Introduction

生成学习算法/生成模型(Generative learning algorithms / Generative model)是一种建模可观察变量  $\mathbf{X}$  和目标变量  $Y$  的联合概率分布  $\mathbb{P}(\mathbf{X}, Y)$  的统计模型，在预测时生成模型可用于“生成”新的观察  $\mathbf{x}$  的随机实例<sup>a</sup><sup>b</sup>.

生成学习算法主要包括两方面:

1. Gaussian Discriminative analysis (GDA) – 高斯判别分析
2. Naive Bayes – 朴素贝叶斯

在前面的Lecture 中所讲的模型都是Discriminative learning algorithm, 因为我们一直在对模型参数化, 然后试图寻找最优的参数, 例如在Exponential family 模型中

$$y|\mathbf{x}; \theta \sim \text{Exponential family}(\eta), \quad \eta = \theta^T \mathbf{x}$$

其中  $\theta$  是参数.

<sup>a</sup>更多介绍可见Wikipedia的Generative model

<sup>b</sup>注意这里的生成模型仅仅是一个统计机器学习算法, 而非目前很火的生成式人工智能等等

#### Generative learning algorithms

**Model:** 在Generative learning algorithms 中, 我们希望建模/参数化(Model / Parameterize) 特征  $\mathbf{x}$  与标签  $y$  的联合概率分布  $\mathbb{P}(\mathbf{x}, y)$ , 根据条件概率公式, 我们有:

$$\mathbb{P}(\mathbf{x}, y) = \mathbb{P}(\mathbf{x}|y)\mathbb{P}(y) \tag{29}$$

其中  $y$  是所有的标签, 一般考虑离散情形, 因此如果标签有  $N$  个类别, 那么就需要建模  $N$  个式(29).

**Learning Time:** 在训练时, 我们需要学习分布  $\mathbb{P}(\mathbf{x}|y)$  和  $\mathbb{P}(y)$ , 其中  $\mathbb{P}(y)$  是label 的先验分布(prior), 一般通过假设/观察确定一种分布, 再用参数描述, 最后通过学习得到.

**Testing Time:** 在测试时, 我们仍是预测给定特征  $\mathbf{x}$  的相应标签  $y$ , 本质上是计算条件概率  $\mathbb{P}(y|\mathbf{x})$ . 根据贝叶斯公式(Bayes Rule)和全概率公式(Law of total probability), 有:

$$\mathbb{P}(y|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}, y)}{\mathbb{P}(\mathbf{x})} = \frac{\mathbb{P}(\mathbf{x}|y)\mathbb{P}(y)}{\mathbb{P}(\mathbf{x})} = \frac{\mathbb{P}(\mathbf{x}|y)\mathbb{P}(y)}{\sum_{y'} \mathbb{P}(\mathbf{x}|y')\mathbb{P}(y')} \tag{30}$$

根据式(29), 训练阶段已经学习了各个标签的分布  $\mathbb{P}(\mathbf{x}|y)\mathbb{P}(y)$ , 因此只需要根据式(30)计算得到每个标签的  $\mathbb{P}(y|\mathbf{x})$ , 然后选择概率最大的标签作为预测即可.

## 5.2 Gaussian Discriminative analysis(GDA)

### Gaussian Discriminative analysis(GDA) – Introduction

在高斯判别分析中，我们考虑高维情形，即  $\mathbf{x} \in \mathbb{R}^d$ ，并且为方便我们规定第一个坐标  $x_0 = 1$ .

\***Assumption<sup>1</sup>**: 假设对于各个标签  $y$ ,  $\mathbb{P}(\mathbf{x}|y)$  满足高维高斯分布，即  $\mathbb{P}(\mathbf{x}|y) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ .

**Question:** 为什么不将所有的特征  $\mathbf{x}$  统一建模成一个高斯分布，而是对于每一个标签分别建立一个高斯分布呢？

**Answer:** 在实际问题中，不同类别的样本通常具有不同的分布特征。将所有的特征  $\mathbf{x}$  统一建模为一个高斯分布往往过于简单，无法有效捕捉不同类别之间的差异。

这个假设是高斯判别分析的重要前提，关于高维高斯分布可见附录H

## 5.3 Gaussian Discriminative analysis(GDA) – 2-Classification Case

### GDA – 2-Classification Case – Problem and Model

**Problem:** 考虑一个二分类问题(如图12)，其中训练集为  $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$ ，标签  $y \in \{0, 1\}$ .

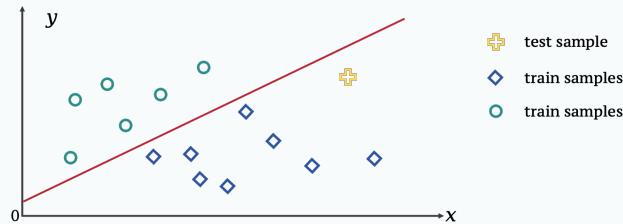


Figure 12: Classification

**Model:** 首先需要假设两个标签所对应的特征的分布都满足高维高斯分布，即<sup>a</sup>

$$\begin{aligned} \mathbf{x}|(y=0) &\sim \mathcal{N}(\boldsymbol{\mu}_0, \Sigma), \quad \boldsymbol{\mu}_0 \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d \times d} \\ \mathbf{x}|(y=1) &\sim \mathcal{N}(\boldsymbol{\mu}_1, \Sigma), \quad \boldsymbol{\mu}_1 \in \mathbb{R}^d, \Sigma \in \mathbb{R}^{d \times d} \end{aligned}$$

由于是二分类问题，因此标签  $y$  的先验概率设为参数为  $\phi$  的 Bernoulli 分布，即

$$y \sim \text{Bernoulli}(\phi), \quad \mathbb{P}(y=1) = \phi, \quad \mathbb{P}(y=0) = 1 - \phi$$

因此在此模型中参数有四个，为；  $\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi$ .

<sup>a</sup>为推导方便设定两个协方差矩阵相同，但当他们不同时也是完全可以推导的，只是复杂一些

## GDA – 2-Classification Case – Learn / Fit parameters

### Learn / Fit parameters

**Before:** 为学习模型参数, 之前的原则是使得训练集中所有特征-标签对 $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^n$ 在这些参数下的联合概率最大, 即最大化似然(maximum likelihood estimation, MLE):

$$\begin{aligned}
 L(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) &= \mathbb{P}((\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \dots, (\mathbf{x}^{(n)}, y^{(n)}); \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) \\
 &\stackrel{i.i.d.}{=} \prod_{i=1}^n \mathbb{P}((\mathbf{x}^{(i)}, y^{(i)}); \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) \\
 &= \prod_{i=1}^n \mathbb{P}(\mathbf{x}^{(i)}|y^{(i)}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) \cdot \mathbb{P}(y^{(i)}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) \\
 &= \prod_{i=1}^n \mathbb{P}(\mathbf{x}^{(i)}|y^{(i)}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma) \cdot \mathbb{P}(y^{(i)}; \phi)
 \end{aligned} \tag{31}$$

其中最后一步化简是因为第一项条件概率与 $\phi$ 无关, 而后一项标签的概率只与 $\phi$ 有关.

**GDA:** 在GDA 中, 与以往不同的是我们需要最大化条件概率的似然:

$$\begin{aligned}
 L(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) &= \mathbb{P}(y^{(1)}, y^{(2)}, \dots, y^{(n)}|\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(n)}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) \\
 &\stackrel{i.i.d.}{=} \prod_{i=1}^n \mathbb{P}(y^{(i)}|\mathbf{x}^{(i)}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi)
 \end{aligned} \tag{32}$$

可以看出, 在GDA 中我们更关心在观测到 $\mathbf{x}$ 后 $y$  的概率, 而并不对 $\mathbf{x}$  进行单独建模。

## GDA – 2-Classification Case – Solutions

**Optimize:** 与前面Lecture 中一样, 在式(31)中最大化似然函数等价于最大化对数似然:

$$\begin{aligned}
 \arg \max L(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) &= \arg \max \log(L(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi)) \triangleq \arg \max l(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) \\
 &= \arg \max \sum_{i=1}^n \left[ \log \mathbb{P}(\mathbf{x}^{(i)}|y^{(i)}) + \log \mathbb{P}(y^{(i)}) \right]
 \end{aligned} \tag{33}$$

此时只需令 $\nabla l(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) = 0$ , 即<sup>a</sup>

$$\frac{\partial l}{\partial \phi} = 0, \frac{\partial l}{\partial \boldsymbol{\mu}_0} = 0, \frac{\partial l}{\partial \boldsymbol{\mu}_1} = 0, \frac{\partial l}{\partial \Sigma} = 0 \tag{34}$$

**Solutions:** 首先为将不同标签对应的特征区分开, 先定义两个指标集合:

$$U_0 = \{i : y^{(i)} = 0\}, \quad U_1 = \{i : y^{(i)} = 1\}$$

那么根据式(34)最终可以解出(证明详见附录I):

$$\phi = \frac{|U_1|}{n} = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(y^{(i)} = 1) \tag{35}$$

$$\boldsymbol{\mu}_0 = \frac{1}{|U_0|} \sum_{i \in U_0} \mathbf{x}^{(i)} = \frac{1}{\sum_{i=1}^n \mathbb{I}(y^{(i)} = 0)} \left( \sum_{i=1}^n \mathbf{x}^{(i)} \cdot \mathbb{I}(y^{(i)} = 0) \right) \quad (36)$$

$$\boldsymbol{\mu}_1 = \frac{1}{|U_1|} \sum_{i \in U_1} \mathbf{x}^{(i)} = \frac{1}{\sum_{i=1}^n \mathbb{I}(y^{(i)} = 1)} \left( \sum_{i=1}^n \mathbf{x}^{(i)} \cdot \mathbb{I}(y^{(i)} = 1) \right)$$

$$\begin{aligned} \Sigma &= \frac{1}{n} \sum_{i=1}^n \left( \mathbf{x}^{(i)} - \boldsymbol{\mu}_{y^{(i)}} \right) \left( \mathbf{x}^{(i)} - \boldsymbol{\mu}_{y^{(i)}} \right)^T \\ &= \frac{1}{n} \left[ \sum_{i \in U_0} \left( \mathbf{x}^{(i)} \right) \left( \mathbf{x}^{(i)} \right)^T + \sum_{i \in U_1} \left( \mathbf{x}^{(i)} - \boldsymbol{\mu}_1 \right) \left( \mathbf{x}^{(i)} - \boldsymbol{\mu}_1 \right)^T \right] \end{aligned} \quad (37)$$

<sup>a</sup>这四个方程的维数都是与相应参数相对应

## GDA – 2-Classification Case – Prediction

**Prediction:** 给定一个  $\mathbf{x}$ , 需要输出  $y \in \{0, 1\}$ , 而此时我们的输出为  $\arg \max \mathbb{P}(y|\mathbf{x})$ , 实际上只有如下两种可能:

$$\arg \max \{ \mathbb{P}(y = 0|\mathbf{x}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi), \mathbb{P}(y = 1|\mathbf{x}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) \}$$

根据贝叶斯公式可以得到(证明见附录J, 可作为练习)

$$\begin{aligned} \mathbb{P}(y = 1|\mathbf{x}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) &= \frac{\mathbb{P}(\mathbf{x}|y = 1; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma) \cdot \mathbb{P}(y = 1; \phi)}{\mathbb{P}(\mathbf{x}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi)} \\ &= \frac{1}{1 + \exp[-(\theta^T \mathbf{x} + \theta_0)]}, \quad \theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}, \text{均与参数 } \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi \text{ 相关} \end{aligned} \quad (38)$$

**Decision Boundary:** 不妨记  $a = \mathbb{P}(y = 0|\mathbf{x}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi)$ ,  $b = \mathbb{P}(y = 1|\mathbf{x}; \boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi)$ , 显然有  $a + b = 1$ , 那么

$$\max\{a, b\} = \begin{cases} a, & \text{if } a \geq 0.5 > b \\ b, & \text{if } b \geq 0.5 > a \end{cases}$$

当  $a = b$  时, 我们称此时的特征集合  $\{\mathbf{x} : \mathbb{P}(y = 0|\mathbf{x}) = 0.5\}$  为决策边界(Decision Boundary).

由式(38)可得:

$$\begin{aligned} \text{Decision boundary: } \frac{1}{1 + \exp[-(\theta^T \mathbf{x} + \theta_0)]} &= 0.5 \\ \Leftrightarrow \exp[-(\theta^T \mathbf{x} + \theta_0)] &= 1 \Leftrightarrow \theta^T \mathbf{x} + \theta_0 = 0 \end{aligned} \quad (39)$$

因此如果判定  $\mathbf{x}$  的类别是  $y = 1$ , 那么就等价于:

$$\mathbb{P}(y = 1|\mathbf{x}) > 0.5 \Leftrightarrow \theta^T \mathbf{x} + \theta_0 > 0 \quad (40)$$

事实上, 当数据不满足高斯性时, 也有可能最后得到相同形式的 Decision Boundary (38). 例如  $x \in \mathbb{N}, x|(y = i) \sim \text{Poisson}(\lambda_i)$ ,  $\mathbb{P}(x = k) = e^{-\lambda_i} \frac{\lambda_i^k}{k!}, i \in \{0, 1\}, \mathbb{P}(y = 1) = \phi$ , 此时仍然可以得到式(38).

## 5.4 Summary

### Questions and Answers

**Q1:** 可以看到在二分类 GDA 中只需要求解 Decision Boundary =  $\{x : \theta^T x + \theta_0 = 0\}$ , 那么对于多分类也有 Decision Boundary 吗?

**A1:** 有, 但是会更加复杂, 需要仔细判定和设计.

**Q2:** 在逻辑回归中我们的形式和式(38)是一样的 (均为线性判别器), 那么这两个模型有什么区别呢?

**A2:**

	GDA(Generative)	Logistic(Discriminative)
<b>Assumption</b>	$x y = k \sim \mathcal{N}(\mu_k, \Sigma), k \in \{0, 1\}$ $y \sim \text{Bernoulli}$ (假设分布)	$\mathbb{P}(y=1 x) = \frac{1}{1+\exp^{-\theta^T x}}$ (直接假设模型)
<b>Modeling</b>	对 $\mathbb{P}(x, y)$ 建模, 由条件概率公式有 $\mathbb{P}(x, y) = \mathbb{P}(x y)\mathbb{P}(y)$	仅对 $\mathbb{P}(y x)$ 建模
<b>Process</b>	模型先学参数 $\mu_0, \mu_1, \Sigma, \phi$ , 再计算 得到 $\theta, \theta_0$	模型直接学习 $\theta$

Table 4: Comparison between GDA and Logistic regression

### High level perspective

可以看到相较于 Logistic regression, GDA 有更多的假设 (高斯性), 更多的正确的假设会带来更好的模型表现, 因为引入了正确的先验知识. 然而, 引入假设都伴随着假设错误的风险, 因此也有风险使模型表现很糟糕.

**Good:** More assumption + Correct Assumption  $\Rightarrow$  Better performance

**Risk:** You might make wrong assumption!  $\Rightarrow$  Worse performance

不同的生成式学习算法(Generative learning algorithm)互相间性能的差异很大程度上是由其假设与问题的贴合性导致的; 生成式学习算法与判别式学习算法(Discriminative learning algorithm)性能的差异往往是由生成式学习算法做出了好的/不好的假设导致.

在解决问题时, 模型获取知识的来源有两个: 1. Assumption, 2. Data. 当数据足够多时, 有时做先验假设引入的风险反而使做假设引入知识变得不值得, 因此在现代的深度学习/大规模机器学习中, 数据量已经很大, 往往先进的算法已不再有很多的假设. 但是在一些特殊领域, 例如医疗领域, 数据量并不大, 因此 GDA 这些方法仍然奏效.

此外, 不同的问题可能需要仔细地根据问题“定制”假设, 这需要一些行业经验才能做到. 在现代的机器学习/深度学习中, GDA 的使用不如以前那么多, 因为现在很多任务甚至都没有标签, 例如只有图像或者无标记文本 (语言模型) 作为  $x$ .

from December 14, 2024 to December 16, 2024

First updated: December 16, 2024

Last updated: December 18, 2024

## 6 Lecture 6: Naive Bayes, Laplace Smoothing

YouTube:Stanford CS229 Machine Learning, Naive Bayes, Laplace Smoothing, 2022, Lecture 6

### 6.1 Introduction

#### Introduction

在 GDA 中, 可以看到一个重要的假设是  $\mathbf{x}|(y = i) \sim \mathcal{N}(\boldsymbol{\mu}_i, \Sigma)$ , 但是当数据本身不能做出高斯性假设时, 例如完全是离散的数据时, GDA 就无法使用了。下面介绍的朴素贝叶斯方法(Naive Bayes)仍然可以发挥作用。

### 6.2 Naive Bayes

#### Naive Bayes – Spam classification for example

下面以邮箱中垃圾邮件分类为例, 介绍 Naive Bayes 算法的假设和流程。

在垃圾邮件分类中, 我们仍然做的是二分类,  $y \in \{0, 1\}$ , 其中  $y = 0$  表示是垃圾邮件, 是负例(negative),  $y = 1$  表示不是垃圾邮件, 是正例(positive)。而将一封邮件记为  $\mathbf{x}$ .



Figure 13: Spam Classification

**Vectorization:** 邮件的内容都是文本类型(text)的自然语言数据, 要将其转化为机器可以“读懂的语言”, 首先要进行向量化, 下面介绍一种最简单的向量化操作。

假设已有一个包含足够多英文单词的有序词汇表(Vocabulary), 其记录了  $d$  个单词, 对于一封邮件  $\mathbf{x}$ , 若某个词语在此邮件中出现了, 不管出现了多少次, 都统一记录该位置的值为 1, 否则为 0(如图 14), 因此  $\mathbf{x} = (x_1, x_2, \dots, x_d) \in \{0, 1\}^d$ , 且称向量化后的向量为特征向量(feature vector)。可以看到这种向量化方式不考虑字符的顺序, 也不考虑字符在一封信中出现的频率。

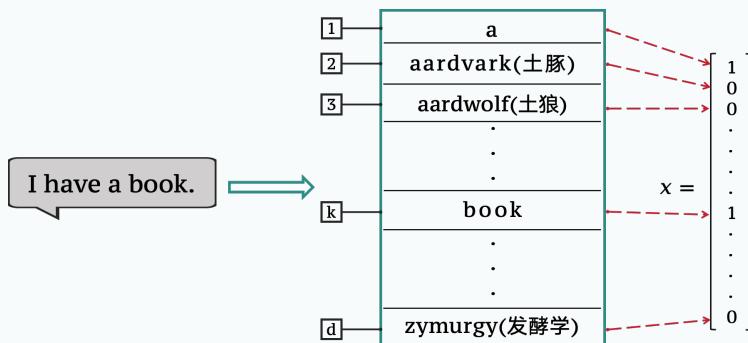


Figure 14: I have a book.

## Spam classification – Modeling to Prediction

**Modeling:** 与 GDA 相同, 此时仍然需要对  $\mathbf{x}|y$  和  $y$  进行建模, 但是由于现在  $\mathbf{x} \in \{0, 1\}^d$ , 其已不能再做高斯性的假设。在 Naive Bayes 中我们转而假设高维向量  $\mathbf{x}$  的各分量是相互条件独立的, 即  $x_1|y, x_2|y, \dots, x_d|y$  是相互独立的<sup>a</sup>:

$$\mathbb{P}(\mathbf{x}) = \mathbb{P}(x_1, x_2, \dots, x_d|y) = \prod_{i=1}^d \mathbb{P}(x_i) \quad (41)$$

**Parameters of the model:** 记正例条件下分量  $x_j = 1$  的概率为

$$\mathbb{P}(x_j = 1|y = 1) = \phi_{j|y=1} \in [0, 1], j = 1, \dots, d$$

同理记负例条件下分量  $x_j = 1$  的概率为

$$\mathbb{P}(x_j = 1|y = 0) = \phi_{j|y=0} \in [0, 1], j = 1, \dots, d$$

全部邮件中正例的概率为  $\mathbb{P}(y = 1) = \phi_y$ .

