

2023-02-22

机器学习模型与算法

何琨

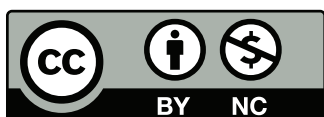
华中科技大学计算机科学与技术学院

$$\rho := \frac{1 + \sqrt{2}}{2}$$

Version 2.0

编译日期: 2023-02-22

对本讲义的订正或改进建议请发邮箱:
brooklet60@hust.edu.cn



注：本模板采用知识共享署名-非商业性使用 4.0 国际许可协议进行许可。访问<http://creativecommons.org/licenses/by-nc/4.0/>查看该许可协议。

前 言

本机器学习讲义的 V1.0 版本为华中科技大学计算机科学与技术学院机器学习与数据挖掘实验室 (John Hopcroft Lab) 的研究生同学们于 2019-2020 年根据我的授课内容，参考康奈尔大学机器学习课程 CS4780 整理而成。感谢 JHL 实验室 18、19 和部分 20 级同学们的整理。此 V2.0 版本于 2021 年暑期和 2022 年暑期，在此基础上进行了一些修正和完善。感谢 JHL 实验室部分 21 级、22 级同学的修改润色。此版本将用于 2023 年春季华中科技大学计算机科学与技术学院本科生的教学内部讲义，后续将在此内容上不断完善。

何 琨

2022.2.20

目 录

前 言	i
第一章 引言	1
1.1 什么是机器学习	1
1.2 机器学习的历史	2
1.3 机器学习算法的类型	4
1.3.1 监督学习算法	5
1.3.2 无监督学习算法	5
1.3.3 强化学习算法	5
第二章 监督学习	7
2.1 介绍	7
2.2 模型	7
2.2.1 标签空间实例	8
2.2.2 特征向量实例	8
2.3 损失函数	9
2.3.1 常见损失函数	9
2.4 泛化	10
2.5 训练集/测试集划分	10
2.5.1 如何划分数据	11
2.6 总结	11
第三章 k 近邻和维数灾难	13
3.1 k 近邻算法	13
3.1.1 概念介绍	13
3.1.2 正式定义	14

3.1.3	k 值的选取	14
3.1.4	距离函数的选择	14
3.2	最近邻分类器	14
3.2.1	贝叶斯最优分类器	15
3.2.2	最近邻分类器的收敛性	15
3.3	维数灾难	16
3.3.1	点和点之间的距离	16
3.3.2	点和超平面之间的距离	16
3.3.3	低维结构的数据流形	18
3.4	k -均值算法	18
第四章	概率估计与朴素贝叶斯	19
4.1	概率分布	19
4.1.1	离散型变量和概率质量函数	19
4.1.2	连续型变量和概率密度函数	20
4.1.3	边缘概率	20
4.1.4	条件概率	21
4.2	最大似然估计 (MLE, Maximum Likelihood Estimation)	21
4.3	最大后验估计 (MAP, Maximum A Posteriori Probability Estimate)	23
4.4	朴素贝叶斯分类器	24
4.5	朴素贝叶斯法	25
4.6	$P([\mathbf{x}]_a y)$ 参数估计	27
4.6.1	实例 1: 离散特征	27
4.6.2	实例 2: 多项式特征	28
4.6.3	实例 3: 连续特征 (高斯朴素贝叶斯法)	29
4.7	朴素贝叶斯法是一种线性分类器	29
4.8	思考题	31
第五章	线性模型	33
5.1	感知机	33
5.1.1	感知机分类模型	33
5.1.2	感知机算法	34
5.1.3	感知机收敛性证明	36
5.2	线性回归	37

5.2.1	假设	38
5.2.2	最大似然估计	38
5.2.3	最大后验估计	40
5.3	逻辑回归	41
5.3.1	逻辑回归	41
5.3.2	最大似然估计	43
5.3.3	最大后验估计	44
5.3.4	逻辑回归与朴素贝叶斯	44
5.4	本章总结	44
第六章	梯度下降	47
6.1	数学基础: 泰勒展开	47
6.2	梯度下降法	48
6.3	基本一阶算法	49
6.3.1	小批量样本梯度下降: 使用一阶近似	49
6.3.2	动量梯度更新	50
6.3.3	Nesterov 动量方法	50
6.4	二阶近似方法	51
6.4.1	牛顿法	51
6.4.2	共轭梯度法	52
6.5	自适应学习率算法	53
6.5.1	AdaGrad 算法	53
6.5.2	RMSProp 算法	54
6.5.3	Adam	54
6.5.4	思考题	56
第七章	支持向量机	57
7.1	线性分类问题	57
7.2	SVM——线性可分情形	57
7.2.1	主优化问题	58
7.2.2	对偶优化问题	58
7.2.3	支持向量	59
7.3	SVM——线性不可分情形	59
7.3.1	对偶优化问题	60