**Likelihood:** 与 GDA 中一样, 定义似然函数为:

$$\begin{aligned} L(\phi_y, \phi_{1|y=1}, \dots, \phi_{d|y=1}, \phi_{1|y=0}, \dots, \phi_{d|y=0}) \\ \text{nexamples are i.i.d} \prod_{i=1}^n \mathbb{P}(\mathbf{x}^{(i)}, y^{(i)}; \phi_y, \phi_{j|y}) \\ \text{chain rule} \prod_{i=1}^n \mathbb{P}(\mathbf{x}^{(i)}|y^{(i)}; \phi_y, \phi_{j|y}) \cdot \mathbb{P}(y^{(i)}; \phi_y, \phi_{j|y}) \\ (41) \prod_{i=1}^n \mathbb{P} \left( \mathbb{P}(y^{(i)}; \phi_y, \phi_{j|y}) \cdot \prod_{j=1}^d \mathbb{P}(x_j^{(i)}, y^{(i)}) \right) \end{aligned} \quad (42)$$

因此最大化似然函数有:

$$\arg \max L = \arg \max \log L = \arg \max \sum_{i=1}^n \left( \log \mathbb{P}(y^{(i)}; \phi_y, \phi_{j|y}) + \sum_{j=1}^d \log \mathbb{P}(x_j^{(i)}, y^{(i)}) \right) \quad (43)$$

**Solutions:** 令  $\nabla \ell(\phi_y, \phi_{j|y}) = 0$ , 最后得到

$$\phi_y = \frac{\sum_{i=1}^n \mathbb{I}(y^{(i)} = 1)}{n} : \text{fraction of positive example} \quad (44a)$$

$$\phi_{j|y=1} = \frac{\sum_{i=1}^n \mathbb{I}(x_j^{(i)} = 1, y^{(i)} = 1)}{\sum_{i=1}^n \mathbb{I}(y^{(i)} = 1)} : \text{fraction of j-th word in positive examples} \quad (44b)$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^n \mathbb{I}(x_j^{(i)} = 0, y^{(i)} = 1)}{\sum_{i=1}^n \mathbb{I}(y^{(i)} = 0)} : \text{fraction of j-th word in negative examples} \quad (44c)$$

**Prediction:** 使用条件概率公式就可以得到

$$\mathbb{P}(y=1|\mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y=1)\mathbb{P}(y)}{\mathbb{P}(\mathbf{x})} \quad (45)$$

<sup>a</sup>事实上这个假设现实中是不对的，因为我们的信件使用的自然语言一定有固定组合、逻辑等关系，因为不会都是条件独立的，但是实验表明这样假设在实际使用中效果已经很好了。

### 6.3 Laplace Smoothing

#### Laplace Smoothing

在预测时，如果有一个不常见的词语在训练集中没有出现，即 $\exists k \in [1, d], x_k$ 恒为0，那么由于我们假设 $\mathbb{P}(\mathbf{x}) = \prod_{i=1}^d \mathbb{P}(x_i)$ ，那么此时对于含有此词语的新信件，预测时式(45)分母恒为0，这显然是不合适的，因此需要使用 Laplace Smoothing 的技巧解决。

Laplace Smoothing 是指在计算涉及到多种类别相除的分式计算时，在每个类别中都加上1，即 $z \in \{0, 1, \dots, k-1\}$ ，

$$\mathbb{P}(z=j) = \frac{\sum_{i=1}^n \mathbb{I}(z^{(i)}=j) + 1}{n+k} \quad (46)$$

可以看到在使用了 Laplace Smoothing 后，我们学习到的参数在数据量小的时候不会特别极端，而在数据量大时，使用 Laplace Smoothing 对最终的结果影响也不大。

使用了 Laplace Smoothing 后，我们得到改进后的 Solution(44b)(44c)为：

$$\phi_{j|y=1} = \frac{\sum_{i=1}^n \mathbb{I}(x_j^{(i)}=1, y^{(i)}=1) + 1}{\sum_{i=1}^n \mathbb{I}(y^{(i)}=1) + 2} \quad (47a)$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^n \mathbb{I}(x_j^{(i)}=0, y^{(i)}=1) + 1}{\sum_{i=1}^n \mathbb{I}(y^{(i)}=0) + 2} \quad (47b)$$

---

from December 18, 2024 to December 19, 2024

First updated: December 19, 2024

Last updated: August 13, 2025, modify eq. (46)

## 7 Lecture 7: Kernels

Link on YouTube: Stanford CS229 Machine Learning, Kernels, 2022, Lecture 7

### 7.1 Feature Function

#### Introduction: Cubic Polynomial Regression, where data set is $\{(x^{(1)}, y^{(1)}), \dots, (x^{(n)}, y^{(n)})\}$

在基本线性回归(Linear Regression)模型  $h_\theta(x) = \theta x + \theta_0$  中<sup>a</sup>, 我们仅仅使用了  $x$  自身进行数据拟合, 但我们常需要更复杂的模型, 例如三次多项式(cubic polynomial) 模型:

$$h_\theta(x) = \theta_3 x^3 + \theta_2 x^2 + \theta_1 x_1 + \theta_0, \quad x \in \mathbb{R}$$

显然模型  $h_\theta(x)$  关于  $x$  是非线性的, 但关于参数  $\theta$  却是线性的<sup>b</sup>。我们只需做出一些改变, 就可以将关于  $x$  的非线性模型变为关于  $\theta$  的线性模型。

<sup>a</sup>其中  $\theta$  代表全体参数

<sup>b</sup>我们关注的核心实际上是在参数空间中对  $\theta$  进行优化

#### Feature Map / Feature Extractor (cubic polynomial regression case)

定义函数  $\phi : \mathbb{R} \rightarrow \mathbb{R}^4$ ,  $\phi(x) = [1 \ x \ x^2 \ x^3]^T$ , 这样模型就可以写为

$$h_\theta(x) = [\theta_0 \ \theta_1 \ \theta_2 \ \theta_3] [1 \ x \ x^2 \ x^3]^T = \theta^T \phi(x)$$

如此一来我们原本的模型输入从  $[1 \ x]$  (二维) 变成  $\phi(x)$  (四维), 因此我们构建了新数据集:

$$\{(\phi(x^{(1)}), y^{(1)}), \dots, (\phi(x^{(n)}), y^{(n)})\}$$

其中函数  $\phi(\cdot)$  被称为特征提取函数(feature function / feature extractor),  $\phi(x)$  被称为特征(features),  $x$  被称为属性(attribute)。如果  $x \in \mathbb{R}^d$ , 那么相应的  $\theta \in \mathbb{R}^d$ ; 在经过  $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^p$  的作用后, 相应的  $\theta \in \mathbb{R}^p$ 。

### 7.2 Kernel Trick / Kernelized

#### Basic Settings

在新数据集上做线性回归, 并使用梯度下降(GD)进行优化, 学习率为  $\alpha$ , 此时损失函数和参数  $\theta$  的更新过程分别为式(48a)(48b)

$$loss = \frac{1}{2} \sum_{i=1}^n \left( y^{(i)} - \theta^T \phi(x^{(i)}) \right)^2 \quad (48a)$$

$$\theta := \theta + \alpha \cdot \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) \quad (48b)$$

考虑  $x \in \mathbb{R}^d$ , 使用三次多项式回归, 那么  $\phi(x) \in \mathbb{R}^p$ , 其中  $p = 1 + |\{x_i\}| + |\{x_i \cdot x_j\}| + |\{x_i \cdot x_j \cdot x_k\}| = 1 + d + d^2 + d^3$  (考虑重复情形)。因此每一次参数  $\theta$  的更新都会有  $O(np)$  的计算量, 这是十分巨大的。

## Kernel Trick

**Proposition 1** (Rey Observation). 若  $\theta^0 = 0$ , 那么  $\theta$  可以表示为 *features* 的线性组合, 即

$$\theta = \sum_{i=1}^n \beta_i \phi(x^{(i)}) \quad (49)$$

其中  $\beta_1, \dots, \beta_n \in \mathbb{R}$ ,  $\theta^0$  是  $\theta$  的初始值。

*Proof.* 我们使用数学归纳法(induction)。

当 iteration = 0 时,  $\theta = 0 = \sum_{i=1}^n 0 \cdot \phi(x^{(i)})$  显然是 *features* 的线性组合;

当 iteration = 1 时,  $\theta := \theta + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) = \alpha \sum_{i=1}^n y^{(i)} \phi(x^{(i)})$ , 此时  $\beta_i := \alpha y^{(i)}$

假设 iteration =  $t$  时结论成立, 即  $\theta = \sum_{i=1}^n \beta_i \phi(x^{(i)})$ , 则 iteration =  $t + 1$  时

$$\begin{aligned} \theta &:= \theta + \alpha \sum_{i=1}^n (y^{(i)} - \theta^T \phi(x^{(i)})) \phi(x^{(i)}) = \sum_{i=1}^n \left( (\beta_i + \alpha (y^{(i)} - \theta^T \phi(x^{(i)}))) \phi(x^{(i)}) \right) \\ \beta_i &:= \beta_i + \alpha (y^{(i)} - \theta^T \phi(x^{(i)})) \end{aligned} \quad (50)$$

□

**Trick 1: parameters storage** 已知  $\theta \in \mathbb{R}^p$ , 但是现在我们不需要直接存储  $\theta$  而是存储系数  $\beta_i$ , 这只需要  $n$  个存储, 不妨假设  $p \gg n$ , 那么这将带来一些计算量的降低。

**Problem 1: still related to  $\theta$**  从式(50)可以看到  $\beta$  的更新实际上与  $\theta$  相关, 仍需  $O(p)$  的计算量, 因此我们需要找到  $\beta$  只依赖于自身的更新模式:

$$\begin{aligned} \beta_i &:= \beta_i + \alpha (y^{(i)} - \theta^T \phi(x^{(i)})) = \beta_i + \alpha \left( y^{(i)} - \left( \sum_{j=1}^n \beta_j \phi(x^{(j)}) \right)^T \phi(x^{(i)}) \right) \\ &= \beta_i + \alpha (y^{(i)} - \sum_{j=1}^n \beta_j \langle \phi(x^{(j)}), \phi(x^{(i)}) \rangle) \end{aligned} \quad (51)$$

**Problem 2: inner product costs a lot** 但是可以看到在计算内积  $\langle \phi(x^{(j)}), \phi(x^{(j)}) \rangle$  时, 仍需要  $O(np)$  的计算量, 下面有两个简化方式(**Trick 2**):

**preprocessed:**  $\langle \phi(x^{(j)}), \phi(x^{(j)}) \rangle$  可以被预处理(preprocessed)并存储, 因为  $\forall i, j$ , 内积计算过一次后就无需重复计算了

**formal compute:** 无需精确地将复杂的  $\phi(\cdot)$  带入  $\langle \phi(x^{(j)}), \phi(x^{(j)}) \rangle$  中也可以进行计算:

$$\begin{aligned} \langle \phi(x), \phi(z) \rangle &= [1, x_1, \dots, x_d, x_1^2, \dots, x_d^2, x_1^3, \dots, x_d^3] [1, x_1, \dots, x_d, x_1^2, \dots, x_d^2, x_1^3, \dots, x_d^3]^T \\ &= 1 + \sum_{i=1}^d x_i z_i + \sum_{i=1, j=1}^d x_i x_j \cdot z_i z_j + \sum_{i, j, k=1}^d x_i x_j x_k z_i z_j z_k \\ &= 1 + \langle x, z \rangle + \sum_{i=1}^d x_i z_i \sum_{j=1}^d x_j z_j + \sum_{i=1}^d x_i z_i \cdot \sum_{j=1}^d x_j z_j \cdot \sum_{k=1}^d x_k z_k = 1 + \langle x, z \rangle + \langle x, z \rangle^2 + \langle x, z \rangle^3 \end{aligned}$$

### Algorithm and Time complexity

可以看到计算单个内积的时间复杂度为  $O(d)$ 。称  $K : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ ,  $K(x, z) = \langle \phi(x), \phi(z) \rangle$  为核函数(kernel function)。这样的算法称为Kernel method for regression, 最终的算法为:

#### Algorithm 1 Algorithm for Kernel Computation and $\beta$ Update

```

1: Input: Data points  $\{x^{(i)}\}_{i=1}^n$ , labels  $\{y^{(i)}\}_{i=1}^n$ , learning rate  $\alpha$ 
2: Initialize:  $\beta = 0 \in \mathbb{R}^n$ 
3: for  $i = 1, \dots, n$  do
4:   for  $j = 1, \dots, n$  do
5:     Compute the kernel  $K(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$ 
6:   end for
7: end for
8: for  $i = 1, \dots, n$  do
9:    $\beta_i \leftarrow \beta_i + \alpha \left( y^{(i)} - \sum_{j=1}^n \beta_j K(x^{(i)}, x^{(j)}) \right)$ 
10: end for

```

考虑维度  $i, j$  后, 计算所有的kernel 的总时间复杂度为  $O(n^2d) = O(n^2)$ , 因此对于  $\beta$ , 每一次迭代的时间复杂度为  $O(n^2)$ 。可以看到单次迭代  $\beta$  的时间复杂度变化为  $O(np) \rightarrow O(n^2)$ , 因此若  $n \ll p$ , 那么就会带来很大的计算量提升。<sup>a</sup>

<sup>a</sup>整个算法的时间复杂度为  $O(n^2d) + O(n^2 \cdot T)$ , 其中  $T$  是迭代次数

### 7.3 Design Kernel Function

#### Change the algorithm flow

从上述推导过程来看, 我们将  $K(\cdot, \cdot)$  从特征函数的内积形式化成为原数据的内积进而减少了计算量。在进行测试时, 给定一个  $x$ , 根据式(50)和Proposition 1需要计算

$$\theta^T \phi(x) = \left( \sum_{i=1}^n \beta_i \phi(x^{(i)}) \right)^T \phi(x) = \sum_{i=1}^n \beta_i K(x^{(i)}, x)$$

<sup>a</sup>, 因此我们的算法最终仅与核函数  $K$  有关, 而核函数是特征函数的内积形式, 因此显示定义了特征函数就显式定义了核函数, 反过来直接定义核函数也就隐式定义了特征函数, 因此<sup>b</sup>:

$$\text{Design a good } \phi \Leftrightarrow \text{Design a good } K$$

**Theorem 2** (necessary and equivalent condition for designing a  $K$ ). Suppose  $x^{(1)}, \dots, x^{(n)}$  are  $n$  data points, and let  $K \in \mathbb{R}^{n \times n}$  be kernel matrix, in which  $K_{ij} = K(x^{(i)}, x^{(j)})$ .  $K$  is a valid kernel function  $\Leftrightarrow \forall x_1^{(n)}, \dots, x^{(n)}$ , the kernel function  $K$  is PSD (positive semi-definite).

*Proof.* 暂略 □

如此一来我们的工作流程就反过来变为了:

设计一个好的kernel function  $K \rightarrow$  确定  $K$  有效(通过Thm 2 或者解出  $\phi(\cdot)$ )  $\rightarrow$  运行算法

<sup>a</sup>可以看到此时测试时仍然需要训练数据, 与之前训练好之后就无需训练数据只需训练好的参数的情况不同

<sup>b</sup>这里的  $K$  的设计要 make sense, 要有意义

## 7.4 Extended content

### Extended content

常用的核函数有

1.  $K(x, z) = (\langle x, z \rangle + c)^k$ , for  $k = 2$ ,  $\phi(x) = \begin{bmatrix} c \\ \sqrt{2}x_1 \\ \sqrt{2}x_2 \\ \vdots \\ x_1^2 \\ \vdots \\ x_d^2 \end{bmatrix}$
2. Gaussian Kernel:  $K(x, z) = \exp(-\frac{\|x-z\|_2^2}{2\sigma^2}) = \langle \phi(x), \phi(z) \rangle$ , 但是实际上 $\phi(\cdot)$ 很复杂, 是一个无穷维形式

有时特征也被视为相似性度量(similarity metric), 可以将 $K(x, z)$ 看作一种测量 $x$ 和 $z$ 的方式(measure)

kernel method 真正的改变是将 $O(np) \rightarrow O(n^2)$ , 当 $n \ll p$ 时这较有效, 但是现在的数据集往往很大, 因此kernel method 现在并不是一个很高效和常用的方法; 同时核函数的设计并不很容易, 同时可解释性也是一个困难。事实上NN也可以写为 $\theta^T \phi_w(x)$ , 其中 $w$ 代表NN中的参数, 而此 $\phi_w$ 是直接从数据中学习得到的, 而非人工设计的, 但是有效性却出奇的好。

---

from October 5, 2024 to October 8, 2024

First updated: October 8, 2024

Last updated: December 23, 2024

## 8 Lecture 8: Neural Networks

YouTube:Stanford CS229 Machine Learning, Neural Networks 1, 2022, Lecture 8

### 8.1 Outline

#### Outline

- Outline  $\left\{ \begin{array}{l} 1. \text{ Supervised learning with non-linear model} \\ 2. \text{ Neural networks} \left\{ \begin{array}{l} 1. \text{ How to define } h_{\theta}(\mathbf{x})? \\ 2. \text{ How to compute } \nabla J^{(i)}(0)? \end{array} \right. \end{array} \right.$

### 8.2 Review

#### Linear and Non-linear models

**Data set:**  $\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^n, \mathbf{x}^{(i)} \in \mathbb{R}^d, y^{(i)} \in \mathbb{R}; \quad h_{\theta} : \mathbb{R}^d \rightarrow \mathbb{R}$

#### Linear regression

**Model:**  $h_{\theta}(\mathbf{x}) = \theta^T \mathbf{x} + b$  (linear)

**Cost / Loss function:**  $J(\theta) = \frac{1}{2} \sum_{i=1}^n (h_{\theta}(\mathbf{x}^{(i)}) - y^{(i)})^2$

**Optimize<sup>2</sup>:** run Gradient Decent(GD) or Stochastic Gradient Decent(SGD) to optimize

#### Non-linear model: Kernel method

**Model:**  $h_{\theta}(\mathbf{x}) = \theta^T \phi(\mathbf{x})$  (linear in parameters, non-linear in  $\mathbf{x}$ )

可以看到即使是非线性模型核方法也只是对输入  $\mathbf{x}$  非线性, 但是对参数  $\theta$  仍然是线形的, 那么如果希望完全是非线性的模型该怎么办呢? 例如  $h_{\theta}(\mathbf{x}) = \sqrt{\theta_1^3 x_2 + \sqrt{\theta_5 x_4}}$ .

详见附录L

### 8.3 Neural network

#### Neural network – Introduction

例如我们考虑房价预测问题, 使用房屋面积线性预测房屋价格, 如图15所示.

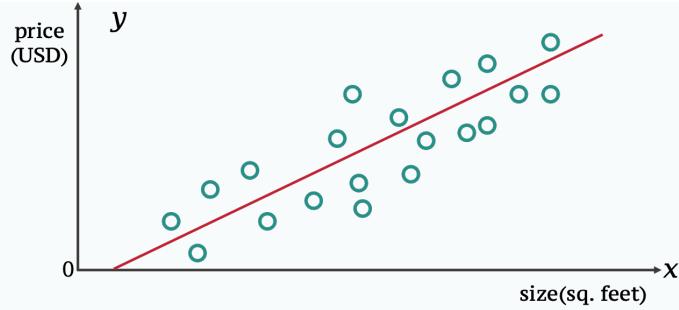


Figure 15: house price prediction

但是时候会有两个明显的问题:

1. 房屋面积与价格可能并非简单的线性关系, 而是有更加复杂的非线性关系
  2. 线形模型可能导致某些面积下房屋价格为负, 显然不合常理, 图15就是这种情况
- 线性整流函数 (Rectified Linear Unit, ReLU) 事实上可以解决上述问题,

$$\text{ReLU}(x) = \max\{0, x\}$$

因此显然ReLU 1. 是非线性函数<sup>a</sup>; 2. 可以防止房价出现负数的情况. 因此我们将输出写为

$$h_{\theta}(x) = \text{ReLU}(\omega x + b) \quad (52)$$

在高维  $\mathbf{x} \in \mathbb{R}^d$  情形下, 写为:

$$h_{\theta}(\mathbf{x}) = \text{ReLU}(\boldsymbol{\omega} \mathbf{x} + b), \quad \mathbf{x} \in \mathbb{R}^d, \boldsymbol{\omega} \in \mathbb{R}^d, b \in \mathbb{R} \quad (53)$$

在神经网络(Neural networks)中, 一个式(53)被称为一个神经元(neuron), 深度学习要做的是堆叠若干个、若干层这样的神经元, 且上一层的输出就是下一层的输入. 此时  $\boldsymbol{\omega} \mathbf{x} + b$  被称为预激活值(pre-activation), 其被激活函数作用后称为激活值(activation):

output of activation  $\rightarrow$  input of the next neuron  $\rightarrow$  pre-activation  $\rightarrow$  activation

<sup>a</sup>非线形体现在“转折点”, 事实上在深度学习中这就是激活函数(activation function)

### Neural network – Example

以  $\mathbf{x} \in \mathbb{R}^4$ , 为例, 其中  $x_1$ : size,  $x_2$ : # bedrooms,  $x_3$ : zip code,  $x_4$ : wealth

除了这些特征外, 我们可以根据经验再依据这些特征构建一些中间变量(intermediate variables)以更好进行预测, 例如

1.  $a_1$ : max family size, 与 size 和 # bedrooms 相关, 故  $a_1 = \text{ReLU}(\omega_1 x_1 + \omega_2 x_2 + b_1)$
2.  $a_2$ : walkable, 与 zip code 相关, 故  $a_2 = \text{ReLU}(\omega_3 x_3 + b_2)$
3.  $a_3$ : school quality, 与 zip code 和 wealth 相关, 故  $a_3 = \text{ReLU}(\omega_4 x_3 + \omega_5 x_4 + b_3)$

这样最后得到的输出(如图16所示)为:

$$h_{\theta}(\mathbf{x}) = \omega_6 a_1 + \omega_7 a_2 + \omega_8 a_3 + b_4, \quad \theta = \{\omega_1, \dots, \omega_8, b_1, \dots, b_4\} \quad (54)$$

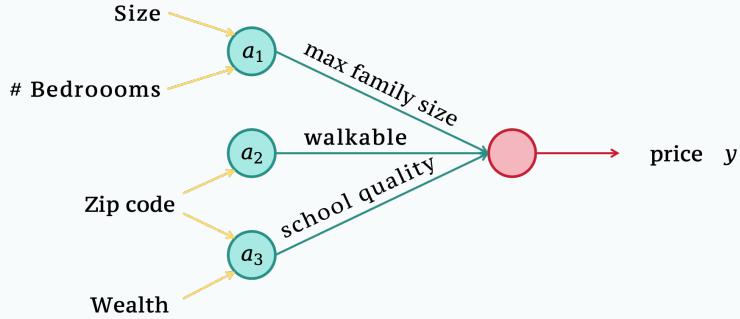


Figure 16: house price prediction example

### Neural network – General case

从上面的例子可以发现，我们组建新的特征时是基于我们的先验知识的，这需要很高的代价，并且很多时候也并不全部准确。因此我们可以考虑将层间所有的神经元都链接起来，让模型自己提取有用的特征，即如图17所示，此时称这种网络为全链接神经网络(fully connected neural network, FCNN)。图中共有两层神经元，除开输出层的都称为隐藏层。

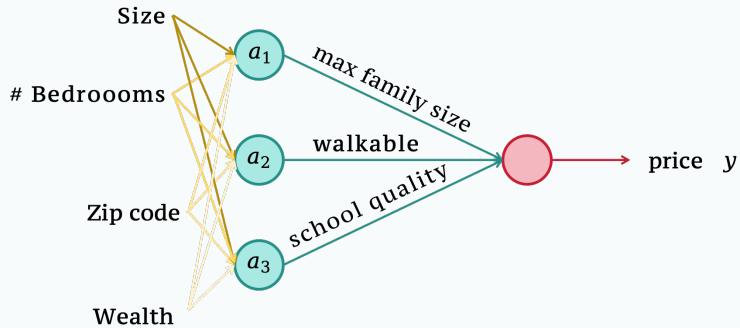


Figure 17: fuly connected neural networks

这样之后，我们就可以将此全链接神经网络数学地写为：

$$\begin{cases} a_1 = \text{ReLU}(\mathbf{w}_1^{[1]}\mathbf{x} + b_1^{[1]}), & \mathbf{w}_1^{[1]} \in \mathbb{R}^4, \mathbf{x} \in \mathbb{R}^4, b_1^{[1]} \in \mathbb{R} \\ a_2 = \text{ReLU}(\mathbf{w}_2^{[1]}\mathbf{x} + b_2^{[1]}), & \mathbf{w}_2^{[1]} \in \mathbb{R}^4, \mathbf{x} \in \mathbb{R}^4, b_2^{[1]} \in \mathbb{R} \\ a_3 = \text{ReLU}(\mathbf{w}_3^{[1]}\mathbf{x} + b_3^{[1]}), & \mathbf{w}_3^{[1]} \in \mathbb{R}^4, \mathbf{x} \in \mathbb{R}^4, b_3^{[1]} \in \mathbb{R} \end{cases} \quad (55a)$$

$$\Rightarrow h_{\theta}(\mathbf{x}) = \mathbf{w}^{[2]}\mathbf{a} + b^{[2]}, \quad \mathbf{w}^{[2]} \in \mathbb{R}^3, \mathbf{a} \in \mathbb{R}^3, b^{[2]} \in \mathbb{R} \quad (55b)$$

**Vectorization:** 为得到更简洁的表示，并且帮助 GPU 并行化，需要将上述数学表示向量化。同时我们将其推广到共  $r$  层神经元、隐藏层中每一层有  $m_k$ ,  $k = 1, \dots, r-1$  个神经元的一般情形：

$$\mathbf{W}^{[1]} = \begin{bmatrix} \cdots & (\mathbf{w}_1^{[1]})^T & \cdots \\ \cdots & (\mathbf{w}_2^{[1]})^T & \cdots \\ \cdots & \cdots & \cdots \\ \cdots & (\mathbf{w}_{m_1}^{[1]})^T & \cdots \end{bmatrix} \in \mathbb{R}^{m_1 \times d}, \quad \mathbf{b}^{[1]} = \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{m_1}^{[1]} \end{bmatrix}, \in \mathbb{R}^{m_1 \times 1} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^{d \times 1} \quad (56)$$

其中上标<sup>[1]</sup>表示第1层，这样得到第一层的预激活值(pre-activation)为

$$z^{[1]} = W^{[1]}x + b^{[1]}, \quad \underbrace{\begin{bmatrix} z_1^{[1]} \\ \vdots \\ z_{m_1}^{[1]} \end{bmatrix}}_{z^{[1]} \in \mathbb{R}^{m_1 \times 1}} = \underbrace{\begin{bmatrix} \cdots & w_1^{[1]\top} & \cdots \\ \cdots & w_2^{[1]\top} & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & w_{m_1}^{[1]\top} & \cdots \end{bmatrix}}_{W^{[1]} \in \mathbb{R}^{m_1 \times d}} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix}}_{x \in \mathbb{R}^{d \times 1}} + \underbrace{\begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ \vdots \\ b_{m_1}^{[1]} \end{bmatrix}}_{b^{[1]} \in \mathbb{R}^{m_1 \times 1}} \quad (57)$$

第一层的激活值(activation)为

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ \vdots \\ a_{m_1}^{[1]} \end{bmatrix} = \begin{bmatrix} \text{ReLU}(z_1^{[1]}) \\ \text{ReLU}(z_2^{[1]}) \\ \vdots \\ \text{ReLU}(z_{m_1}^{[1]}) \end{bmatrix} \triangleq \text{ReLU}(z^{[1]}) \quad (58)$$

第 $k$ 层的预激活值和激活值分别为 $z^{[k]} = W^{[k]}a^{[k-1]} + b^{[k]}$ 和 $a^{[k]} = \text{ReLU}(z^{[k]})$ :

$$\underbrace{\begin{bmatrix} z_1^{[k]} \\ \vdots \\ z_{m_k}^{[k]} \end{bmatrix}}_{z^{[k]} \in \mathbb{R}^{m_k \times k}} = \underbrace{\begin{bmatrix} \cdots & w_1^{[k]\top} & \cdots \\ \cdots & w_2^{[k]\top} & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & w_{m_k}^{[k]\top} & \cdots \end{bmatrix}}_{W^{[k]} \in \mathbb{R}^{m_k \times m_{k-1}}} \underbrace{\begin{bmatrix} a_1^{[k-1]} \\ a_2^{[k-1]} \\ \vdots \\ a_{k-1}^{[k-1]} \end{bmatrix}}_{a^{[k-1]} \in \mathbb{R}^{m_{k-1} \times 1}} + \underbrace{\begin{bmatrix} b_1^{[k]} \\ b_2^{[k]} \\ \vdots \\ b_{m_k}^{[k]} \end{bmatrix}}_{b^{[k]} \in \mathbb{R}^{m_k \times 1}} \quad (59)$$

$$a^{[k]} = \begin{bmatrix} a_1^{[k]} \\ a_2^{[k]} \\ \vdots \\ a_{m_k}^{[k]} \end{bmatrix} = \begin{bmatrix} \text{ReLU}(z_1^{[k]}) \\ \text{ReLU}(z_2^{[k]}) \\ \vdots \\ \text{ReLU}(z_{m_k}^{[k]}) \end{bmatrix} \triangleq \text{ReLU}(z^{[k]}) \quad (60)$$

最后全部层的向前传播公式为:

$$\begin{aligned} a^{[1]} &= \text{ReLU}(W^{[1]}x + b^{[1]}) \\ a^{[2]} &= \text{ReLU}(W^{[2]}a^{[1]} + b^{[2]}) \\ &\vdots \\ a^{[r-1]} &= \text{ReLU}(W^{[r-1]}a^{[r-2]} + b^{[r-1]}) \\ h_{\theta}(x) &= W^{[r]}a^{[r-1]} + b^{[r]} \end{aligned} \quad (61)$$

### Why do we need activation function?

激活函数(activation function)在神经网络中是非常重要的，其作用是引入非线性关系，从而增强模型的表达能力，常见的激活函数有：

Comparison of Common Activation Functions			
Function	Formula	Range	Derivative Characteristics
Sigmoid	$\sigma(x) = \frac{1}{1+e^{-x}}$	(0, 1)	Vanishing gradient for large $ x $
Tanh	$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	(-1, 1)	Zero-centered, vanishing gradient for large $ x $
ReLU	$\text{ReLU}(x) = \max(0, x)$	$[0, \infty)$	Non-saturating, gradient is 0 for $x < 0$
Leaky ReLU	$\text{LReLU}(x) = \begin{cases} x, & x \geq 0 \\ \alpha x, & x < 0 \end{cases}$	$(-\infty, \infty)$	Non-zero gradient for $x < 0$ (controlled by $\alpha$ )
Softmax	$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$	(0, 1)	Used for multi-class classification, sum of outputs = 1

<sup>1</sup> ReLU: Rectified Linear Unit;  $\alpha$  in Leaky ReLU is typically a small positive constant (e.g., 0.01).

<sup>2</sup> Sigmoid and Tanh can suffer from the vanishing gradient problem, especially in deep networks.

Table 5: Comparison of Common Activation Functions

如果没有激活函数作用, 那么以两层神经网络为例:

$$a^{[1]} = W^{[1]}x + b^{[1]}, \quad h_{\theta}(x) = W^{[2]}a^{[1]} + b^{[2]} = W^{[2]}W^{[1]}x + W^{[2]}b^{[1]} + b^{[2]} \quad (62)$$

其仍然是一个线形的, 并且等价于一层线性层作用.

在 kernel methods 中:

$$a = \phi(x), \quad h(x) = Wa + b = W\phi(x) + b \quad (63)$$

我们使用 feature map  $\phi(x)$  自主选择了需要的特征(features), 但是在神经网络中, 我们实际上是通过学习得到的  $\phi(\cdot)$ , 因此在隐藏层也称为 features / representations learning, 中间得到的  $a^{[k]}$  也被称为 features / representations.

## 8.4 Questions

### Questions: kernel and Neural networks

如果在神经网络中每一层再加上一部 kernel 的作用会怎么样呢? 即:

$$\begin{aligned} a^{[r-1]} &= \phi_{\beta}(a^{[r-2]}), \quad \beta = \{W^{[1]}, \dots, W^{[r-1]}, b^{[1]}, \dots, b^{[r-1]}\} \\ h(x) &= W^{[r]}a^{[r-1]} + b^{[r]} \end{aligned} \quad (64)$$

1. 这样会导致神经网络表达能力更强吗?
2. 还是说理论上可以证明即使加了 kernel methods 二者的表达能力相同?
3. 即使理论上表达能力相同, 实际效果是否有差异?
4. 如果理论上就强那可以弥补带来的计算量上升吗?

## 9 Lecture 9: Backprop

YouTube:Stanford CS229 Machine Learning, Neural Networks 2(backprop), 2022, Lecture 9

### 9.1 Outline

#### Outline

- Outline
- 1. Review
  - 2. Differential circuits / networks
  - 3. Chain rule(Preliminary)
  - 4. Back propagation

### 9.2 Review

#### Loss and parameters update

在神经网络中, 第  $i$  个样本(sample)  $x^{(i)}$  产生的 loss 为

$$J^{(i)} = \frac{1}{2}(y^{(i)} - h_{\theta}(x^{(i)}))^2$$

用该样本更新参数  $\theta$  的算法为

$$\theta := \theta - \alpha \nabla J^{(i)}(\theta)$$

其中  $\alpha$  为学习率。此时问题的重点就变成如何求得梯度  $\nabla J^{(i)}(\theta)$ .

### 9.3 Differentiable circuits / networks

#### Differentiable circuits / networks

在介绍如何求得梯度之前, 先了解一下梯度是否能计算、好计算是必要的, 下面先介绍 Differentiable circuits / network, 其是 second-order method 和 meta learning 的基础.

**Definition 1** (Differentiable circuits / networks). 称由一系列基本算符( $+$   $-$   $\times$   $/$ ) 和基本函数(例如  $\cos$ ,  $\sin$ ,  $\exp$ ,  $\log$ ,  $ReLU$  等)组合而成的组合体为 *differentiable circuits / networks*<sup>a</sup>.

对于此 Differentiable circuits, 事实上可以说明用起计算导数是可以做到且并不困难:

**Theorem 3.** 设一实值函数  $f : \mathbb{R}^l \rightarrow \mathbb{R}$  可被大小为  $N$  的 *Differentiable circuit* 计算出, 那么其梯度  $\nabla f \in \mathbb{R}^l$  可以被大小为  $O(N)$  *Differentiable circuit* 计算出, 且计算时间复杂度为  $O(N)$ .

**Note 6.** 1. *Theorem 3* 中隐含地假设了  $N > l$ , 因为要计算出所有  $l$  维度的值.

2. *Theorem 3* 表明对于一个 *circuit*, 计算函数值  $f$  与计算其梯度  $\nabla f$  所需时间差不多, 因此计算梯度并不比计算 loss 本身更困难!

3. 应用于 *neural network* 中,  $f$  即  $J^{(i)}(\theta)$ ,  $l = \# \text{parameters}$ ,  $N = O(\# \text{parameters})$

**Corollary 1.** 在相同的设置下,  $\forall v \in \mathbb{R}^l$ , 计算  $\nabla^2 f(x) \cdot v$  的时间复杂度为  $O(N + l)$ .

*Proof.* 事实上, 如果先计算  $\nabla^2 f(x)$  就有  $O(l^2)$  的计算量, 再做内积就有  $O(l^2 + l)$  就算量。但是, 令  $g(x) := \langle \nabla f(x), v \rangle : \mathbb{R}^l \rightarrow \mathbb{R}$ , 由 Theorem 3 可知  $g(x)$  的计算量为  $O(N + l)$ , 再次使用 Theorem 3 可知  $\nabla g = \nabla \langle \nabla f, v \rangle$  的计算量依然为  $O(N + l)$ .  $\square$

<sup>a</sup>更多介绍可见 Differentiable Circuits And PyTorch

### 9.3.1 Preliminary – Chain rule

#### Chain rule

**Settings:**  $J(\theta_1, \theta_2, \dots, \theta_p)$  为自变量为  $\theta_1, \theta_2, \dots, \theta_p$  的函数  
并有中间变量  $g_1 = g_1(\theta_1, \theta_2, \dots, \theta_p), \dots, g_k = g_k(\theta_1, \theta_2, \dots, \theta_p)$ , 因此  $J$  也可以写为  $J(g_1, g_2, \dots, g_k)$

#### Chain rule:

$$\frac{\partial J}{\partial \theta_i} = \sum_{j=1}^k \frac{\partial J}{\partial g_j} \cdot \frac{\partial g_j}{\partial \theta_i}$$

若  $\theta \in \mathbb{R}$ , 则  $\frac{\partial J}{\partial \theta} \in \mathbb{R}$ , 若  $\theta \in \mathbb{R}^d$ , 则  $\frac{\partial J}{\partial \theta} = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \vdots \\ \frac{\partial J}{\partial \theta_d} \end{bmatrix} \in \mathbb{R}^d$ , 若  $A \in \mathbb{R}^{d_1 \times d_2}$ , 则  $\frac{\partial J}{\partial A} = \begin{bmatrix} \frac{\partial J}{\partial A_{11}} & \dots & \frac{\partial J}{\partial A_{1d_2}} \\ \vdots & \ddots & \vdots \\ \frac{\partial J}{\partial A_{d_11}} & \dots & \frac{\partial J}{\partial A_{d_1d_2}} \end{bmatrix} \in \mathbb{R}^{d_1 \times d_2}$

### 9.3.2 Back propagation

#### Chain for two layer Neural network

**Settings:** 我们考虑简单的两层神经网络:

$$\begin{aligned} z &= W^{[1]}x + b^{[1]} \in \mathbb{R}^m, \quad x \in \mathbb{R}^d, W^{[1]} \in \mathbb{R}^{m \times d}, b^{[1]} \in \mathbb{R}^d \\ a &= \sigma(z) \in \mathbb{R}^m \\ h_\theta(x) &= o = W^{[2]}a + b^{[2]} \in \mathbb{R}^1 \\ J &= \frac{1}{2}(y - o)^2 \end{aligned}$$

**Abstraction:** 我们首先抽象地形式化看看计算梯度会有什么, 先设:

$$J = J(z), z = Wu + b, x \in \mathbb{R}^d, W \in \mathbb{R}^{m \times d}, b \in \mathbb{R}^d$$

那么根据链式法则就有:

$$\underbrace{\frac{\partial J}{\partial W}}_{\in \mathbb{R}^{m \times d}} = \underbrace{\frac{\partial J}{\partial z}}_{\in \mathbb{R}^{m \times 1}} \cdot \underbrace{u^T}_{\in \mathbb{R}^{1 \times d}}, \quad \frac{\partial J}{\partial b} = \frac{\partial J}{\partial z}$$

$$\begin{aligned} \frac{\partial J}{\partial W_{ij}} &= \sum_{k=1}^m \frac{\partial J}{\partial z_k} \frac{\partial z_k}{\partial W_{ij}} = \sum_{k=1}^m \frac{\partial J}{\partial z_k} \cdot \frac{\partial (W_{k1}u_1 + W_{k2}u_2 + \dots + W_{kd}u_d + b_k)}{\partial W_{ij}} \\ &= \frac{\partial J}{\partial z_k} \cdot \frac{\partial (W_{i1}u_1 + W_{i2}u_2 + \dots + W_{id}u_d + b_k)}{\partial W_{ij}} = \frac{\partial J}{\partial z_k} \cdot u_j \end{aligned}$$

**Application to 2-layers NN:** 我们想要计算损失函数关于神经网络参数的梯度, 根据上面的结果就有:

$$\frac{\partial J}{\partial W^{[1]}} = \frac{\partial J}{\partial z} \cdot x^T, \quad \frac{\partial J}{\partial b^{[1]}} = \frac{\partial J}{\partial z}, \quad \frac{\partial J}{\partial W^{[2]}} = \frac{\partial J}{\partial o} \cdot a^T, \quad \frac{\partial J}{\partial b^{[2]}} = \frac{\partial J}{\partial o}$$

由于  $a = \sigma(z) \in \mathbb{R}^m$ ,  $J = J(a)$ , 因此有:

$$\underbrace{\frac{\partial J}{\partial z}}_{\in \mathbb{R}^m} = \underbrace{\frac{\partial J}{\partial a}}_{\in \mathbb{R}^m} \odot \underbrace{\sigma'(z)}_{\in \mathbb{R}^m}$$

其中  $\odot$  指 entry-wise, 因为激活函数  $\sigma$  的作用也是 entry-wise 的.

对于  $\frac{\partial J}{\partial a}$ , 有:

$$\frac{\partial J}{\partial a} = \frac{\partial J}{\partial o} \cdot \frac{\partial o}{\partial a} = w^{[2]} \cdot \frac{\partial J}{\partial o}$$

对于  $\frac{\partial J}{\partial o}$ , 有:

$$\frac{\partial J}{\partial o} = -(y - o)$$

### Chain rule for deep neural networks

**Settings:** 考虑一个  $r$  层的深度神经网络:

$$\begin{aligned} z^{[1]} &= w^{[1]}x + b^{[1]} \in \mathbb{R}^m \\ a^{[1]} &= \sigma(z^{[1]}) \\ z^{[2]} &= w^{[2]}a^{[1]} + b^{[2]} \\ &\dots \\ a^{[r-1]} &= \sigma(z^{[r-1]}) \\ z^{[r]} &= w^{[r]}a^{[r-1]} + b^{[r]} \\ J &= \frac{1}{2}(y - z^{[r]})^2 \end{aligned}$$

**Chain rule:** 我们要计算损失  $J$  对第  $k$  层参数  $W^{[k]}, b^{[k]}$  的梯度。由于

$$\begin{aligned} z^{[k]} &= W^{[k]} \cdot a^{[k-1]} + b^{[k]} \\ J &= J(z^{[k]}) \end{aligned}$$

因此:

$$\frac{\partial J}{\partial W^{[k]}} = \frac{\partial J}{\partial z^{[k]}} \cdot \left( a^{[k-1]} \right)^T$$

由于

$$\begin{aligned} a^{[k]} &= \sigma(z^{[k]}) \\ J &= J(a^{[k]}) \end{aligned}$$

因此:

$$\frac{\partial J}{\partial z^{[k]}} = \frac{\partial J}{\partial a^{[k]}} \odot \sigma'(z^{[k]})$$

由于

$$z^{[k+1]} = W^{[k+1]} \cdot a^{[k]} + b^{[k+1]}$$
$$J = J(z^{[k+1]})$$

因此:

$$\frac{\partial J}{\partial a^{[k]}} = (W^{[k+1]})^T \frac{\partial J}{\partial a^{[k+1]}}$$

...

...

## Summary

在神经网络向前传播中, 如图18计算顺序是:

forward pass:  $z^{[1]}, a^{[1]} \rightarrow z^{[2]}, a^{[2]}, \dots \rightarrow z^{[r]}, a^{[r]}$

$x \rightarrow \boxed{MM} \rightarrow z^{[1]} \rightarrow \boxed{activation} \rightarrow a^{[1]} \rightarrow \boxed{MM} \rightarrow z^{[2]} \rightarrow \dots \rightarrow z^{[r]} \rightarrow J$

Figure 18: Forward pass

其中  $MM$  是一个模块(module), 指矩阵乘法(matrix multiplication)  $W^{[k]}x + b^{[k]}$ .  
但是在计算梯度时, 计算的顺序是:

backward pass:  $\frac{\partial J}{\partial z^{[r]}} = -(y - z^{[r]}) \rightarrow \frac{\partial J}{\partial a^{[r-1]}} = (W^{[r]})^T \frac{\partial J}{\partial z^{[r]}}$ ,  $\frac{\partial J}{\partial z^{[r-1]}} = \frac{\partial J}{\partial a^{[r-1]}} \odot \sigma'(z^{[r-1]})$   
 $\rightarrow \dots \rightarrow \frac{\partial J}{\partial a^{[k]}}, \frac{\partial J}{\partial z^{[k]}} \rightarrow \dots$

因此计算梯度时,  $MM$ -module 原本的输出变成了输入, 输入变成了输出, 这也是反向传播名称的由来.

---

from December 31, 2024 to January 18, 2025

Last updated: January 18, 2025

# 10 Lecture 10: Bias - Variance, Regularization

YouTube:Stanford CS229 Machine Learning, Bias - Variance, Regularization, 2022, Lecture 10

## 10.1 Introduction

### Introduction

首先介绍一些会使用到的名词:

**training loss / error / cost:** 训练损失 / 误差, 指训练期间训练数据的误差:

$$J(\theta) = \frac{1}{n} \sum_{i=1}^n \left( y^{(i)} - h_{\theta}(x^{(i)}) \right)^2$$

**testing loss / error / cost:** 测试损失 / 误差, 指测试期间测试数据的误差:

$$L(\theta) = \mathbb{E}_{(x,y) \sim \mathcal{D}} [(y - h_{\theta}(x))^2]$$

其中  $\theta$  为训练得到的模型参数,  $(x, y) \sim D$  指测试数据  $(x, y)$  满足分布  $\mathcal{D}$ , 并且测试数据不能包含测试数据集中数据。由于对分布求期望只能理论上的写出来, 在实际中, 是使用足够多的 i.i.d. 样本  $(x_{\text{test}}^{(1)}, y_{\text{test}}^{(1)})^{\top}, \dots, (x_{\text{test}}^{(m)}, y_{\text{test}}^{(m)})^{\top} \stackrel{\text{i.i.d.}}{\sim} \mathcal{D}$  进行计算。

**Generation gap:** 指测试误差与训练误差之间的差距:

$$\text{Generation gap: } L(\theta) - J(\theta)$$

**Note 7.** *Generation gap* 可以用于评估训练是否过拟合/欠拟合等情况。一般来说  $L(\theta) - J(\theta) \geq 0$ , 现实情况中一般不会出现  $J(\theta) > L(\theta)$  的情况。我们期望的情况是  $J(\theta)$  和  $L(\theta) - J(\theta)$  都很小, 但是由于二者分属两个过程, 故同时控制二者是很困难的, 因此 *generation gap* 不能够直接控制也很难控制。

## 10.2 Bias-Variance Theory

### 2 Failure Mode

**$L(\theta)$  is big:** 当  $L(\theta)$  较小时, 认为模型的泛化性能较好, 因此我们并不希望  $L(\theta)$  很大, 但是当出现这种情况时, 一般有两种情形:

1. **overfitting**, 过拟合: 此时  $J(\theta) \approx 0$  很小, 但是  $L(\theta)$  很大, 说明训练得到的模型过于逼近训练数据, 对于新的数据泛化能力很差。
2. **underfitting**, 欠拟合: 此时  $J(\theta)$  也很大, 说明模型对训练数据都没有很好拟合。

当发生 overfitting 时, 说明模型学到的并非真实的数据满足的模式, 而是学习到了噪声的 “spurious pattern”, 因此为检测模型是否过拟合可以通过重新提取(redrawn)数据训练, 若发生了过拟合那么不同数据训练得到的模型会有很大差异。

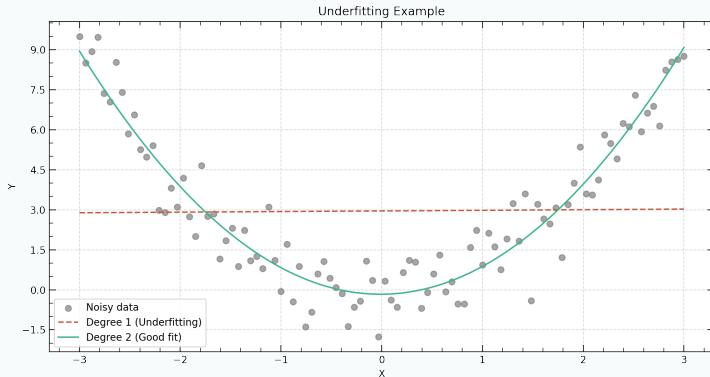


Figure 19: Underfitting example

### Bias and Variance

Bias-Variance theory 用于定性分析已训练模型的测试误差  $L(\theta)$ 。其将测试误差分解为 bias 与 variance 两部分的和，用于解释模型复杂度<sup>a</sup>与  $L(\theta)$  的关系，在理论上可以证明：

$$L(\theta) = \text{bias}^2 + \text{variance} \quad (65)$$

其中 bias 定义为 在数据量无限的前提下模型可以实现的最小误差；variance 定义为 选用的模型在不同数据集上的表现带来的差异性大小（类似于不稳定性的衡量，类比方差）。

Bias 主要由模型表达能力过低导致，而与训练数据量关系不大。

Variance 主要由两个原因导致：1. 数据量少；2. 模型表达能力过强。相应地减小 variance 的方法有：1. 增加更多 training data；2. 使用更简单的模型。一般来说训练时都会将所有数据用于训练，因此主要讨论后者。

Bias-Variance theory 指出 bias 随模型复杂度递增而递减，variance 随模型复杂度递增而增加，因此二者求和后的测试误差  $L(\theta)$  会随模型复杂度先增加后减少，意味着从欠拟合向过拟合转变，如下图所示：

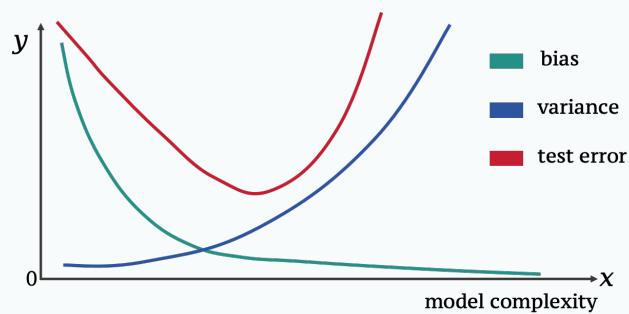


Figure 20: Bias-Variance decomposition

<sup>a</sup>模型复杂度指模型的复杂程度，可以简单理解为参数量，例如简单线性模型、二次多项式模型、五次多项式模型、神经网络模型的模型复杂度在递增。

### 10.3 Double decent

#### Double decent

Double decent, 即双下降现象是机器学习与深度学习中非常重要的现象，其最早的观察可以追溯到1989年[8]，在2019年重新被提出[9]，并且成为近期的研究热门。一般认为，随着模型的复杂度上升，会从欠拟合向过拟合转变，因此测试误差会先降低再上升，但是双下降现象说明，如果继续增加模型复杂度，测试误差在达到一个顶点时会“反常地”开始下降（如图21所示），此时模型会得到很好的泛化能力。

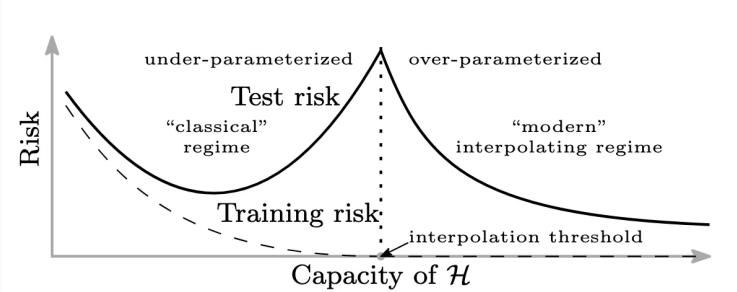


Figure 21: Double decent shown in [9], where  $\mathcal{H}$  is the model / function.