7.3.2	KKT 条件	61
7.4	核函数	61
7.4.1	多项式核函数	61
7.4.2	高斯核函数	62
7.4.3	sigmoid 核函数	62
第八章	学习准则与模型选择	63
8.1	学习准则	63
8.1.1	损失函数和经验风险最小化准则	63
8.1.2	过拟合和欠拟合	64
8.1.3	结构风险最小化	64
8.2	偏差-方差分析	65
8.2.1	偏差-方差分解	66
8.2.2	偏差-方差窘境	67
8.2.3	模型选择	67
第九章	核方法	69
9.1	手动扩展特征	69
9.2	核技巧	70
9.2.1	平方损失的梯度下降	70
9.2.2	内积计算	72
9.3	常见核函数	73
9.4	定义良好的核	73
9.5	核机器学习	74
9.5.1	核回归	75
9.5.2	核化最近邻算法	76
9.5.3	核支持向量机	76
第十章	高斯过程	79
10.1	高斯分布的性质	79
10.2	高斯过程回归	80
10.2.1	后验预测分布	80
10.2.2	高斯过程	80
10.2.3	高斯过程回归	81

10.2.4 加性高斯噪声	82
10.3 高斯过程的参数学习	82
10.3.1 边缘似然和超参数学习	82
10.3.2 协方差函数- 高斯过程模型的核心	83
10.3.3 高斯过程总结	83
10.4 贝叶斯全局优化	83
第十一章 KD Trees	85
11.1 KD 树	85
11.1.1 创建 KD 树	86
11.1.2 近邻搜索	86
11.2 球树	87
11.2.1 创建球树	88
11.2.2 球树的应用场景	88
11.3 总结	88
第十二章 决策树	91
12.1 基本概念	92
12.1.1 信息熵	93
12.1.2 信息增益	93
12.1.3 增益率	94
12.1.4 基尼指数	94
12.2 ID3	95
12.3 CART	95
第十三章 Bagging	97
13.1 Bagging	97
13.1.1 bagging 减小方差	97
13.1.2 方法: Bagging(Bootstrap Aggregating)	98
13.1.3 Bagging 概括	99
13.1.4 Bagging 的优点	99
13.2 随机森林	100

第十四章 Boosting	103
14.1 Boosting 减小偏差	103
14.2 函数空间中的梯度下降	104
14.3 泛化的 Boosting 算法 (Anyboost)	104
14.4 案例 1: 梯度增强回归树 (GBRT)	105
14.5 案例 2: 自适应提升 (AdaBoost)	106
14.5.1 寻找最好的弱学习器	106
14.5.2 寻找步长 α	107
14.5.3 Re-normalization	108
14.5.4 进一步分析	109
14.6 总结	110
参考文献	111

1

引言

机器学习 (Machine Learning, ML) 是人工智能 (Artificial Intelligence, AI) 的一个重要分支。不同的人对机器学习的理解和定义可能不同,但都具有以下共同点,输入数据的维度较高,数据量较大,学习过程中只需少量甚至无需进行人工干预或指导。本章将对什么是机器学习给出一个概述。

1.1 什么是机器学习

我们知道,算法作为计算机科学的核心和灵魂,是被定义好的、计算机可施行其指示的有限步骤或次序的方法。对于待求解的问题,计算机科学的经典途径是通过设计算法并将之实现为程序。我们有这样一种认知,即“程序 = 数据结构 + 算法”,其中,数据结构是计算机中存储、组织数据的方式与结构。然后,对于输入数据即问题实例将其使用数据结构存储,然后进行程序计算,输出求解的结果。

不同于经典的算法,机器学习是一类全新的问题求解范式,并非通过已设计好的程序方法,而是设计与分析让计算机可以自动“学习”的算法,从而对未来数据或未知数据进行预测。在有监督模式下,机器学习通过对已知输入和输出的示例数据进行学习来产生模型 (Model),使模型可以对该问题的新的实例输出结果。Tom Mitchell [1] 于 1997 年给出了机器学习的形式化定义: 设度量 P 可以用来评估算法在某任务类 T 上的性能,若计算机程序 A 通过经验 E 使得其在该类任务上的性能得到了提升,则称该程序对经验 E 进行了学习。从 20 世纪 80 年代开始,特别是 90 年代至今,机器学习快速发展并形成了一个独立的学科领域,这与信息化时代的海量数据、计算机的计算能力的显著提升和更好的算法开发工具和开源包等因素紧密相关。