**Note 8.** 对于 *double decent* 现象，有如下几点说明：

1. 训练误差(risk)第二次下降时（图中标记为 *interpolation threshold*）一般是数据量与参数量相等时，即  $n \approx d$ ，其中  $n = \# \text{parameters}$ ,  $d = \# \text{data points}$ . 当  $n > d$  后 risk 开始再次下降。
2. 实际上还有一种 *data-wise double decent*，是指横坐标为  $\# \text{data points}$ ，其二次下降的临界点仍然一般是  $n \approx d$ .[10]
3. 简单的模型也可以有很大的模型参数量或模型复杂度，例如对于线性模型，使用核方法可以使其同样拥有很大参数量。

#### Some explanations for double decent

当模型的参数量与训练数据量相近，即  $n \approx d$  时，模型的泛化性能会急剧下降。可以从如下两个方面解释（只说明结论，不说明原理）：

1. 模型的参数  $\theta$  的范数(norm)随着参数量(# parameters)的上升是先上升后下降的趋势，而最大 norm 存在于  $n \approx d$  时，这也导致了此时模型性能很差，因此我们可以尽量选择一个  $\theta$  的 norm 小的模型。当  $n \gg d$  时优化算法存在隐式正则化(implicit regularization effect)使得norm 变小。
2. 更深层次来说，原因在于模型训练过程中产生的某些随机矩阵(random matrix)在  $n \approx d$  (近似于方阵) 时表现不佳。若有兴趣更深层次的原因可以参见[11]。

此外，对于 norm，其可以作为衡量模型复杂度(complexity) 的一种指标，但是并不唯一，也很难说是最好的度量指标。同样地，# parameters 也说不好是不是最好的指标，因为在训练过程中，很可能很多的参数的系数会趋于 0。

# 11 Lecture 11: Feature / Model selection, ML Advice

YouTube:Stanford CS229 Machine Learning, Feature / Model selection, ML Advice, 2022, Lecture 11

## 11.1 Complexity Measure

### Complexity

在上一节中 overfitting 与 underfitting 所围绕的核心是 model complexity, 部分可供选择的 complexity measure 如下:

1. **# parameters.** 即参数数量. 但缺陷在于训练完的模型可能很多参数都很小甚至为 0, 导致实际参数量  $< \# \text{ parameters}$ .
2. **Norm of parameters.** 常用 norm 有  $\|\cdot\|_1, \|\cdot\|_2$ . 可以解决某些参数为 0 的问题, 但缺陷在于当 norm 过小时容易受到噪声影响.
3. **Lipschitzness / Smoothness.** 用于衡量模型表示函数的光滑性 (这里暂略) .

### How to Decrease Model Complexity? – Regularization

解决 overfitting 的方法之一是降低 model complexity. 针对不同的 complexity measure, 其中一种方式是直接减小模型参数量, 另一种方式是降低模型参数的 norm, 即正则化(regularization).

简单来说, regularization 是在 training loss 中加上附加项以促使得到较低复杂度的模型:

$$\text{new loss} = J(\theta) + \lambda \cdot \mathcal{R}(\theta), \quad (66)$$

其中  $J(\theta)$  为 loss function,  $\mathcal{R}(\theta)$  被称为 regularizer,  $\lambda$  为 regularization strength, 用以平衡 loss 与 regularizer 之间的 trade-off.

常用的 regularizer 有:

1.  $\mathcal{R}(\theta) = \frac{1}{2} \|\theta\|_2^2$ , 被称为  $L_2$ -regularization 或 weight decay<sup>a</sup>.
2.  $\mathcal{R}(\theta) = \|\theta\|_0 = \# \text{ non-zero in } \theta$ , 被称为 sparsity<sup>b</sup>.
3.  $\mathcal{R}(\theta) = \|\theta\|_1 = \sum_{i=1}^d |\theta_i|$ , 是  $\|\theta\|_0$  的替代, 解决了其不可导的问题.

**Note 9.** Regularization 实际上反映了模型的结构(structure), 这是建立在先验知识(prior belief)上的, 例如若希望只使用一部分特征, 那么就可以选择  $\|\theta\|_0$  和  $\|\theta\|_1$ ; 若相信每一个特征都会有作用, 需要将他们组合起来使用, 就需要选择  $\|\theta\|_2$ . 对于线性模型,  $J(\theta) + \|\theta\|_1$  被称为 **LASSO**; 对于深度学习任务, 使用的主要是  $L_2$ -regularization.

<sup>a</sup>事实上二者对于 gradient decent 等价, 详见附录. 但是对于 Adam 等并不等价, 详见[12]

<sup>b</sup>在线性模型中  $\theta^T x = \sum_{i=1}^d \theta_i x_i$ , 分量  $\theta_j$  为 0 说明模型并没有选择特征  $x_j$ , 故称 sparsity.

## 11.2 Implicit Regularization

### Implicit Regularization

**Motivation.** 在现代深度学习模型往往是过参数化的(over-parameterized), 此时并没有很强的正则化(例如 regularization strength  $\lambda$  很小或为 0)但是仍然有很好泛化能力. 一种解释是在优化过程中发生了隐式正则化(implicit regularization).

对于 training loss 来说, 其 loss landscape 可能有很多局部极小点(local minima)甚至有数个极小点, 但是 implicit regularization 会让模型最终选择更好的, 即使得 test error 更小的模式. 如下图22 所示, 对于 training error 来说  $A, B$  均为极小点, 但是对于 test error 来说  $B$  点是极小点, implicit regularization 的作用就是使得在优化过程中选择  $B$  模型使其泛化能力更好(test error 更小)<sup>a</sup>.

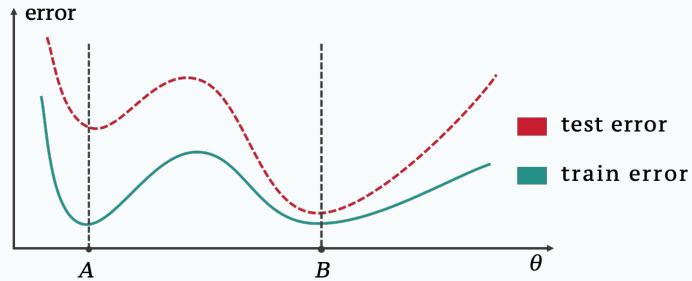


Figure 22: Implicit regularization example

**Example – Linear Model.** 考虑过参数化(over-parameterized)的线性模型: 给定训练集  $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \dots, (x^{(n)}, y^{(n)})\}$ , 其中  $x^{(i)} \in \mathbb{R}^d$ ,  $n \ll d$ , 损失函数  $J(\theta) = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \theta^T x^{(i)})^2$ . 此时 training error 存在很多 global minima<sup>b</sup>, 但结论表明: 使用初始化于 0 点的梯度下降(GD)优化该模型会收敛至 norm 最小的解, 即

$$\begin{aligned} & \arg \min \|\theta\|_2^2 \\ & \text{s.t. } J(\theta) = 0 \end{aligned} \tag{67}$$

在下面图23 的例子中, 考虑  $n = 1, d = 3$  的可视化情形, 此时  $J(\theta) = (\theta_1 x_1 - y_1)^2 + (\theta_2 x_2 - y_2)^2 + (\theta_3 x_3 - y_3)^2$ , 最终通过梯度下降收敛的点即为点  $A$ , 此时向量  $\vec{A}$  与平面  $J(\theta) = 0$  垂直.

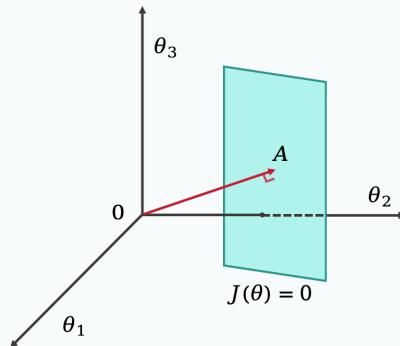


Figure 23: Caption

<sup>a</sup>此处不做更多详细数学化解释, 此领域仍然是一个开放性探索领域.

<sup>b</sup>向量方程  $\theta^T x = y$  共  $n$  个方程、 $d$  个未知量, 由于  $n \ll d$  因此有很多组解, 导致存在很多 global minima.

### 11.3 Validation

#### How to choose hyperparameters?

在介绍完 regularization 之后可以看到 model size, regularizer, regularization strength, optimization, learning rate 等参数的选择也会对模型效果产生影响, 我们将上述并非模型参数的参数称为超参数(hyperparameter). 参数与超参数的调整是一个循环的过程:

调整参数 → 调整超参数 → 在更优超参数上重新调整参数 → …

由于调整超参数的目的是提高模型的泛化能力, 需要在模型“没见过”的数据上进行, 因此不能使用训练集; 而如果在测试集上调整超参数会导致模型拟合测试集的数据, 使得模型评价失去意义, 故需要新划分出集合用于调整超参数, 称之为验证集(validation set / development set). 因此三种集合的作用如下:

1. **Training set.** 用于调整模型参数(parameter)
2. **Validation set / Development set.** 用于调整模型超参数(hyperparameter)
3. **Test set.** 用于测试模型泛化能力, 只使用一次

**Note 10.** 三种数据集的划分一般而言是随机选取数据进行归类, 但并不完全是, 例如对于时序数据就需要按照时序排列进行划分以模拟时序预测的情形. 模型在验证集上表现好并不一定代表在测试集上表现好, 但有结果表明二者具有强相关性[13].

### 11.4 Machine Learning Advice

#### Machine Learning Advice

参考 ML Advice(Clipart day), 在构建机器学习系统(ML system)时有如下步骤:

1. **Acquire data.** 在获取 data 时要收集尽量好的数据, 而非 spam data, 例如期待 training data 与 test data 的分布要相近.
2. **Look at your data.** 观察数据, 确保数据无异常, 实际上在每一步后都需要做这件事.
3. **Create train / validation / test set.** 进行数据集划分, 常见比例为 6 : 2 : 2.
4. **Create / Refine a specification.** 建立一个好的评定标准, 如合适的损失函数.
5. **Build model.** 构建模型, 实际上这一步是最简单的.
6. **Measurement.** 建立合适的模型效果评定标准.
7. **Repeat.**

---

from February 9, 2025 to February 12, 2025

Last updated: August 15, 2025

## 12 Lecture 12: K-Means, GMM(non EM), Expectation Maximization

Link on YouTube: Stanford CS229 Machine Learning, K-Means, GMM, 2022, Lecture 12

### 12.1 Introduction

#### Introduction

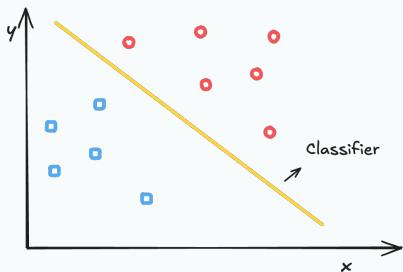
本节内容介绍了无监督学习方法，具体而言介绍了三种聚类算法：

1. K-means: 最经典的迭代型聚类算法，使用 **hard assignment** 进行数据点分配
2. Gaussian Mixture Model: 使用 **soft assignment** 进行分配，并通过 **E-M** 算法求解

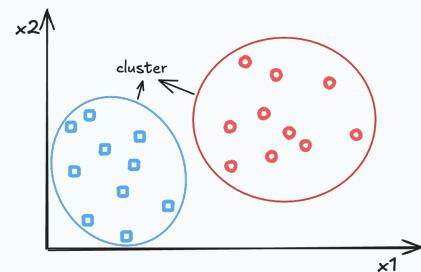
### 12.2 K-Means

#### K-Means: Introduction

在前面所有课程中所介绍的机器学习算法均为有监督学习(supervised learning)，其一重要特征是数据集  $\{x_i, y_i\}$  中存在标签(label)  $y_i$ 。而当标签并不存在时就需要使用无监督学习(unsupervised)方法(如图24b所示)。



(a) Supervised learning with labels



(b) Unsupervised learning without labels

Figure 24: Comparison between supervised learning and unsupervised learning.

由于无法利用标签信息作为拟合目标，因此并没有所谓的“正确答案”<sup>a</sup>，故较监督学习实现更困难，为此需要更多的假设。例如一些工作假设数据存在某种潜在结构。因此对比监督学习而言，无监督学习 1. 需要更强的假设 2. 只能得出较弱的理论保证。

<sup>a</sup>例如图 24b 中将数据点分为几类并无标准答案

#### K-Means: Setup

K-means 算法目的是将无标签的数据分为  $k$  类，将每一类称为一“聚类”(cluster)，例如图 24b 中有两个聚类。

**Data:** 给定无标签数据集  $\{x^{(1)}, \dots, x^{(n)}\}$ ，其中  $x^{(i)} \in \mathbb{R}^d$ 。

**Do:** 预先设定  $k$  个聚类，找到一种分配方式  $C$  将数据归类于相应的聚类中：

$$C^{(i)} = j : \text{point } x^{(i)} \rightarrow \text{cluster } j, i = 1, \dots, n, j = 1, \dots, k \quad (68)$$

## K-Means: Algorithm

K-means 算法是迭代型算法(见算法 2), 其流程如下:

1. 随机选取  $k$  个聚类中心点(cluster center point)  $\mu^{(i)}$ , 即定义了  $k$  个 cluster
2. 根据距离的远近将所有的点分配给最近的聚类中心点, 即形成了  $k$  个 cluster  $\Omega_j$
3. 对每个聚类中的点的坐标计算均值  $\mu^{(j)} = \frac{1}{|\Omega_j|} \sum_{i \in \Omega_j} x^{(i)}$ , 作为新的聚类中心点
4. 重复步骤 2-3 直至聚类不再变化

---

### Algorithm 2 K-means Clustering Algorithm

---

**Require:** Dataset  $X = \{x_1, x_2, \dots, x_n\}$ , number of clusters  $k$

- 1: Randomly initialize  $k$  cluster centers  $\mu^{(j)}, j = 1 \dots, k$
- 2: **repeat**
- 3:     **Cluster assignment:** Assign each data point to the nearest cluster center:

$$C^{(i)} = \arg \min_j \|\mu^{(j)} - x_i\|^2, j = 1 \dots, k$$

- 4:     **New center:** Update each cluster center as the mean of the points assigned to it:

$$\mu^{(j)} = \frac{1}{|\Omega_j|} \sum_{i \in \Omega_j} x^{(i)}, \Omega_j = \{i : C^{(i)} = j\}, j = 1 \dots, k$$

- 5: **until** cluster assignments and cluster centers are no longer change.
- 

**Note 11.** 定义算法损失函数为

$$J(C, \mu) = \sum_{j=1}^k \sum_{i \in \Omega_j} \|x^{(i)} - \mu^{C^{(i)}}\|^2 \quad (69)$$

则其单调递减, 因此 K-means 算法一定会收敛. 并且抽象来看 K-means 算法是在对  $J(C, \mu)$  做梯度下降. 需要额外注意的是:

1. 由于聚类问题本身的 *NP-Hard* 性质, K-means 并不能保证收敛至 *global minima*
2. 在计算新的聚类中心点时并不将现有的中心点纳入计算中, 因为实际上此中心点是通过计算平均值得出的虚拟数据点.
3. K-means 容易受初始化(*initialization*) 的影响, 目前最主流的 K-means 算法是由 *Stanford* 提出的 K-means ++ [14], 其已经成为 *scikit-learn* 中 K-means 的默认算法.

**Note 12.** 在 K-means 中聚类数  $k$  是提前给定的, 实际上在没有相关信息(*domain knowledge*)的前提下并没有所谓的最优  $k$ . 此处我们更聚焦于能否给定想要的聚类数后通过算法自动得出一个令人满意的 *cluster assignment* 方案.

**Note 13.** 无监督学习方法也可以和监督学习一起使用. 例如当标签充满噪声以至于无法信任时, 就可以先使用无监督学习对数据内部进行探索以获取更多信息以帮助进行监督学习.

## 12.3 Mixture of Gaussian

### Mixture of Gaussian: Example

**Motivation:** Mixture of Gaussian 算法的一个简单例子来自于天文学中的光子(photon)检测问题: 地球上放置探测器(light detector)用以接受来自太空中的光子. 其源自普通行星(regular star)和类星体(quasar)两类星体, 其中普通行星的脉冲(pulse)向各个方向均匀发散, 而类星体的脉冲则相对集中. 探测器在接受到光子时无法知晓其来源, 即无标签.

**Given:** 探测器上的光子分布位置.

**Do:** 将每个光子分配给一个来源, 但由于无法完全确定, 因此采用 **soft assignment** 方式进行分配, 即用概率表示每个光子来自某一类星体的可能性, 即:

$$P(z^{(i)} = j) : \text{probability that photon } i \text{ comes from source } j$$

其中  $z^{(i)}$  实际为一随机变量, 表示将点  $i$  分配给 cluster  $j$ .

**Challenge:** 1. 光子的来源可能有很多, 但是此时我们假设已知有  $k$  个来源 (与 K-means 假设相同); 2. 不同来源的光子呈现的强度(intensity) 和形状(shape) 不同.

**Assumption:** 假设每个来源的光子呈 Gaussian 分布 ( $\mu_j, \sigma_j$ ).

**Note 14.** 这里并不假设每个来源的光子的数量相同, 因为有可能某个来源的光子集中且数量更多, 例如占比 90%.

### Mixture of Gaussian: $\mathbb{R}^1$ for Simplicity

以一维为例, 如图25a 所示, 若已知类别, 只需要进行简单的 Gaussian fit 即可 (例如 GDA), 但目前的困难在于如图25b 所示, 只能获取数据的分布但类别未知.



Figure 25:  $\mathbb{R}^1$  for simplicity.

**Given:** 给定数据  $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}$ , 和类别数 (cluster number)  $k \in \mathbb{R}^*$ .

**Do:** 找到 soft assignment, 即概率分布  $P(z^{(i)} = j), i = 1, \dots, n, j = 1, \dots, k$ , 其中  $z^{(i)}$  实际为一随机变量, 表示将点  $i$  分配给 cluster  $j$ , 并将其称为 latent.

**Note 15.** 实际上目前我们的任务可以认为是未知分布反推数据的生成过程, 如果反过来思考, 正向的数据过程应该是: 每一个类别的数据分别满足 Gaussian:

$$x^{(i)} | (z^{(i)} = j) \sim \mathcal{N}(\mu_j, \sigma_j^2) \quad (70)$$

每个类别中数据量满足一定分布(例如某一类数据点占 90%)

$$z^{(i)} \sim \text{multinational}(\Phi), \sum_{j=1}^k \phi_j = 1, \phi_j \geq 0, \quad (71)$$

在得到了每一类数据点如何分布、每一类数据点占比后根据 Bayes 进行数据点生成:

$$P(x^{(i)}, z^{(i)}) = P(x^{(i)} | z^{(i)}) \cdot P(z^{(i)}) \quad (72)$$

## Mixture of Gaussian: Algorithm

*Gaussian Mixture Model (GMM)* 算法是著名的 *E-M* 算法(Expectation–Maximization algorithm) 的实例应用，其分为两步：

1. (E-step) 给定数据和当前 latent values  $(\Phi, \mu, \sigma)$ ，预测 latent variable  $z^{(i)}$ ，即

$$\begin{aligned} w_j^{(i)} &= P(z^{(i)} = j | x^{(i)}; \Phi, \mu, \sigma) = \frac{P(z^{(i)} = j, x^{(i)}; \Phi, \mu, \sigma)}{P(x^{(i)}; \Phi, \mu, \sigma)} \\ &= \frac{P(x^{(i)} | z^{(i)} = j; \Phi, \mu, \sigma) P(z^{(i)} = j)}{\sum_{l=1}^k P(x^{(i)} | z^{(i)} = l; \Phi, \mu, \sigma) P(z^{(i)} = l)} = \frac{\mathcal{N}(\mu_j, \sigma_j^2) \cdot \phi_j}{\sum_{l=1}^k \mathcal{N}(\mu_l, \sigma_l^2) \cdot \phi_l} \end{aligned} \quad (73)$$

2. (M-step) 给定当前  $P(z^{(i)}=j)$  估计值  $w_j^i$ ，使用 MLE 估计 observed parameters:

$$\phi_j = \frac{1}{n} \sum_{i=1}^n \approx \text{fraction of points from cluster } j \quad (74)$$

**Note 16.** 这里 *observed parameters* 指目前可以观测到的，例如目前分配下每个聚类中的点数量，相较 *latent  $z^{(i)}$*  而言，其是可观测到的。

---

from August 26, 2025 to August 28, 2025

Last update: December 15, 2025

## 13 Lecture 13: GMM (EM)

Link on YouTube: Stanford CS229 Machine Learning, GMM (EM), 2022, Lecture 13

### 13.1 Introduction

#### Introduction

本节内容介绍聚类算法中 *GMM* 算法及 *EM* 算法. 首先介绍两个基本工具: 凸性和 Jensen 不等式, 在此基础上, 通过 *EM algorithm as MLE* 介绍 *EM* 算法原理, 进一步通过 *GMM as EM*, 将 *GMM* 作为 *EM* 算法的实例介绍 *GMM*.

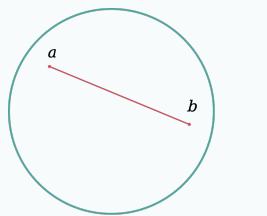
### 13.2 Convexity and Jensen's inequality

#### Convexity

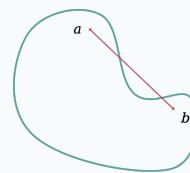
**Definition 2. Convex and Nonconvex.** 称一个集合  $\Omega$  是凸的(convex), 若对于集合  $\Omega$  中任意两点  $a, b \in \Omega$ , 其二者之间的连线也均在该集合中. 数学上即:

$$\forall a, b \in \Omega, \quad (1 - \lambda)a + b \in \Omega, \quad \lambda \in [0, 1].$$

如图31 所示, 图26a 中集合为凸的, 但图26b 中集合为非凸的.



(a) Convex



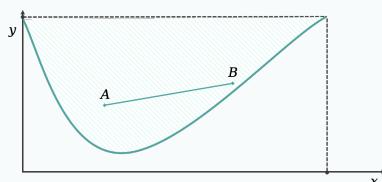
(b) Nonconvex

Figure 26: Comparison between convex and nonconvex.

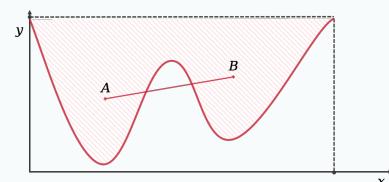
**Definition 3. Convex Function.** 对于函数  $f$ , 定义其 graph 为  $G_f = \{(x, y) : y \geq f(x)\}$ , 则称  $f$  为凸函数(convex function)若其 graph 是凸的. 数学上即 (推导见附录O) :

$$\forall \lambda \in [0, 1], \lambda f(a) + (1 - \lambda)f(b) \geq f(\lambda a + (1 - \lambda)b) \quad (75)$$

如图27a 中函数为凸函数, 其中每一条弦 (即两点连线) 均在函数之上(every chord is beyond the function); 图27b 中为非凸函数, 不能保证每一条弦均在函数之上.



(a) Convex function



(b) Nonconvex function

Figure 27: Comparison between convex and nonconvex function

**Theorem 4.** 若函数  $f$  的二阶导恒大于等于 0, 即  $f''(x) \geq 0, \forall x$ , 则  $f$  为凸函数.

*Proof.* 见附录P □

**Definition 4. Strictly Convex.** 称  $f$  是严格凸函数(*strictly convex*), 若  $f''(x) > 0, \forall x \in \text{dom}$ .

**Definition 5. Concave Function.** 相对于凸函数, 称函数  $g$  为凹函数(*concave function*)若  $-g$  为凸函数.

### Jensen's inequality

**Theorem 5. Jensen's Inequality.** 若  $f$  为凸函数, 则

$$\mathbb{E}[f(x)] \geq f(\mathbb{E}(x)). \quad (76)$$

例如,  $P(x = a) = \lambda, P(x = b) = 1 - \lambda$ , 则  $\mathbb{E}[f(x)] = \lambda f(a) + (1 - \lambda)f(b), f(\mathbb{E}(x)) = f(\lambda a + (1 - \lambda)b)$ , 因此结合凸函数定义, 显然 Jensen's inequality 成立.

## 13.3 EM Algorithm as MLE

### EM Algorithm as MLE

给定数据  $\{x_i\}_{i=1}^n$  和参数  $\theta$ , 标准的最大似然估计(maximum likelihood estimation, MLE) 形式为:

$$l(\theta) = \sum_{i=1}^n \log P(x_i; \theta), \quad (77)$$

在已知信息的基础上, 若希望对问题引入一些特定(但可能未知)的结构时就会引入 latent variable, 例如 Lecture 12 的光子接收器中对每一个光子的来源比例做出  $\Theta$  的结构性假设. 此时  $P(x_i; \theta)$  变为:

$$P(x; \theta) = \sum_z P(x, z; \theta). \quad (78)$$

### EM Algorithm: Idea

如图28 所示, 由于原目标函数  $l(\theta)$  较难优化(一般无凸、凹性), 因此 EM 算法首先在  $\theta^{(t)}$  处使用存在凸/凹性的新目标函数  $L_t(\theta)$  替换  $l(\theta)$ , 因此比  $l(\theta)$  更容易优化, 接着再更新  $\theta^{(t+1)} = \arg \max_{\theta} L_t(\theta)$ . 其中替代函数(surrogate)  $L_t(\theta)$  需要满足两点性质:

1. *Low bound*:  $L_t(\theta) \leq l(\theta)$ , 即替代函数是原目标函数的下界
2. *Tight*:  $L_t(\theta^{(t)}) = l(\theta^{(t)})$ , 即替代函数与原目标函数在当前迭代点的函数值相等

因此, EM 算法简单来说分为两步:

1. (E-step): 给定  $\theta^{(t)}$  寻找  $L_t(\theta)$
2. (M-step): 给定  $L_t(\theta)$ , 更新  $\theta^{(t+1)} = \arg \max_{\theta} L_t(\theta)$

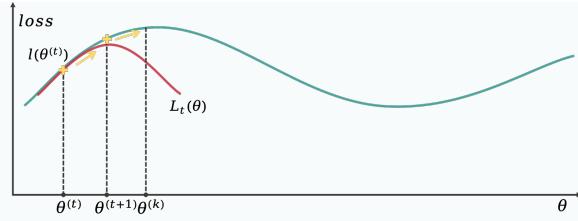


Figure 28: EM algorithm.

### EM Algorithm: Select $L_t$

**Lower bounds:** 在引入 latent variable  $z$  后,  $\log P(x; \theta)$  变为:

$$\log P(x; \theta) = \log \sum_z P(x, z; \theta) = \log \sum_z \frac{Q(z) \cdot P(x, z; \theta)}{Q(z)}, \forall Q(z), \quad (79)$$

其中  $Q(z)$  为一离散分布, 满足  $\sum_z Q(z) = 1, Q(z) \geq 0$ .

式(79)可以进一步改写, 并应用 Jensen's inequality 可得:

$$\begin{aligned} \log \sum_z \frac{Q(z) \cdot P(x, z; \theta)}{Q(z)} &= \log \mathbb{E}_{z \sim Q(z)} \frac{P(x, z; \theta)}{Q(z)} \geq \mathbb{E}_{z \sim Q(z)} \left[ \log \frac{P(x, z; \theta)}{Q(z)} \right] \\ &= \sum_z Q(z) \cdot \log \frac{P(x, z; \theta)}{Q(z)} \end{aligned} \quad (80)$$

**Note 17.** 注意此时满足  $\sum_z Q(z) = 1, Q(z) \geq 0$  的  $Q(z)$  都满足式(80), 这样我们就获得了  $l(\theta)$  的下界集合, 集合中的函数  $L$  均满足  $L(\theta) \leq l(\theta)$ .

**Note 18.** 式(80)中  $\sum_z Q(z) \cdot \log \frac{P(x, z; \theta)}{Q(z)}$  被称为 **Evidence-based lower bound (ELBO)**, 即

$$ELBO(x, Q, z) = \sum_z Q(z) \cdot \log \frac{P(x, z; \theta)}{Q(z)}. \quad (81)$$

因此有

$$l(\theta) \geq \sum_{i=1}^n ELBO(x^{(i)}, Q^{(i)}; \theta). \quad (82)$$

在获得了  $l(\theta)$  的下界集合后, 符合要求的  $L_t$  还需要其满足 **Tight** 的性质.

**Make it tight.** 从式(80)可以看到, 不等号是由求  $\log \frac{P(x, z; \theta)}{Q(z)}$  的期望产生的, 因此为消除不等号可以令其为独立于  $z$  的常数, 即令  $\log \frac{P(x, z; \theta)}{Q(z)} = C$ . 此时令  $Q(z) = P(z|x; \theta)$  就有:

$$\log \frac{P(x, z; \theta)}{Q(z)} = \log \frac{P(z|x; \theta) \cdot P(x; \theta)}{P(z|x; \theta)} = \log P(x; \theta). \quad (83)$$

由于消除了关于  $z$  的随机性, 此时就有

$$l(\theta^{(t)}) = \sum_{i=1}^n \log P(x_i; \theta^{(t)}) = \sum_{i=1}^n ELBO(x^{(i)}, Q^{(i)}; \theta^{(t)}), \quad (84)$$

其中  $Q^{(i)} = P(z^{(i)}|x^{(i)}; \theta^{(t)})$ .

**Note 19.** 由于  $Q^{(i)}$  的选取与  $x, \theta$  相关, 因此针对  $\theta^{(i)}$  点处的替代函数而言, 因为选取了  $Q^{(i)} = P(z|x^{(i)}; \theta^{(t)})$ , 因此只有  $\theta^{(t)}$  处满足等式, 而其他  $\theta_\beta$  处只有  $Q = P(z|x^{(i)}; \theta_\beta)$  才满足等式, 因此只能满足不等式  $l(\theta_\beta) \geq \sum_{i=1}^n ELBO(x^{(i)}, Q^{(i)}; \theta^{(i)})$ .

**Note 20.**  $z^{(i)}$  是对问题作一个潜在的结构假设, 例如在 Lecture 12 的光子实验中, 对于每个 sample  $i$ ,  $z^{(i)}$  是对其来源的假设, 例如如果目前有两种来源, 那么可以假设  $P(z^{(i)} = 1) = \phi_1, P(z^{(i)} = 2) = \phi_2$ , 这就是对此问题的一种潜在的结构假设.

### EM Algorithm: Conclusion

综上所述 EM 算法分为如下两步:

1. (E-step): 给定  $\theta^{(t)}$ , 令  $Q^{(i)} = P(z^{(i)}|x^{(i)}; \theta), i = 1, \dots, n$
2. (M-step): 给定  $L_t(\theta)$ , 更新  $\theta^{(t+1)} = \arg \max_\theta L_t(\theta) = \sum_{i=1}^n ELBO(x^{(i)}, Q^{(i)}; \theta)$
1. 由于  $\theta^{(t+1)} = \arg \max_\theta L_t(\theta) = \sum_{i=1}^n ELBO(x^{(i)}, Q^{(i)}; \theta)$ , 因此  $l(\theta^{(t+1)}) \geq l(\theta^{(t)})$ , 故 EM 算法一定会收敛/终止.
2. 但是由于每次更新都选择了更优的参数, 因此并不能够跳出局部最优点, 故不能保证最终能收敛至全局最优.

## 13.4 EM for GMM

### EM for GMM

**Recall.** 根据 Lecture 12 中 GMM 模型:

$$P(x^{(i)}, z^{(i)}) = P(x^{(i)}|z^{(i)}) \cdot P(z^{(i)}), \quad (85)$$

其中  $z^{(i)} \sim \text{Multinomial}(\Phi), \phi_i \geq 0, \sum_i \phi_i = 1, P(x^{(i)}|z^{(i)}) \sim \mathcal{N}(\mu_j, \sigma_j^2)$ .

记  $Q^{(i)}(z^{(i)}) = P(z^{(i)}|x^{(i)}; \theta)$ , 则此时 EM 算法中 M-step 为:

$$\max_{\Theta} \sum_{i=1}^n \sum_{z^{(i)}} Q^{(i)}(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q^{(i)}(z^{(i)})}, \quad (86)$$

其中  $\Theta = \{\Phi, \mu, \Sigma\}$ .

**Derivative for  $\mu_j$ .** 记  $f_i(\theta) = \sum_{z^{(i)}} Q^{(i)}(z^{(i)}) \log \frac{P(x^{(i)}, z^{(i)}; \theta)}{Q^{(i)}(z^{(i)})}, w_j^{(i)} = Q^{(i)}(z^{(i)} = j) = P(z^{(i)} = j|x^{(i)}; \theta)$ , 则:

$$f_i(\theta) = \sum_j w_j^{(i)} \log \frac{\frac{1}{2\pi|\Sigma|^{1/2}} \exp\{-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)\} \phi_j}{w_j^{(i)}}, \quad (87a)$$

$$\nabla_{\mu_j} \sum_{i=1}^n f_i(\theta) = \sum_{i=1}^n \nabla_{\mu_j} (w_j^{(i)} - \frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)) = -\frac{1}{2} \sum_{i=1}^n w_j^{(i)} \Sigma_j^{-1} (x^{(i)} - \mu_j). \quad (87b)$$

令梯度  $\nabla_{\mu_j} \sum_{i=1}^n f_i(\theta) = 0$ , 即可以得到:

$$\mu_j = -\frac{\sum_i w_j^{(i)}(x^{(i)} - \mu_j)}{\sum_i w_j^{(i)}}. \quad (88)$$

**Derivative for  $\phi_j$ .** 由于  $\phi_j$  需要满足约束条件  $\sum_j \phi_j = 1, \phi_j \geq 0$ , 因此根据 Lagrangian, 重新构造损失函数为<sup>a</sup>:

$$\sum_{i=1}^n f_i(\theta) = \sum_{i=1}^n w_j^{(i)} \log \phi_j + \lambda(\sum_j \phi_j - 1) \quad (89)$$

再对  $\phi_j$  求导可得:

$$\nabla_{\phi_j} \sum_{i=1}^n f_i(\theta) = \sum_{i=1}^n w_j^{(i)} \nabla_{\phi_j} \log \phi_j + \nabla_{\phi_j} \lambda(\sum_j \phi_j - 1) = \sum_{i=1}^n \frac{w_j^{(i)}}{\phi_j} + \lambda. \quad (90)$$

令  $\nabla_{\phi_j} \sum_{i=1}^n f_i(\theta) = 0$ , 得

$$\sum_j \phi_j = 1 = -\frac{1}{\lambda} \sum_{i,j} w_j^{(i)} = -\frac{n}{\lambda} \Rightarrow \lambda = -n. \quad (91)$$

---

<sup>a</sup>这里为方便书写, 省略了一些与求导无关的项.

---

from August 26, 2025 to August 28, 2025

Last update: December 17, 2025

## 14 Lecture 14: Factor Analysis / PCA

Link on YouTube: Stanford CS229 Machine Learning, Factor Analysis / PCA, 2022, Lecture 14

### 14.1 Introduction

#### Introduction

本节内容主要介绍高维小样本情形下的子空间型无监督学习方法，包括基于概率模型的因子分析（Factor Analysis）与非概率方法的主成分分析（PCA）。

### 14.2 Factor Analysis

#### Factor Analysis: Idea

**Challenge.** 在现实情况下，有时数据量( $n$ )很少但数据的维度( $d$ )很大，即  $n \ll d$ ，此时使用模型例如最小二乘拟合这些数据得到的模型有很多可能的解。（未知数数量>方程数）

**Idea.** 类似于 GMM，此时可以假设存在 **latent variable**，其本身结构简单、便于估计，但能够刻画数据的生成机制（从而解释观测数据的结构）。

**Example 1: Fit Gaussian.** 对于高斯分布  $x \sim \mathcal{N}(\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1} (x - \mu)\right)$ ,  $\mu \in \mathbb{R}^d$ ,  $\Sigma \in \mathbb{R}^{d \times d}$ , 此时已有数据  $x^{(1)}, \dots, x^{(n)}$ , 则参数估计为：

$$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n x^{(i)} \quad (\in \mathbb{R}^d), \quad \hat{\Sigma} = \frac{1}{n} \sum_{i=1}^n (x^{(i)} - \hat{\mu}) (x^{(i)} - \hat{\mu})^\top \quad (\in \mathbb{R}^{d \times d}). \quad (92)$$

但是由于  $\text{rank}(\Sigma) < \{n, d\}$ ,  $n \ll d$ , 因此  $\text{rank}(\Sigma) < n$ , 因此  $\Sigma$  并非满秩, 这就导致 ①  $|\Sigma|^{1/2} = 0$ ; ②  $\Sigma^{-1}$  不存在. (MLE 细节详见附录Q)

#### Factor Analysis: Examples

**Example 1.** 假设此时的高斯分布为独立(independent) 且各向同性(identical), 即:

$$\begin{aligned} &(\text{independent}): x^{(1)}, \dots, x^{(n)} \stackrel{i.i.d.}{\sim} \mathcal{N}(\mu, \Sigma) \\ &(\text{identical}): \Sigma = \sigma^2 I, \quad \sigma^2 \in \mathbb{R}_+ \text{ (scalar)}, \end{aligned} \quad (93)$$

那么此时由于各向同性, 此高斯分布的等高线为圆形(如图29a)并且此时行列式  $|\Sigma| = |\sigma^2 I| = \sigma^{2d}$ . 此时 MLE 如下:

$$\begin{aligned} \max_{\mu, \sigma^2} \ell(\mu, \sigma^2) &= \max_{\mu, \sigma^2} \sum_{i=1}^n \left[ -\frac{1}{2\sigma^2} (x^{(i)} - \mu)^\top (x^{(i)} - \mu) - \frac{d}{2} \log \sigma^2 \right] \\ &\Rightarrow \min_{\mu, \sigma^2} \sigma^{-2} \sum_{i=1}^n \|x^{(i)} - \mu\|^2 + nd \log \sigma^2 \Rightarrow \hat{\mu} = \frac{1}{n} \sum_{i=1}^n x^{(i)}, \hat{\sigma}^2 = \frac{1}{nd} \sum_{i=1}^n \|x^{(i)} - \hat{\mu}\|^2 \end{aligned} \quad (94)$$

**Example 2.** 假设此时的高斯分布独立且各维独立(协方差矩阵为对角阵), 即:

$$\begin{aligned} & \text{(independent): } x^{(1)}, \dots, x^{(n)} \stackrel{i.i.d.}{\sim} \mathcal{N}(\mu, \Sigma) \\ & \text{(diagonal): } \Sigma = \text{diag}(\sigma_1^2, \sigma_2^2, \dots, \sigma_d^2), \end{aligned} \quad (95)$$

由于各维独立, 此高斯分布的等高线为轴对齐的椭圆(如图29b). 记  $z_j = \sigma_j^2$ , 此时 MLE:

$$\begin{aligned} \max_{\mu, \{z_j\}} \ell(\mu, \{z_j\}) &= -\frac{1}{2} \sum_{i=1}^n \sum_{j=1}^d \left[ z_j^{-1} (x_j^{(i)} - \mu_j)^2 + \log z_j \right] \\ &\Rightarrow \min_{z_1, \dots, z_d} \sum_{i=1}^n \sum_{j=1}^d \left( z_j^{-1} (x_j^{(i)} - \mu_j)^2 + \log z_j \right). \end{aligned} \quad (96)$$

由于目标函数在各维度独立, 因此此时只需要对于各个维度  $j$  作 MLE:

$$\begin{aligned} \min_{z_j} \sum_{i=1}^n \left( z_j^{-1} (x_j^{(i)} - \mu_j)^2 + \log z_j \right) &\Rightarrow -z_j^{-2} \sum_{i=1}^n (x_j^{(i)} - \mu_j)^2 + \frac{n}{z_j} = 0 \\ \Rightarrow \hat{\sigma}_j^2 &= z_j = \frac{1}{n} \sum_{i=1}^n (x_j^{(i)} - \mu_j)^2. \end{aligned} \quad (97)$$



(a) Example 1



(b) Example 2

Figure 29: Examples

### Factor Analysis: Factor Model

**Parameter.** 在因子模型(factor model) 中, 共有  $\mu \in \mathbb{R}^d, \Lambda \in \mathbb{R}^{d \times s}, \Phi \in \mathbb{R}^{d \times d}$  三个参数, 其中  $\mu$  为均值向量,  $\Lambda$  为因子载荷矩阵,  $\Phi$  为噪声协方差矩阵.

**Model.** 与之前的建模相同, 此时的生成模型仍然包含 latent variable  $z$ , 即为

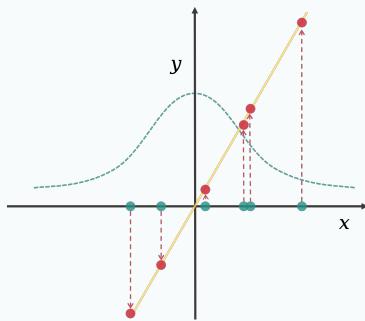
$$p(x, z) = p(x | z) p(z), \quad (98)$$

其中  $z$  为 latent variable, 且  $z \sim \mathcal{N}(0, I) \in \mathbb{R}^s$ ,  $s < d$ . 此时条件分布  $x | z$  被建模为  $x | z \sim \mathcal{N}(\mu + \Lambda z, \Phi)$ , 实际计算方式即:

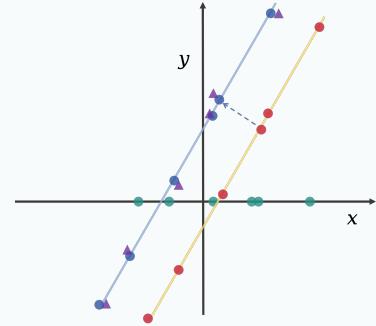
$$x = \mu + \Lambda z + \epsilon, \epsilon \sim \mathcal{N}(0, \Phi). \quad (99)$$

**Example.** 以  $d = 2, s = 1, n = 5$  为例, 最终的模型输出为  $x = \mu + \Lambda z + \epsilon$ :

1. Step 1. 隐变量生成:  $z^{(1)}, \dots, z^{(n)} \stackrel{i.i.d.}{\sim} \mathcal{N}(0, 1) \rightarrow$  对应图30a 中绿色点.
2. Step 2. 载荷矩阵 (假设) :  $\Lambda = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \rightarrow$  对应图30a 中将点映射至直线  $y = 2x$  上.
3. Step 3. 加入偏移量  $\mu$ :  $\tilde{x} = \mu + \Lambda z \rightarrow$  对应图30b 中将点移动至蓝色直线上.
4. Step 4. 加入噪声  $\epsilon$ :  $x = \mu + \Lambda z + \epsilon \rightarrow$  对应图30b 中将点进行随机扰动至紫色点.



(a) Example for Factor Model: Step 1-2



(b) Example for Factor Model: Step 3-4

Figure 30: Example for Factor Model

**Fact for Gaussian.** 记  $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \in \mathbb{R}^d$ , 其中  $x_1 \in \mathbb{R}^{d_1}, x_2 \in \mathbb{R}^{d_2}, d = d_1 + d_2$ , 协方差矩阵写为分块矩阵  $\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$ ,  $\Sigma_{ij} \in \mathbb{R}^{d_i \times d_j}, i, j \in \{1, 2\}$ . 回顾 Gaussian 分布的两个基本事实:

1. **Marginalization.** 边缘分布  $p(x_1) = \int p(x_1, x_2) dx_2$ , Gaussian 有  $p(x_1) \sim \mathcal{N}(\mu_1, \Sigma_{11})$
2. **Conditioning.**  $p(x_1 | x_2) \sim \mathcal{N}(\mu_{1|2}, \Sigma_{1|2})$ , 其中  $\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$ ,  $\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$ .

**Note 21.**  $\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$  推导可由矩阵求逆引理 (Matrix Inversion Lemma) 得到.

根据上述事实, 对于 factor analysis  $x = \mu + \Lambda z + \epsilon, \epsilon \sim \mathcal{N}(0, \Phi)$ , 此时由于  $\mathbb{E}[z] = 0, \mathbb{E}[x] = \mu$ , 因此联合分布  $\begin{pmatrix} z \\ x \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ \mu \end{pmatrix}, \Sigma\right)$ . 对于协方差矩阵  $\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$ :

$$\begin{aligned}
 \Sigma_{11} &= \text{Cov}(z) = \mathbb{E}[zz^\top] = I \\
 \Sigma_{12} &= \text{Cov}(z, x) = \mathbb{E}\left[z(x - \mu)^\top\right] = \mathbb{E}\left[z(\Lambda z + \epsilon)^\top\right] = \mathbb{E}[zz^\top]\Lambda^\top + \mathbb{E}[z\epsilon^\top] = I\Lambda^\top + 0 = \Lambda^\top \\
 \Sigma_{21} &= \Sigma_{12}^\top = \Lambda \\
 \Sigma_{22} &= \mathbb{E}\left[(x - \mu)(x - \mu)^\top\right] = \mathbb{E}\left[(\Lambda z + \epsilon)(\Lambda z + \epsilon)^\top\right] = \Lambda\mathbb{E}[zz^\top]\Lambda^\top + \mathbb{E}[\epsilon\epsilon^\top] = \Lambda\Lambda^\top + \Phi. \tag{100}
 \end{aligned}$$

因此  $\Sigma = \begin{pmatrix} I & \Lambda^\top \\ \Lambda & \Lambda\Lambda^\top + \Phi \end{pmatrix}$ .

### Factor Analysis: Conclusion

对于 factor analysis 模型, 已经假设存在 latent variable  $z \sim \mathcal{N}(0, I)$ , 并有  $\mu, \Lambda, \Phi$  三个参数需要估计, 此时  $x = \mu + \Lambda z + \epsilon, \epsilon \sim \mathcal{N}(0, \Phi)$ , 将其写为联合分布  $\begin{pmatrix} z \\ x \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ \mu \end{pmatrix}, \Sigma\right)$ , 使用 EM 算法:

1. **E-step.**  $Q_i(z) = p(z^{(i)} | x^{(i)}, \theta)$
2. **M-step.** 使用闭式解  $\Sigma = \begin{pmatrix} I & \Lambda^\top \\ \Lambda & \Lambda\Lambda^\top + \Phi \end{pmatrix}$  更新参数.

## 14.3 Principle Component Analysis (PCA)

### PCA: Introduction

无监督学习的分类可以根据是否是概率模型, 也可以根据是“聚类”型还是“子空间”型进行区分, 直至目前所接触到的无监督学习方法区分如下:

Structure	Probabilistic Model	Non-probabilistic Method
Clustering	GMM	K-means
Subspace	Factor Analysis	PCA

**PCA example.** 给定一批水果, 记录其体积( $x$ -轴)、质量( $y$ -轴)两种数据, 我们的目的 是区分其品种, 但是仅根据体积大小或质量并不能显著区分. 此时可以使用  $y = x$  方向作为区分标准, 并将其解释为“密度”, 这样将所有的点投影至  $y = x$  上可以将不同品种很好地区分. 在这个例子中,  $y = x$  实际上就是对数据变化(variation)解释性更强的方向.

**Idea.** PCA 的基本思想是寻找主成分方向(principle component), 使得数据在这些方向上能够更容易区分开 (例如使数据点投影后更分散开), 即更能够刻画数据的变化(variation).

### PCA

**Pre-processing.** 给定数据  $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$ , 首先需要对数据进行预处理:

1. **Center the data.** 首先中心化数据:

$$\mu = \frac{1}{n} \sum_{i=1}^n x^{(i)}, x^{(i)} \leftarrow x^{(i)} - \mu. \quad (101)$$

2. **Rescaling.** 再进行特征缩放以归一化消除量纲:

$$x_j^{(i)} \leftarrow x_j^{(i)} / \sigma_j. \quad (102)$$

**Note 22.** 中心化使 PCA 关注方差而非偏移, 如果不中心化 PCA 会浪费一个主成分去解释均值. 由于  $Var(kx) = k^2 Var(x)$ , 因此不归一化数值大的特征会主导主成分, 因此为更关注方向上的方差而非绝对大小需要进行归一化.

**PCA as Optimization Problem.** 当只有一个主成分方向(principle component)  $u_1 \in \mathbb{R}^d$  后, 需要在子空间  $\mathcal{S}_1 = \{tu_1 : t \in \mathbb{R}\}$  中寻找距离数据  $x$  最近的点, 即投影点:

$$\begin{aligned}\alpha_1 &= \arg \min_{\alpha} \|x - \alpha u_1\|^2 = \|x\|^2 + \alpha^2 \|u_1\|^2 - 2\alpha \langle u_1, x \rangle \\ \Rightarrow \frac{\partial}{\partial \alpha} (\|x\|^2 + \alpha^2 - 2\alpha \langle u_1, x \rangle) &= 2\alpha - 2\langle u_1, x \rangle = 0 \Rightarrow \alpha = \langle u_1, x \rangle.\end{aligned}\quad (103)$$

因此数据可以被重新表示为  $\alpha u_1 = \langle u_1, x \rangle u_1$ ,  $\alpha$  就是数据在新坐标轴  $u_1$  上的坐标.

类似地, 已有  $k$  个方向  $u_1, \dots, u_k \in \mathbb{R}^d$ , 且满足  $\|u_i\| = 1, u_i^\top u_j = \delta_{ij}$ , 此时:

$$\arg \min_{\alpha_1, \dots, \alpha_k} \|x - \sum_{i=1}^k \alpha_i u_i\|^2 \Rightarrow \alpha_i = \langle u_i, x \rangle. \quad (104)$$

PCA 可以从两个完全等价的角度来理解:

1. **Maximizing Projected Subspace.** 最大化投影: 寻找一个单位向量 (或子空间), 使数据投影后的方差最大.
2. **Minimizing Residual.** 最小化残差: 寻找一个低维子空间, 使数据点到该子空间的重构误差最小.

---

from August 26, 2025 to August 28, 2025

Last update: December 19, 2025

## 15 Lecture 15: PCA / ICA

Link on YouTube: Stanford CS229 Machine Learning, PCA / ICA, 2022, Lecture 15

### 15.1 Introduction

#### Introduction

本节内容包括 PCA 和 ICA 两部分内容. 在 PCA 中通过 PCA 的两个等价目标以线代视角讲解 PCA 的算法原理. 在 ICA 中通过简单的案例讲解 ICA 的问题设定和目标.

### 15.2 Principle Component Analysis

#### Principle Component Analysis: Recall

**Find the closet point to the line.** 在给定了单位方向  $u$  之后, 寻找点  $x$  在  $u$  方向的坐标就变成了寻找到  $u$  张成的直线的最近点, 即:

$$\alpha^* = \arg \min_{\alpha} \|x - \alpha u\|^2. \quad (105)$$

由于  $\|x - \alpha u\|^2 = \langle x - \alpha u, x - \alpha u \rangle = \|x\|^2 + \alpha^2 \|u\|^2 - 2\alpha \langle x, u \rangle$ , 因此

$$\|x - \alpha u\|^2 \|u\|^2 = 1\|x\|^2 + \alpha^2 - 2\alpha \langle x, u \rangle \Rightarrow \frac{d}{d\alpha} (\|x\|^2 + \alpha^2 - 2\alpha \langle x, u \rangle) = 0 \Rightarrow \alpha^* = \langle x, u \rangle. \quad (106)$$

**General to  $K$  components.** 若已得到了  $K$  个单位方向  $u_1, \dots, u_k \in \mathbb{R}^d$ ,  $x \in \mathbb{R}^d$ , 则式(105)拓展为:

$$(\alpha_1^*, \dots, \alpha_K^*) = \arg \min_{\alpha_1, \dots, \alpha_d} \left\| x - \sum_{k=1}^d \alpha_k u_k \right\|^2 \Rightarrow \alpha_k^* = \langle x, u_k \rangle, \quad (107)$$

其中  $\left\| x - \sum_{k=1}^d \alpha_k^* u_k \right\|^2$  被称为残差(Residual), 其表示  $x$  中无法被子空间表示的部分.

**Goal.** PCA 的目标可以通过两种等价的方式来定义:

- 最大化投影到子空间上的方差:  $\max_{u \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (u^\top x^{(i)})^2$  s.t.  $\|u\|_2 = 1$ .
- 最小化重构残差 (Residual) :  $\min_{\alpha} \|x - \alpha u\|^2$ .

#### Principle Component Analysis: Linear Algebra Review

**Linear Algebra Review.** ① 设  $A \in \mathbb{R}^{d \times d}$  为对称矩阵(symmetric), 则有如下分解:

$$A = U \Lambda U^\top, \quad (108)$$

其中  $U$  为正交矩阵(orthogonal), 即  $U U^\top = I$ ;  $\Lambda$  为对角阵(diagonal), 且  $\Lambda_{ii} = \lambda_i$ .<sup>a</sup>

② 设  $x = \sum_{k=1}^n \alpha_k u_k$ , 其中  $U = [u_1, \dots, u_n]$ , 则:

$$Ax = U \Lambda U^\top x = U \Lambda \sum_{k=1}^n \alpha_k e_k = U \sum_{k=1}^n \lambda_k \alpha_k e_k = \sum_{k=1}^n \lambda_k \alpha_k u_k, \quad (109)$$

因此

$$x^\top Ax = \left(\sum_{k=1}^n \alpha_k u_k\right)^\top \left(\sum_{k=1}^n \lambda_k \alpha_k u_k\right) = \sum_{k=1}^n \lambda_k \alpha_k^2. \quad (110)$$

从而可得:

$$\max_{\|x\|^2=1} x^\top Ax \iff \max_{\|\alpha\|^2=\sum_k \alpha_k^2=1} \sum_{k=1}^n \lambda_k \alpha_k^2 \implies \alpha_1 = 1, \quad \alpha_{k \neq 1} = 0. \quad (111)$$

**Note 23.** 若  $\lambda_1 = \lambda_2$ , 则任意选择  $\lambda_1, \lambda_2$  使得  $\lambda_1^2 + \lambda_2^2 = 1$  即可.

<sup>a</sup>即为特征值, 其中  $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_d$ . 能够排列是因为对角阵的特征值均为实数因此可以比较大小.

### Principle Component Analysis: Eigenvalue

在上述线性代数基础上, 对于 PCA 的目标可以进行如下改写:

$$\max_{u \in \mathbb{R}^d, \|u\|_2=1} \frac{1}{n} \sum_{i=1}^n (u^\top x^{(i)})^2 (u^\top x^{(i)})^2 = u^\top x^{(i)} x^{(i)\top} u u^\top \left( \frac{1}{n} \sum_{i=1}^n x^{(i)} x^{(i)\top} \right) u, \quad (112)$$

其中由于数据已经过中心化, 因此  $\Sigma = \frac{1}{n} \sum_{i=1}^n x^{(i)} x^{(i)\top}$  即为协方差矩阵, 故问题转化为:

$$\max_{\|u\|^2=1} u^\top \Sigma u. \quad (113)$$

根据式(111)结论可知每一次只需要选取最大的特征值即可, 因此原始数据  $x^{(i)}$  表示为:

$$x^{(i)} \mapsto \sum_{j=1}^k (x^{(i)\top} u_j) u_j, \quad \mathbb{R}^d \rightarrow \mathbb{R}^k. \quad (114)$$

$k$  的选取一般是使得累计解释方差  $\geq 90\%$ , 即选取累计占比  $\geq 90\%$  的特征值:

$$\frac{\sum_{i=1}^k \lambda_i}{\sum_{i=1}^d \lambda_i} \geq 0.9. \quad (115)$$

**Note 24.** PCA 仍然是使用与原先坐标一样的  $\mathbb{R}^d$  维基向量, 但不同的是原来  $\mathbb{R}^d$  维空间有  $d$  个基向量、需要  $d$  个坐标, 例如三维空间中  $(1, 2, 1)$  才能表示一个点, 但 PCA 重新选取了  $k \ll d$  个基向量, 只需要  $k$  个系数就可以将该坐标表示, 因此数据的表示  $\mathbb{R}^d \rightarrow \mathbb{R}^k$ .

**Note 25.** 式(115)中分母虽然包含了所有的特征值, 但是并不需要计算, 因为只需要计算  $\text{trace}(\Sigma)$  即可, 因此还是只需要计算  $k$  个特征值. 如果有相同的特征值, 那么可能每次进行 PCA 得到的主成分不相同.

### 15.3 Independent Component Analysis (ICA)

#### Independent Component Analysis: Motivation

假设在一个房间中有两个说话者(speaker)  $S_1, S_2$  和两个麦克风(microphone), 他们的位置均固定(如图31a). 说话者可以发出 2 个独立信号  $s_1^{(t)}, s_2^{(t)}$  (如图31b), 麦克风可以观测

到 2 个混合信号  $x_1^{(t)}, x_2^{(t)}$ . 假设仅观测数据已知  $x_1^{(t)}, x_2^{(t)}$ , 且其为  $s_1^{(t)}, s_2^{(t)}$  的线性组合:

$$x_j(t) = a_{j1}s_1^{(t)} + a_{j2}s_2^{(t)}, j = \{1, 2\} \Rightarrow x(t) = As(t) \quad (116)$$

其中  $A, s(t)$  均视为 latent (未知). 我们的目的是想通过观测恢复背后“相互独立的源信号”.

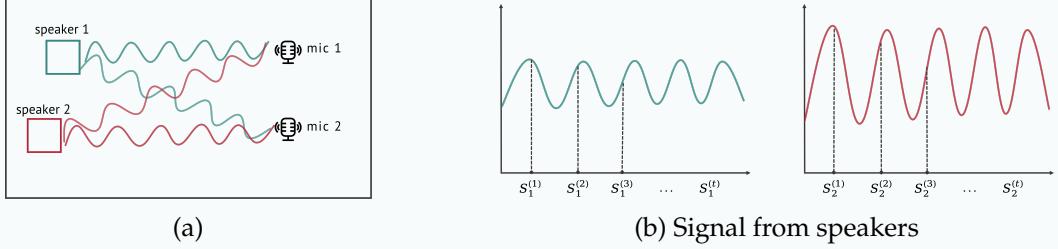


Figure 31: Example for ICA

下面将上述例子拓展至一般形式.

**Given:** 观测数据  $x^{(1)}, \dots, x^{(n)} \in \mathbb{R}^d$ ,  $d$  为麦克风数 = 说话源数.

**Do:** 找到  $s^{(1)}, \dots, s^{(n)} \in \mathbb{R}^d$  及  $A \in \mathbb{R}^{d \times d}$ , 使得  $x^{(t)} = As^{(t)}$ , 其中称  $A$  为混合矩阵(mixing matrix), 称  $W = A^{-1}$  为 unmixing matrix.

如果已知  $A$  或  $W$ , 那么  $s^{(t)} = Wx^{(t)}$ , 写成行向量为

$$W = \begin{bmatrix} w_1^\top \\ \vdots \\ w_d^\top \end{bmatrix} \Rightarrow s_j^{(t)} = w_j^\top x^{(t)}. \quad (117)$$

**Caveats.** 需要注意的是:

1. 假设源信号相互独立, 且混合矩阵  $A$  不随时间变化.
2. ICA 存在内在的不确定性(intrinsic ambiguity). 源信号在如下方面不可区分:
  - 顺序(permuation): 例如 speaker 1 与 speaker 2 可交换, 不能确定顺序
  - 尺度/强度=scaling): 由于  $(CA)(C^{-1}s^{(t)}) = As^{(t)}$ , 因此对于已观测到的  $As^{(t)}$ , 任意改变  $A$  都可以间接改变源信息  $s^{(t)}$ , 因此只能相对区分强度.
3. 源信号不能为高斯分布: 若源信号为高斯分布  $s \sim \mathcal{N}(\mu_s, I)$ , 则

$$x^{(i)} \sim \mathcal{N}(\mu_x, AA^\top) \Rightarrow \text{若 } U^\top U = I, AU \text{ 生成与之相同的观测分布} \quad (118)$$

**Note 26.** 若源信号为高斯分布  $s \sim \mathcal{N}(\mu_s, I)$ , 则  $x = As \sim \mathcal{N}(\mu_x, AA^\top)$ . 由于  $\mathcal{N}(\mu_x, AA^\top)$  的 covariance matrix  $AA^\top$  是对称的, 因此其具有旋转不变性(rotation invariance). 取正交矩阵  $UU^\top = I$ , 定义新源  $s' = Us$ , 由于高斯分布在正交变换下不变因此  $s' \sim \mathcal{N}(\mu_s, I)$ , 故  $x = As = A(U^\top U)s = (AU^\top)\tilde{s}$ , 即新源可以生成一样的观测分布, 从而根本无法区分.

### Independent Component Analysis: Algorithm

**Review: Density under linear transform.** 对于均匀分布  $s \sim U([0, 1])$ ,  $p_s(x) = \begin{cases} 1, & [0, 1] \\ 0, & \text{o.w.} \end{cases}$

若  $u = 2s$ , 则归一化常数必须改变:  $p_u(x) = p_s\left(\frac{x}{2}\right) \cdot \frac{1}{2}$ . 对于一般情形  $x = As$ :

$$p_x(x) = p_s(A^{-1}x) \cdot |\det(A^{-1})| \xrightarrow{W=A^{-1}} p_x(x) = p_s(Wx) \cdot |\det(W)| \quad (119)$$

**ICA is MLE.** 由于源信号相互独立, 因此  $p(s) = \prod_{j=1}^d p_s(s_j)$ , 其中  $d$  为源信号数, 故

$$x = As, W = A^{-1}, s = Wx \Rightarrow p(x) = \prod_{j=1}^d p_s(w_j^\top x) \cdot |\det(W)|. \quad (120)$$

**Key technical trick.** 在计算概率时使用 likelihood function  $g$  进行替代:  $p_s(s_j) \propto g'(s_j)$ , 其中  $g(x) = (1 + e^{-x})^{-1}$ , 此时 Log-likelihood:

$$\ell(W) = \sum_{t=1}^n \left[ \sum_{j=1}^d \log g'(w_j^\top x^{(t)}) + \log |\det(W)| \right]. \quad (121)$$

**Note 27.** 事实上此处的 likelihood function  $g$  只需要是非 *relationally invariant* 即可. 此处  $g(x) = (1 + e^{-x})^{-1}$  为 sigmoid, 使用的是其导数  $g'$ , 即 logistic density.

---

from December 21, 2025 to December 22, 2025

Last update: December 22, 2025

## 16 Lecture 16: Self-supervised learning

Link on YouTube: Stanford CS229 Machine Learning, Self-supervised learning, 2022, Lecture 16

### 16.1 Introduction

#### Introduction

本节主要讨论在标签稀缺但无标签数据极其丰富的现实条件下，如何通过自监督方式学习高质量表示，并将其迁移到下游任务中，重点包括：1. Self-supervised Learning 的基本框架，2. Adaptation 的两种典型方式，3. Contrastive Learning (对比学习)，4. 大语言模型 (LLM)。

### 16.2 Self-supervised Learning

#### Basic Concept

网络架构、数据、硬件是深度学习训练的三大组成部分。其中对于训练数据而言，在现实场景中有标签数据相对无标签数据是非常少的，为使模型“见过”更多数据，一个想法是先在大量无标签数据上进行无监督学习再针对特定任务使用有标签数据进行监督学习。这种现代化训练方法被称为 **Self-supervised learning**。

**Pre-training.** 预训练(pre-train) 是指在大量的无标签数据上进行无监督学习以训练一个大型模型。

**Adaptation.** 将预训练模型应用于各种特定下游任务(downstream task)，通常使用监督学习进一步微调(tuning)。

**Note 28.** *Adaptation* 也被称为 *transfer learning*. 现代的 *trasfer learning (TL)* 使用无标签数据进行预训练，但 2000 年左右 *TL* 还是使用监督学习进行预训练，并且上下游任务相似。

**Foundation model.** 也被称为 pre-trained model [15].

#### Pretraining

**Data.**  $\{x^{(1)}, \dots, x^{(n)}\}$ , 此处为无标签数据。

**Model.**  $\phi_\theta : x \mapsto \phi_\theta(x) \in \mathbb{R}^m$ , 将数据 (可能是文本、图像等) 映射为向量。

**Note 29.**  $\phi_\theta$  可以被称为 *representation / feature / embedding*, 在 *kernel method* 中被称为 *feature map*, 但区别是 *kernel method* 中  $\phi$  是给定的, 此处的  $\phi_\theta$  是一个神经网络,  $\theta$  需要学习。

**Loss.**  $L_{\text{pre}}(\theta) = \frac{1}{n} \sum_{i=1}^n \ell_{\text{pre}}(\theta, x^{(i)})$ . 对于不同的任务  $\ell_{\text{pre}}(\theta, x^{(i)})$  的定义也不相同。

**Optimize.**  $\hat{\theta} = \arg \min_\theta L_{\text{pre}}(\theta)$ , 并将  $\hat{\theta}$  称为 pretrained model.

#### Adaptation

**Adaptation.** 作用于下游(监督学习)任务数据集  $\{(x_{\text{task}}^{(1)}, y_{\text{task}}^{(1)}), \dots, (x_{\text{task}}^{(n_t)}, y_{\text{task}}^{(n_t)})\}$ , 其中  $n_t = \# \text{downstream task examples}$ . 当  $n_t = 0$  时被称为 **zero-shot learning**; 当  $n_t \neq 0$  但仍较小(例如 10) 时被称为 **few-shot learning**.

**Linear probe.** Adaptation 的最简单的方式是冻结预训练模型, 只在其输出表示上训练一个线性模型, 即 linear probe:

$$\text{Prediction model: } \hat{y} = w^\top \phi_{\hat{\theta}}(x), w \in \mathbb{R}^m \Rightarrow \min_w \frac{1}{n_t} \sum_{i=1}^{n_t} \ell_{\text{task}} \left( y_{\text{task}}^{(i)}, w^\top \phi_{\hat{\theta}}(x_{\text{task}}^{(i)}) \right). \quad (122)$$

以计算机视觉(Computer Vision) 任务为例, 在有标签数据集(例如 ImageNet) 上学习一个神经网络  $u^\top \phi_{\theta}(x)$ , 其中  $u$  为最后一层线性层,  $\phi_{\theta}$  为最后一层前所有层, 此时  $\phi_{\theta}$  即作为 pretrained model.

**Finetune.** Adaptation 的另一种方法是微调(finetune) pretrained model 的参数  $\hat{\theta}$ :

$$\begin{aligned} \text{Prediction model: } & \hat{y} = w^\top \phi_{\theta}(x) \\ \text{Optimize: } & \text{both } w \text{ and } \theta \text{ on downstream task} \\ \text{Initialization: } & \theta \leftarrow \hat{\theta} \text{ (from pretraining)}, w \leftarrow w_0 \text{ (random)} \end{aligned} \quad (123)$$

### 16.3 Contrastive Learning

#### Data Augmentation

数据增强 (data augmentation) 是通过特定方式使用原始图像生成不同“视角/视图 (views)”并将其作为新的数据用于模型训练. 常见的数据增强方式包括:

1. 平移(translation), 翻转(flip), 旋转(rotation), 缩放(scale)
2. 增加噪点(add pixel noise), 改变颜色(color)、亮度(brightness)、对比度(contrast)

#### Contrastive Learning

数据增强在监督学习中可以增大数据集并且让模型对于这些改变具有不变性, 其也可以应用于无监督学习中. 设原始样本  $x$  经过两次随机数据增强后得到两视图  $\hat{x}, \tilde{x}$ , 称  $(\hat{x}, \tilde{x})$  为正样本 (positive pair). 另一随机样本  $z$  经数据增强后得到视图  $\hat{z}$ , 称  $(\hat{x}, \hat{z})$  为负样本 (negative / random pair). 那么在对比学习的目标在于:

1. 让正样本对在表征空间里更接近, 即  $\phi_{\theta}(\hat{x})$  与  $\phi_{\theta}(\tilde{x})$  接近.
2. 让负样本对在表征空间里更远离, 即  $\phi_{\theta}(\hat{x})$  与  $\phi_{\theta}(\hat{z})$  远离. (防止坍缩 (collapse)<sup>a</sup>)

**SIMCLR.** Contrastive learning 的代表性开创工作是 SIMCLR [16]. 其想法是取一个 batch 的原始样本  $x^{(1)}, x^{(2)}, \dots, x^{(B)}$ , 对每个样本做两次独立的数据增强得到  $\hat{x}^{(1)}, \dots, \hat{x}^{(B)}$  和  $\tilde{x}^{(1)}, \dots, \tilde{x}^{(B)}$ . 定义相似度分数:

$$s_{i,j} \triangleq \phi_{\theta}(\hat{x}^{(i)})^\top \phi_{\theta}(\tilde{x}^{(j)}), \quad (124)$$

则第  $i$  个样本的对比损失定义为

$$\ell_i = -\log \frac{\exp(s_{i,i})}{\exp(s_{i,i}) + \sum_{j \neq i} \exp(s_{i,j})}, \quad (125)$$

总损失为

$$\mathcal{L} = \sum_{i=1}^B \ell_i = - \sum_{i=1}^B \log \frac{\exp(\phi_\theta(\hat{x}^{(i)})^\top \phi_\theta(\tilde{x}^{(i)}))}{\exp(\phi_\theta(\hat{x}^{(i)})^\top \phi_\theta(\tilde{x}^{(i)})) + \sum_{j \neq i} \exp(\phi_\theta(\hat{x}^{(i)})^\top \phi_\theta(\tilde{x}^{(j)}))}. \quad (126)$$

可以看出增大正样本相似度 ( $s_{i,i}$ ) 会让损失变小, 增大负样本相似度 ( $s_{i,j}, i \neq j$ ) 会让损失变大. 这与 contrastive learning 的两个目标相符.

**Note 30.** 此处的损失函数式(126) 并未使用标签信息, 因此其为 *unsupervised learning / self-supervised learning*.

<sup>a</sup>当模型坍缩至一点时  $\phi(\hat{x})$  与  $\phi(\tilde{x})$  距离为 0, 即条件 1 满足, 因此为保证模型不坍缩还需要满足第 2 点.

## 16.4 Large Language Model

### Large Language Model

语言模型 (language model) 通常处理文本序列, 记为  $(x_1, \dots, x_T), x_i \in \{1, \dots, V\}$ , 其中  $T$  为序列长度,  $\{1, \dots, V\}$  为词汇表 (vocabulary).

**Note 31.** 语言模型处理文本的细度 (granularity) 并非是单词, 而是 *token*, 即  $x_i$  并不一定是一个单词, 其也有可能是单词的一部分. 例如一些频繁出现的单词其本身可以作为一个 *token*, 但对于如 *suboptimal* 这样的单词可能会被拆为 *sub* + *optimal* 作为两个 *token*.

**Language model.** 语言模型是对序列联合分布的概率模型  $p(x_1, \dots, x_T)$ , 由于每个 *token* 有  $V$  种可能性, 因此共有  $V^T$  种可能, 即该分布的支持集大小为  $V^T$ . 为解决  $V^T$  过大的问题, 引入 链式法则分解 (chain rule):

$$p(x_1, \dots, x_T) = p(x_1) p(x_2 | x_1) p(x_3 | x_1, x_2) \cdots p(x_T | x_1, \dots, x_{T-1}) = \prod_{t=1}^T p(x_t | x_1, \dots, x_{t-1}), \quad (127)$$

因此语言建模可转化为学习一系列条件概率 (下一 *token* 分布) :

$$(x_t | x_1, \dots, x_{t-1}), \quad t = 1, \dots, T. \quad (128)$$

**Embedding.** 由于机器无法直接理解自然语言, 因此需要将 *token*  $i$  转化为向量  $e_i \in \mathbb{R}^{d_1}$ , 即嵌入 (embedding). 再经过黑盒模型  $\phi_\theta$  (一般为 Transformer) 处理后得到  $c_{i+1} \in \mathbb{R}^{d_2}$  (如图32). 在使用  $c_t$  预测  $p(x_t | x_1, \dots, x_{t-1})$  时可以将其建模为多分类任务:

$$\begin{bmatrix} p(x_t = 1 | x_1, \dots, x_{t-1}) \\ \vdots \\ p(x_t = V | x_1, \dots, x_{t-1}) \end{bmatrix} = \text{softmax}(z_t) = \text{softmax}(W_t c_t) \in \mathbb{R}^V \quad (129)$$

其中  $z_t = W_t c_t$  被称为 logits,  $W_t \in \mathbb{R}^{V \times d_2}$ ,  $\text{softmax}(u) = \left[ \frac{\exp(u_1)}{\sum_{i=1}^V \exp(u_i)}, \dots, \frac{\exp(u_V)}{\sum_{i=1}^V \exp(u_i)} \right]^\top$ .

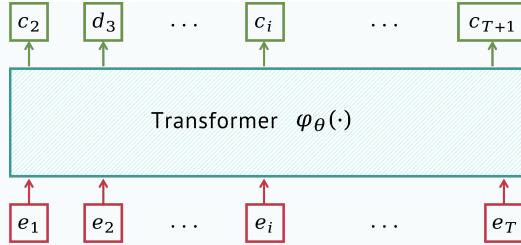


Figure 32: Large language model embedding.

**Loss.** 定义  $p_t = \text{softmax}(W_t c_t)$ , 则整体损失是对每个位置交叉熵损失的求和:

$$\text{loss}(W, \theta, e) = \sum_{t=1}^T (\text{cross-entropy loss at position } t) = \sum_{t=1}^T -\log p_t(x_t). \quad (130)$$

**Zero-shot.** 零样本学习充分利用 LLM 的 generalization 的能力, 其将任意任务转换为问题, 在让模型得到一个回答后通过模型的推理能力不断生成后续的 token, 从而完成整个 sequence 的生成。

**In-context learning.** 在上下文学习中, 任务的例子直接通过“拼接”的方式, 加入到一个大文档中 (即 prompt)。模型通过理解这些例子, 学习任务的规律, 并且基于此做出预测。例如给定以下几个问题和答案, 我们将它们拼接成一个prompt:

Q:  $2 \sim 3 = ?$  A: 5

Q:  $6 \sim 7 = ?$  A: 13

Q:  $15 \sim 2 = ?$  A: \_\_\_ (模型预测)

在这种情况下, 模型通过已经提供的少量样本 (“ $2 \sim 3 = 5$ ” 和 “ $6 \sim 7 = 13$ ”), 来推测和回答新的问题  $15 \sim 2 = ?$ 。

---

from December 2, 2025 to January 5, 2026

Last update: January 5, 2026

## A Generalization

### Appendix

泛化能力的假设前提是训练集中数据的分布与现实中的分布是相同的，例如训练集是几千张猫咪的照片，每张照片由 $1024 \times 1024$ 个像素点构成，所有像素点的数值会满足一个分布，我们希望这个分布与全世界所有的猫咪的照片的分布是一样的。但是，实际上这永远不可能达到，这个分布也许只有上帝才知道，因此这个假设永远是错误的，但是对于这样的一个模型，如果效果足够好当然是可以接受的，这也就是 George Box 1976年所说的名言[17]：

*All models are wrong, but some are useful.*

但是这个前提假设为我们提供了一些显然的指导，例如如果训练集全部是猫咪的照片，那么用在此训练集上训练好的模型去用于小狗的照片，那么结果可想而知会很糟糕。

## B Gaussian Distribution

### Gaussian Distribution

Gauss分布(Gaussian Distribution)是最常用的分布，其概率密度函数为(如图33):

$$P(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

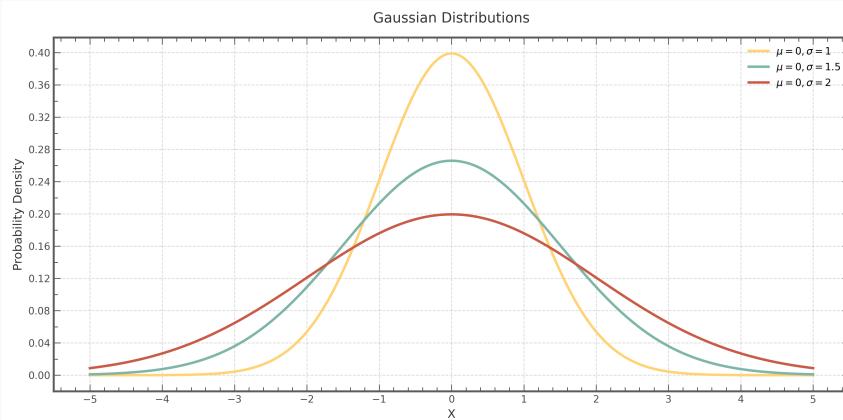


Figure 33: Gaussian distribution

## C Gradient of loss function

### Gaussian Distribution

$$J(\theta) \triangleq - \sum_{i=1}^n \left( y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)})) \right)$$

$$\frac{\partial}{\partial \theta} J(\theta) = \sum_{i=1}^n \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

*Proof.* 由于  $h(x) = \frac{1}{1+\exp(-x)}$ , 因此

$$h'(x) = \frac{\exp(-x)}{(1 + \exp(-x))^2} = h(x)(1 - h(x))$$

从而对于  $h_{\theta}(x^{(i)}) = \frac{1}{1+\exp(-\theta^\top x^{(i)})}$ , 其导数为

$$\frac{\partial h_{\theta}(x^{(i)})}{\partial \theta} = h_{\theta}(x^{(i)}) \left( 1 - h_{\theta}(x^{(i)}) \right) x^{(i)}.$$

将  $J(\theta)$  关于  $\theta$  的梯度表示为:

$$\frac{\partial J(\theta)}{\partial \theta} = - \sum_{i=1}^n \frac{\partial}{\partial \theta} \left[ y^{(i)} \log h_{\theta}(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right].$$

对于第一项  $y^{(i)} \log h_{\theta}(x^{(i)})$ :

$$\frac{\partial}{\partial \theta} \left[ y^{(i)} \log h_{\theta}(x^{(i)}) \right] = y^{(i)} \frac{1}{h_{\theta}(x^{(i)})} \cdot \frac{\partial h_{\theta}(x^{(i)})}{\partial \theta}.$$

对于第二项  $(1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))$ :

$$\frac{\partial}{\partial \theta} \left[ (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)})) \right] = (1 - y^{(i)}) \frac{-1}{1 - h_{\theta}(x^{(i)})} \cdot \frac{\partial h_{\theta}(x^{(i)})}{\partial \theta}.$$

将两部分合并:

$$\frac{\partial J(\theta)}{\partial \theta} = - \sum_{i=1}^n \left[ y^{(i)} \frac{1}{h_{\theta}(x^{(i)})} - (1 - y^{(i)}) \frac{1}{1 - h_{\theta}(x^{(i)})} \right] \cdot \frac{\partial h_{\theta}(x^{(i)})}{\partial \theta}.$$

注意到:

$$\frac{1}{h_{\theta}(x^{(i)})} - \frac{1}{1 - h_{\theta}(x^{(i)})} = \frac{h_{\theta}(x^{(i)}) - y^{(i)}}{h_{\theta}(x^{(i)})(1 - h_{\theta}(x^{(i)}))}.$$

再结合  $\frac{\partial h_{\theta}(x^{(i)})}{\partial \theta} = h_{\theta}(x^{(i)})(1 - h_{\theta}(x^{(i)}))x^{(i)}$ , 最终梯度为:

$$\frac{\partial}{\partial \theta} J(\theta) = \sum_{i=1}^n \left( h_{\theta}(x^{(i)}) - y^{(i)} \right) x^{(i)}$$

□

## D Python codes for plotting Gaussian distribution

```

1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # generate data from gaussian distribution
5 x = np.linspace(-5, 5, 500)

```

```

6 mu_sigma_pairs = [
7     (0, 1, colors["huang"]),
8     (0, 1.5, colors["lv1"]),
9     (0, 2, colors["hong"])
10]
11
12 # create a figure
13 fig, ax = plt.subplots(figsize=(12, 6), dpi=400)
14
15 # plot several lines
16 for mu, sigma, color in mu_sigma_pairs:
17     y = (1 / (np.sqrt(2 * np.pi) * sigma)) * np.exp(-0.5 * ((x - mu) / sigma) ** 2)
18     ax.plot(x, y, color=color, lw=2.5, label=f"$\mu={mu}, \sigma={sigma}$")
19
20 set_figure() # a function for figure plot setting
21 plt.savefig("gaussian_distributions.png", dpi = 400) # save the figure
22 plt.show()

```

## E Proof of Example 1

### Proof of Example 1

*Proof.* 对于一个Bernoulli distribution, 即0-1分布

$$\mathbb{P}(y; \phi) = \phi^y (1 - \phi)^{1-y}, \quad y \in \{0, 1\}$$

我们的目标是将其写为式(20)的形式, 推导过程如下:

$$\begin{aligned} \mathbb{P}(y; \phi) &= \phi^y (1 - \phi)^{1-y} = \exp \left[ \log \left( \phi^y (1 - \phi)^{1-y} \right) \right] \\ &= \exp [y \log \phi + (1 - y) \log(1 - \phi)] = \exp \left[ y \log \left( \frac{\phi}{1 - \phi} \right) + \log(1 - \phi) \right] \end{aligned}$$

因此令  $\eta = \log \left( \frac{\phi}{1 - \phi} \right)$ ,  $T(y) = y$ ,  $b(y) = 1$ ,  $a(\eta) = -\log(1 - \phi)$ , 由于

$$\exp \eta + 1 = \frac{1}{1 - \phi} = \exp a$$

因此  $a(\eta) = \log(e^\eta + 1)$ , 故  $\mathbb{P}(y; \phi)$  可以写为:

$$\mathbb{P}(y; \phi) = \exp [\eta y - \log(e^{\eta+1})]$$

□

同时计算可得

$$\begin{aligned} a'(\eta) &= \frac{e^\eta}{1 + e^\eta} = \frac{\phi}{1 - \phi} \times (1 - \phi) = \phi = \mathbb{E}[T(y)] \\ a''(\eta) &= \left( \frac{e^\eta}{1 + e^\eta} \right)' = \frac{e^\eta}{((1 + e^\eta)^2)} = \frac{\phi}{1 - \phi} \times (1 - \phi)^2 = \phi(1 - \phi) = \text{Var}(T(y)) \end{aligned}$$

## F Proof of Example 2

### Proof of Example 1

*Proof.* 对于一个高斯分布  $y \sim \mathcal{N}(\mu, \sigma^2)$ , 其中均值  $\mu$  需要预测, 方差  $\sigma^2$  固定

$$\mathbb{P}(y; \mu) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{(y-\mu)^2}{2\sigma^2}\right]$$

我们的目标是将其写为式(20)的形式, 推导过程如下:

$$\begin{aligned}\mathbb{P}(y; \mu) &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left[-\frac{y^2}{2\sigma^2} + \frac{y\mu}{\sigma^2} - \frac{\mu^2}{2\sigma^2}\right] \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{y^2}{2\sigma^2}\right) \exp\left[\mu \cdot \left(\frac{1}{\sigma^2}y\right) - \frac{1}{2\sigma^2}\mu^2\right]\end{aligned}$$

因此令  $\eta = \mu$ ,  $b(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{y^2}{2\sigma^2}\right)$ ,  $T(y) = \frac{1}{\sigma^2}y$ ,  $a(\eta) = \frac{1}{2\sigma^2}\mu^2 = \frac{1}{2\sigma^2}\eta^2$  即可。  $\square$

同样地计算可得

$$a'(\eta) = \frac{1}{\sigma^2}\mu = \mathbb{E}[T(y)], \quad a''(\eta) = \frac{1}{\sigma^2} = \text{Var}(T(y))$$

## G Proof of Theorem 1

### Proof of Theorem 1

*Proof.* 要证在  $\mathbb{P}(y; \eta) = b(y) \exp[\eta^T T(y) - a(\eta)]$  下  $a'(\eta) = \mathbb{E}[T(y)]$ ,  $a''(\eta) = \text{Var}(T(y))$ 。  
(20)式等价地可以写为

$$\mathbb{P}(y; \eta) = \frac{b(y) \exp[\eta^T T(y)]}{\exp a(\eta)} \quad (131)$$

由于概率和为1, 因此

$$\exp a(\eta) = \sum_y b(y) \exp[\eta^T T(y)] \quad (132)$$

两边同时对  $\eta$  求导, 得

$$a'(\eta) \exp a(\eta) = \sum_y b(y) T(y) \exp[\eta^T T(y)] \quad (133)$$

即

$$a'(\eta) = \sum_y T(y) \frac{b(y) \exp[\eta^T T(y)]}{\exp a(\eta)} = \sum_y T(y) \mathbb{P}(y; \eta) = \mathbb{E}[T(y)] \quad (134)$$

同理对式(133)两边再同时对  $\eta$  求导, 得

$$[a''(\eta) + (a'(\eta))^2] \exp a(\eta) = \sum_y b(y) (T(y))^2 \exp[\eta^T T(y)] \quad (135)$$

即

$$a''(\eta) + (a'(\eta))^2 = \sum_y (T(y))^2 \frac{b(y) \exp[\eta^T T(y)]}{\exp a(\eta)} = \sum_y (T(y))^2 \mathbb{P}(y; \eta) = \mathbb{E}[(T(y))^2] \quad (136)$$

因此

$$a''(\eta) = \mathbb{E}[(T(y))^2] - (\mathbb{E}[T(y)])^2 = \text{Var}(T(y)) \quad (137)$$

□

## H Multivariate Gaussian Distribution

### Multivariate Gaussian Distribution

高维高斯分布 (Multivariate Gaussian Distribution) 是高维空间中常见的概率分布。对于一个 $d$ -维的随机向量  $\mathbf{x} = (x_1, x_2, \dots, x_d)^\top$ , 其概率密度函数 (PDF) 可以表示为:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x} - \boldsymbol{\mu})\right)$$

其中,

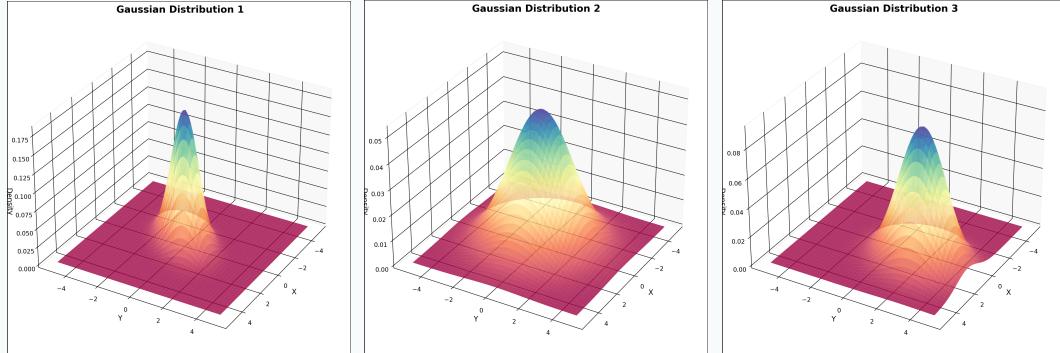
$$\boldsymbol{\mu} = \mathbb{E}[\mathbf{x}] \in \mathbb{R}^d, \quad \Sigma = \mathbb{E}[(\mathbf{x} - \mathbb{E}[\mathbf{x}])(\mathbf{x} - \mathbb{E}[\mathbf{x}])^T] \in \mathbb{R}^{d \times d}$$

$\boldsymbol{\mu}$  是均值向量, 表示高斯分布的中心,  $\Sigma$  是协方差矩阵, 描述了各维度之间的线性依赖关系。具体来说, 协方差矩阵  $\Sigma$  的元素  $\sigma_{ij}$  表示第  $i$  维与第  $j$  维的协方差。

协方差矩阵  $\Sigma$  不仅描述了各个维度之间的相关性, 还决定了分布的形状。协方差矩阵是对角阵意味着各维度之间是独立的, 分布在每一维度上是独立的高斯分布。若协方差矩阵是满秩的, 则各维度之间可能存在相关性, 分布的形状通常是椭圆形的。(见图??)

最大似然估计: 在给定一组样本数据  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$  的情况下, 高维高斯分布的最大似然估计 (MLE) 可以通过样本均值和样本协方差矩阵来得到。具体而言, 均值向量和协方差矩阵的估计分别为:

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{i=1}^N \mathbf{x}_i, \quad \hat{\Sigma} = \frac{1}{N-1} \sum_{i=1}^N (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^\top$$



$$\boldsymbol{\mu}_1 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \Sigma_1 = \begin{bmatrix} 1 & 0.5 \\ 0.5 & 1 \end{bmatrix} \quad \boldsymbol{\mu}_2 = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \quad \Sigma_2 = \begin{bmatrix} 3 & 0 \\ 0 & 3 \end{bmatrix} \quad \boldsymbol{\mu}_3 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \Sigma_3 = \begin{bmatrix} 1.5 & 0.3 \\ 0.3 & 2 \end{bmatrix}$$

Figure 34: Multivariate Gaussian Distribution

## I Proof of the solutions of GDA(2-classification case)

### Proof

我们的证明目标是式(35),(36),(37).

*Proof.* 对数似然函数  $l(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi)$  为:

$$l(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) = \sum_{i=1}^n \left[ \log \mathbb{P}(\mathbf{x}^{(i)} | y^{(i)}) + \log \mathbb{P}(y^{(i)}) \right]$$

其中,  $\mathbb{P}(\mathbf{x}|y=0) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_0, \Sigma)$  和  $\mathbb{P}(\mathbf{x}|y=1) = \mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_1, \Sigma)$ , 分别是标签为0和1时特征的条件概率密度。 $\mathbb{P}(y=0) = \phi$  和  $\mathbb{P}(y=1) = 1 - \phi$  分别为标签为0和1时的先验概率。

我们需要根据类别  $y^{(i)}$  来决定每个样本的似然:

$$\mathbb{P}(\mathbf{x}^{(i)} | y^{(i)} = 0) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)^T \Sigma^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) \right)$$

$$\mathbb{P}(\mathbf{x}^{(i)} | y^{(i)} = 1) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_1)^T \Sigma^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_1) \right)$$

因此, 总对数似然函数为:

$$l(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) = \sum_{i \in U_0} \left[ \log \mathbb{P}(\mathbf{x}^{(i)} | y^{(i)} = 0) + \log(1 - \phi) \right] + \sum_{j \in U_1} \left[ \log \mathbb{P}(\mathbf{x}^{(j)} | y^{(j)} = 1) + \log(\phi) \right]$$

展开对数似然函数时, 我们需要处理每个样本对应的对数概率:

$$\log \mathbb{P}(\mathbf{x}^{(i)} | y^{(i)} = 0) = -\frac{1}{2} \log |\Sigma| - \frac{d}{2} \log(2\pi) - \frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)^T \Sigma^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)$$

$$\log \mathbb{P}(\mathbf{x}^{(i)} | y^{(i)} = 1) = -\frac{1}{2} \log |\Sigma| - \frac{d}{2} \log(2\pi) - \frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_1)^T \Sigma^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_1)$$

因此, 完整的对数似然函数是:

$$\begin{aligned} l(\boldsymbol{\mu}_0, \boldsymbol{\mu}_1, \Sigma, \phi) &= -\frac{n}{2} \log |\Sigma| - \frac{nd}{2} \log(2\pi) + \sum_{i \in U_0} [\log(1 - \phi)] + \sum_{j \in U_1} [\log(\phi)] \\ &+ \sum_{i \in U_0} \left[ -\frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)^T \Sigma^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) \right] + \sum_{j \in U_1} \left[ -\frac{1}{2} (\mathbf{x}^{(j)} - \boldsymbol{\mu}_1)^T \Sigma^{-1} (\mathbf{x}^{(j)} - \boldsymbol{\mu}_1) \right] \end{aligned}$$

对  $\phi$  求导并令其为零:

$$\frac{\partial l}{\partial \phi} = -\frac{|U_0|}{1 - \phi} + \frac{|U_1|}{\phi} = 0 \Rightarrow \phi = \frac{|U_1|}{|U_0| + |U_1|} = \frac{|U_1|}{n} = \frac{1}{n} \sum_{i=1}^n \mathbb{I}(y^{(i)} = 1)$$

对  $\boldsymbol{\mu}_0$  求导并令其为零:

$$\frac{\partial l}{\partial \boldsymbol{\mu}_0} = \frac{\partial}{\partial \boldsymbol{\mu}_0} \sum_{i \in U_0} \left[ -\frac{1}{2} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0)^T \Sigma^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) \right] = \sum_{i \in U_0} \Sigma^{-1} (\mathbf{x}^{(i)} - \boldsymbol{\mu}_0) = 0$$

$$\Rightarrow \boldsymbol{\mu}_0 = \frac{1}{|U_0|} \sum_{i \in U_0} \mathbf{x}^{(i)}$$

同理对  $\mu_0$  求导并令其为零可得:

$$\mu_0 = \frac{1}{|U_1|} \sum_{j \in U_1} x^{(j)}$$

由于  $\frac{\partial |\Sigma|}{\partial \Sigma} = |\Sigma|(\Sigma^{-1})^T$ ,  $\frac{\partial \ln |\Sigma|}{\partial \Sigma} = \Sigma^{-T}$ ,  $\frac{\partial \Sigma^{-1}}{\partial \Sigma} = -\Sigma^{-1}\Sigma^{-1}$ , 因此对  $\Sigma$  求导, 得到:

$$\frac{\partial l}{\partial \Sigma} = \frac{1}{2} \left( \sum_{i=1}^n \left[ (\mathbf{x}^{(i)} - \mu_{y^{(i)}})^T \Sigma^{-1} \Sigma^{-1} (\mathbf{x}^{(i)} - \mu_{y^{(i)}}) \right] - n \Sigma^{-T} \right)$$

令其等于零, 得到:

$$\Sigma = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}^{(i)} - \mu_{y^{(i)}}) (\mathbf{x}^{(i)} - \mu_{y^{(i)}})^T$$

即:

$$\Sigma = \frac{1}{n} \left[ \sum_{i \in U_0} (\mathbf{x}^{(i)} - \mu_0) (\mathbf{x}^{(i)} - \mu_0)^T + \sum_{i \in U_1} (\mathbf{x}^{(i)} - \mu_1) (\mathbf{x}^{(i)} - \mu_1)^T \right]$$

□

## J Proof of the decision boundary (38)

### Proof of (38)

我们的证明目标是:

$$\begin{aligned} \mathbb{P}(y = 1 | \mathbf{x}; \mu_0, \mu_1, \Sigma, \phi) &= \frac{\mathbb{P}(\mathbf{x}|y = 1; \mu_0, \mu_1, \Sigma) \cdot \mathbb{P}(y = 1; \phi)}{\mathbb{P}(\mathbf{x}; \mu_0, \mu_1, \Sigma, \phi)} \\ &= \frac{1}{1 + \exp[-(\theta^T \mathbf{x} + \theta_0)]}, \quad \theta \in \mathbb{R}^d, \theta_0 \in \mathbb{R}, \text{ 均与参数 } \mu_0, \mu_1, \Sigma, \phi \text{ 相关} \end{aligned} \quad (138)$$

*Proof.* 根据贝叶斯公式和全概率公式, 可以得到

$$\mathbb{P}(y = 1 | \mathbf{x}) = \frac{\mathbb{P}(\mathbf{x}|y = 1) \mathbb{P}(y = 1)}{\mathbb{P}(\mathbf{x}|y = 1) \mathbb{P}(y = 1) + \mathbb{P}(\mathbf{x}|y = 0) \mathbb{P}(y = 0)}$$

同时已经假设

$$\mathbb{P}(y = 1) = \phi, \quad \mathbb{P}(y = 0) = 1 - \phi$$

$$\begin{aligned} \mathbb{P}(\mathbf{x}|y = 1) &= \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu_1)^T \Sigma^{-1} (\mathbf{x} - \mu_1) \right) \\ \mathbb{P}(\mathbf{x}|y = 0) &= \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp \left( -\frac{1}{2} (\mathbf{x} - \mu_0)^T \Sigma^{-1} (\mathbf{x} - \mu_0) \right) \end{aligned}$$

代入可得

$$\begin{aligned}
& \mathbb{P}(y = 1 | \mathbf{x}) \\
&= \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1)\right) \cdot \phi}{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1)\right) \cdot \phi + \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_0)^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_0)\right) \cdot (1 - \phi)} \\
&= \frac{1}{1 + \frac{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_0)^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_0)\right) \cdot (1 - \phi)}{\exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1)\right) \cdot \phi}} \\
&= \frac{1}{1 + \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_0)^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_0) + \frac{1}{2}(\mathbf{x} - \boldsymbol{\mu}_1)^T \Sigma^{-1}(\mathbf{x} - \boldsymbol{\mu}_1) + \log \frac{1 - \phi}{\phi}\right)}
\end{aligned}$$

这可以表示为：

$$\mathbb{P}(y = 1 | \mathbf{x}) = \frac{1}{1 + \exp[-(\theta^T \mathbf{x} + \theta_0)]}$$

其中  $\theta = \Sigma^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_0)$  和  $\theta_0 = \log \frac{\phi}{1 - \phi} + \frac{1}{2}(\boldsymbol{\mu}_0^T \Sigma^{-1} \boldsymbol{\mu}_0 - \boldsymbol{\mu}_1^T \Sigma^{-1} \boldsymbol{\mu}_1)$ . □

## K Codes for Multivariate Gaussian Distribution

### Multivariate Gaussian Distribution – Codes

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 from scipy.stats import multivariate_normal
4 from mpl_toolkits.mplot3d import Axes3D
5
6 def plot_3d_gaussian(mu, cov, title="3D Gaussian Distribution", save_path=None):
7     """
8         Plot the 3D Gaussian distribution
9
10    Parameters:
11        mu: Mean, a list of length 2 [mu_x, mu_y]
12        cov: Covariance matrix, a 2x2 2D array
13        title: Title of the plot
14        save_path: Path to save the plot (optional)
15    """
16    # Generate mesh grid data
17    x = np.linspace(-5, 5, 1000)
18    y = np.linspace(-5, 5, 1000)
19    X, Y = np.meshgrid(x, y)
20
21    # Calculate the probability density of the 2D Gaussian distribution
22    pos = np.dstack((X, Y))
23    rv = multivariate_normal(mu, cov)
24    Z = rv.pdf(pos)
25
26    # Create a 3D surface plot
27    fig = plt.figure(figsize=(12, 10), dpi=200)
28    ax = fig.add_subplot(111, projection='3d')
29
30    # Plot a smooth surface
31    surf = ax.plot_surface(X, Y, Z, cmap="Spectral", edgecolor='none',
32                           alpha=0.8)
33
34    # Set title and labels
35    ax.set_title(title, fontsize=16, fontweight='bold')
36    ax.set_xlabel("X", fontsize=12)
37    ax.set_ylabel("Y", fontsize=12)
38    ax.set_zlabel("Density", fontsize=12)
39
40    # Set the view angle to avoid a flat view
41    ax.view_init(30, 30)
42
43    # Set the grid lines to be black
44    ax.grid(True, color='black') # Show grid lines
45    ax.xaxis._axinfo['grid'].update(color='black') # Grid lines for X axis
46    ax.yaxis._axinfo['grid'].update(color='black') # Grid lines for Y axis
47    ax.zaxis._axinfo['grid'].update(color='black') # Grid lines for Z axis
48
49    # Set the axis tick marks to be black
50    ax.tick_params(axis='both', direction='in', length=6, width=1, colors='black')
51
52    # Set the external border lines to be black
53    fig.patch.set_edgecolor('black') # External border lines of the figure
54    fig.patch.set linewidth(2) # Set the thickness of the border
55    lines
```

```

54
55     # Remove the background color of the panels, making them transparent
56     ax.xaxis.pane.fill = True    # Transparent background for X axis
57     ax.yaxis.pane.fill = False   # Transparent background for Y axis
58     ax.zaxis.pane.fill = False   # Transparent background for Z axis
59
60     # ax.set_facecolor('none')    # Uncomment to make the plotting area
61     # background transparent
62     # fig.patch.set_alpha(0)    # Uncomment to make the figure background
63     # transparent
64
65     # fig.colorbar(surf, shrink=0.5, aspect=5)    # Add a color bar (optional)
66
67     plt.show()    # Display the plot
68
69     plt.draw()    # Force redraw
70     plt.savefig(save_path)
71
72     # Example calls:
73     mu1 = [0, 0]
74     cov1 = [[1, 0], [0, 1]]
75     plot_3d_gaussian(mu1, cov1, title="Gaussian Distribution 1", save_path="gaussian-1.png")
76
77     mu2 = [0, 0]
78     cov2 = [[1, 0.8], [0.8, 1]]
79     plot_3d_gaussian(mu2, cov2, title="Gaussian Distribution 2", save_path="gaussian-2.png")
80
81     mu3 = [1, 2]
82     cov3 = [[1.5, 0.3], [0.3, 2]]
83     plot_3d_gaussian(mu3, cov3, title="Gaussian Distribution 3", save_path="gaussian-3.png")

```

## L Different optimize methods

### Different optimize methods

#### Algorithm 3 Full Gradient Descent

**Require:** Dataset  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ , loss function  $J(\boldsymbol{\theta})$ , learning rate  $\eta$

- 1: **Initialize** parameters  $\boldsymbol{\theta}$
- 2: **repeat**
- 3:     Compute gradient:  $\mathbf{g} = \frac{1}{N} \sum_{i=1}^N \nabla J_i(\boldsymbol{\theta})$
- 4:     Update parameters:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \mathbf{g}$
- 5: **until** convergence

---

**Algorithm 4** Stochastic Gradient Descent (SGD)

---

**Require:** Dataset  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ , loss function  $J(\boldsymbol{\theta})$ , learning rate  $\eta$

- 1: **Initialize** parameters  $\boldsymbol{\theta}$
- 2: **repeat**
- 3:   Randomly sample a data point  $(\mathbf{x}_i, y_i)$
- 4:   Compute gradient:  $\mathbf{g} = \nabla J_i(\boldsymbol{\theta})$
- 5:   Update parameters:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \mathbf{g}$
- 6: **until** convergence

---

---

**Algorithm 5** Mini-batch Gradient Descent

---

**Require:** Dataset  $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$ , loss function  $J(\boldsymbol{\theta})$ , learning rate  $\eta$ , batch size  $m$

- 1: **Initialize** parameters  $\boldsymbol{\theta}$
- 2: **repeat**
- 3:   Randomly sample a mini-batch  $\mathcal{B}$  of  $m$  data points
- 4:   Compute gradient:  $\mathbf{g} = \frac{1}{m} \sum_{i \in \mathcal{B}} \nabla J_i(\boldsymbol{\theta})$
- 5:   Update parameters:  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \cdot \mathbf{g}$
- 6: **until** convergence

---

## M Codes for Lecture 10

### Codes

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4 # Set seed for reproducibility
5 np.random.seed(42)
6
7 # Generate training data
8 x = np.linspace(-3, 3, 100)
9 y_true = x**2 # True function (quadratic)
10 noise = np.random.normal(0, 1, size=x.shape) # Gaussian noise
11 y = y_true + noise # Add noise to the true function
12
13 # Polynomial fitting (degree 1, 2, 3)
14 degree_1 = np.polyfit(x, y, 1)
15 degree_2 = np.polyfit(x, y, 2)
16
17 # Create a smooth line for plotting
18 x_smooth = np.linspace(-3, 3, 500)
19 y_true_smooth = np.polyval([0, 0, 1], x_smooth) # Interpolated true
19 function
20
21 y_1 = np.polyval(degree_1, x_smooth)
22 y_2 = np.polyval(degree_2, x_smooth)
23
24 # Plot the results
25 fig, ax = plt.subplots(figsize=(12, 6), dpi = 200)
26
27 # Plot true function and noisy data
28 plt.scatter(x, y, color='gray', label='Noisy data', alpha=0.7)
29
30 # Plot polynomial fits
```

```

31 plt.plot(x_smooth, y_1, color=colors["hong"], label='Degree 1 (Underfitting
32     )', linestyle='--')
33 plt.plot(x_smooth, y_2, color=colors["mint"], label='Degree 2 (Good fit)',
34     linestyle='-')
35
36 # Labels and title
37 plt.xlabel('X')
38 plt.ylabel('Y')
39 plt.title('Underfitting Example')
40 plt.legend()
41
42 # grid
43 ax.grid(
44     linestyle="--",
45     linewidth=0.8,
46     color="gray",
47     alpha=0.3
48 )
49
50 ax.xaxis.set_major_locator(plt.MultipleLocator(1))
51 ax.xaxis.set_minor_locator(plt.MultipleLocator(0.2))
52 ax.yaxis.set_major_locator(plt.MultipleLocator(1.5))
53 ax.yaxis.set_minor_locator(plt.MultipleLocator(0.5))
54 ax.tick_params(axis='x', which='major', length=7)
55 ax.tick_params(axis='x', which='minor', length=4)
56 ax.tick_params(axis='y', which='major', length=7)
57 ax.tick_params(axis='y', which='minor', length=4)
58
59 ax.tick_params(axis='x', which='both', top=True, direction='in')
60 ax.tick_params(axis='y', which='both', right=True, direction='in')
61
62 # Set transparent background
63 plt.gcf().set_facecolor('none')
64
65 plt.savefig("underfitting.png", dpi = 200)
plt.show()

```

## N Weight Decay

### Weight Decay

**Proposition 2.** *SGD with weight decay  $\lambda$  is:*

$$\theta_{t+1} = (1 - \lambda)\theta_t - \alpha \nabla f_t(\theta_t), \quad (139)$$

where  $\lambda$  is the weight decay coefficient. It is equivalent to the following update rule:

$$\theta_{t+1} = \theta_t - \alpha \nabla f_t^{\text{reg}}(\theta_t), \quad f_t^{\text{reg}}(\theta) = f_t(\theta) + \frac{\omega}{2} \|\theta\|_2^2, \quad \omega = \frac{\lambda}{\alpha}. \quad (140)$$

*Proof.* SGD without weight decay has the following iterates on  $f_t^{\text{reg}}(\theta) = f_t(\theta) + \frac{\omega}{2} \|\theta\|_2^2$ :

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla f_t^{\text{reg}}(\theta_t) = \theta_t - \alpha \nabla f_t(\theta_t) - \alpha \omega \theta_t \quad (141)$$

SGD with weight decay has the following iterates on  $f_t(\theta)$ :

$$\theta_{t+1} \leftarrow \theta_t - \alpha \nabla f_t(\theta_t) - \lambda \theta_t. \quad (142)$$

These iterates are identical since  $\omega = \frac{\lambda}{\alpha}$ . □

## O Proof of Eq. (75)

### Proof of Eq. (75)

*Proof.* 根据凸函数定义, 有:

$$\forall \lambda \in [0, 1], \lambda(a, f(a)) + (1 - \lambda)(b, f(b)) \in \Omega \quad (143)$$

因此定义  $z = \lambda a + (1 - \lambda)b$ , 则有

$$(z, \lambda f(a) + (1 - \lambda)f(b)) \in \Omega \Rightarrow \lambda f(a) + (1 - \lambda)f(b) \geq f(z) \quad (144)$$

□

## P Proof of Theorem 4

### Proof of Theorem 4

**Proposition 3.**

$$f''(x) \geq 0, \forall x \Rightarrow f \text{ is convex.} \quad (145)$$

*Proof.* 由 Taylor expansion 可得,  $\forall z \in [a, b]$ :

$$\begin{aligned} f(a) &= f(z) + f'(\alpha)(a - z) + f''(\alpha)(a - z)^2, \alpha \in [a, z] \\ f(b) &= f(z) + f'(\beta)(b - z) + f''(\beta)(b - z)^2, \beta \in [z, b], \end{aligned} \quad (146)$$

因此有

$$\lambda f(a) + (1 - \lambda)f(b) = f(z) + O(z) + C \geq f(z) \quad (147)$$

□

## Q Proof of MLE of Gaussian

### Proof of MLE of Gaussian

*Proof.* 对于高斯分布  $x \sim \mathcal{N}(\mu, \Sigma) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right)$ ,  $\mu \in \mathbb{R}^d$ ,  $\Sigma \in \mathbb{R}^{d \times d}$ , 其 MLE 为:

$$\max_{\mu, \Sigma} \ell(\mu, \Sigma) = \sum_{i=1}^n \log p(x^{(i)} | \mu, \Sigma) = \min_{\mu, \Sigma} \sum_{i=1}^n \left[ (x^{(i)} - \mu)^\top \Sigma^{-1}(x^{(i)} - \mu) + \log |\Sigma| \right]. \quad (148)$$

再对  $\mu$  求梯度 (假设  $\Sigma$  满秩) 寻找一阶最优性条件:

$$\nabla_\mu f(\mu) = \sum_{i=1}^n \Sigma^{-1}(\mu - x^{(i)}) = 0 \Rightarrow \sum_{i=1}^n (\mu - x^{(i)}) = 0 \Rightarrow \hat{\mu} = \frac{1}{n} \sum_{i=1}^n x^{(i)} \quad (149)$$

代入  $\mu = \hat{\mu}$ , 定义

$$S \triangleq \sum_{i=1}^n (x^{(i)} - \hat{\mu})(x^{(i)} - \hat{\mu})^\top \Rightarrow \ell(\Sigma) = -\frac{n}{2} \log |\Sigma| - \frac{1}{2} \text{tr}(\Sigma^{-1} S). \quad (150)$$

对矩阵  $\Sigma$  进行求导, 得到:

$$\frac{\partial}{\partial \Sigma} \log |\Sigma| = \Sigma^{-1}, \quad \frac{\partial}{\partial \Sigma} \text{tr}(\Sigma^{-1} S) = -\Sigma^{-1} S \Sigma^{-1}, \quad (151)$$

得到一阶最优性条件:

$$-\frac{n}{2} \Sigma^{-1} + \frac{1}{2} \Sigma^{-1} S \Sigma^{-1} = 0 \Rightarrow S = n \Sigma \Rightarrow \hat{\Sigma} = \frac{1}{n} S. \quad (152)$$

□

## References

- [1] Arthur L Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of research and development*, 11(6):601–617, 1967.
- [2] Tom M Mitchell and Tom M Mitchell. *Machine learning*, volume 1. McGraw-hill New York, 1997.
- [3] Su-In Lee, Dana Pe'er, Aimée M Dudley, George M Church, and Daphne Koller. Identifying regulatory mechanisms using individual variation reveals key role for chromatin modification. *Proceedings of the National Academy of Sciences*, 103(38):14062–14067, 2006.
- [4] Tomas Mikolov. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.
- [5] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.
- [6] Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models. *arXiv preprint arXiv:2410.05229*, 2024.
- [7] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.
- [8] F Vallet, J-G Cailton, and Ph Refregier. Linear and nonlinear extension of the pseudo-inverse solution for learning boolean functions. *Europhysics Letters*, 9(4):315, 1989.
- [9] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.
- [10] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.
- [11] Song Mei and Andrea Montanari. The generalization error of random features regression: Precise asymptotics and the double descent curve. *Communications on Pure and Applied Mathematics*, 75(4):667–766, 2022.
- [12] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
- [13] Rebecca Roelofs, Vaishaal Shankar, Benjamin Recht, Sara Fridovich-Keil, Moritz Hardt, John Miller, and Ludwig Schmidt. A meta-analysis of overfitting in machine learning. *Advances in neural information processing systems*, 32, 2019.
- [14] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.
- [15] Rishi Bommasani. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

- [16] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.
- [17] George EP Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.