机器学习的应用非常广泛，人们使用的各类设备以及各种应用平台等都用到机器学习技术，如手机的小应用程序、可穿戴式健身追踪器以及智能家居助手，以及打车、在线购物等应用平台。机器学习还可以用于如下的一些应用场景：

- ▶ 预测：如天气预报、股票预测。
- ▶ 图像识别：如人脸检测、车牌识别、物品分类等。
- ▶ 语音识别：如将语音翻译成文本，可用于语音搜索等。
- ▶ 医学诊断：辅助医生对扫描图片进行医学诊断，如识别癌组织、判断疾病类型及轻重程度等。
- ▶ 金融业和交易：公司在欺诈调查和信用检查中可通过机器学习提升效率和准确性。

简单来说，机器学习是指系统不需要编写特定的算法，而从示例数据中学习解决问题的方法。套用人工智能领域的先驱 Arthur Samuel [2] 的观点，机器学习算法使计算机能够从数据中学习，甚至可以自我改进，而无需进行显式的编程¹。机器学习是一种新的算法类别，它允许软件应用程序在未经显式编程的情况下更准确地预测出结果。机器学习的基本前提是构建可以接收输入数据的算法，并使用统计分析来预测输出，同时出现新数据时能够迭代更新得到新的输出。

如图1-1所示，传统的算法是通过分析问题并得到一系列解决问题的清晰指令。在传统计算机科学领域，通过数据结构读取数据并使用其控制逻辑来执行相应的程序，得到期望的结果。这些逻辑代码语句中的每一个都有必须定义的测试数据。但是，在机器学习中定义这些不断变化的测试数据是非常困难的。在机器学习算法中，我们不会编写算法逻辑，而是通过收集数据，运用这些数据分析出数据隐藏的模式，然后将此模式传递给算法，得到一个模型，最后通过该模型对数据产生解决方案。

1.2 机器学习的历史

机器学习是人工智能的一个子领域。人工智能的发展早期有三个标志性事件。

- ▶ 第一个从“经验”中学习的程序是 1959 年由 Arthur Lee Samuel 设计的一种西洋跳棋 (Checker) 程序 [2]。该程序主要是基于 minimax 策略，即每一步操作均最大化自身的价值，并假设对手也从对方的角度最大化其价值。它利用多个简单的学习算法来改进对当前棋面的评估。该程序通过自我学习，使得游戏的过程中

¹原话是：“A computer can be programmed so that it will learn to play a better game of checkers than can be played by the person who wrote the program.” “Programming computers to learn from experience should eventually eliminate the need for much of this detailed programming effort.” 后人将其提炼为：“Field of study that gives computers the ability to learn without being explicitly programmed”。

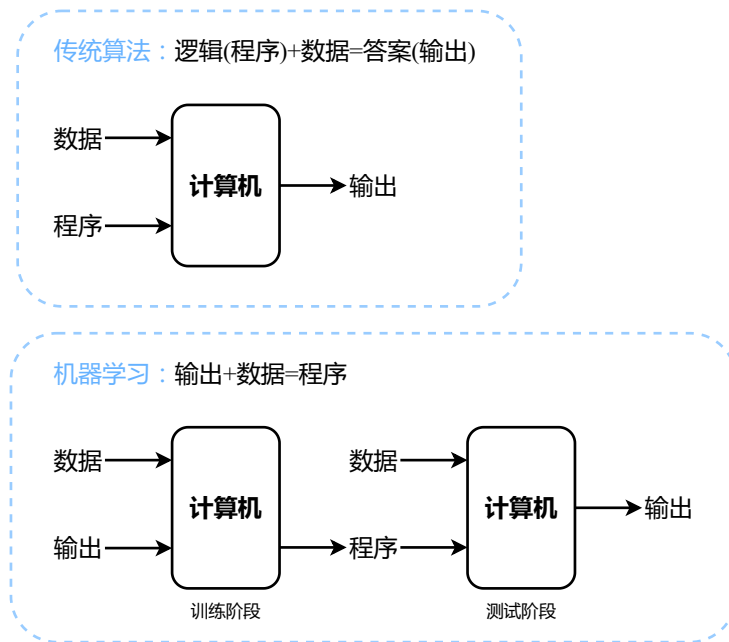


图 1-1 机器学习算法与传统计算机算法的主要区别

Figure 1-1 Algorithms in traditional computer science versus machine learning algorithms

不断更新其效果。

- ▶ 1957 年，康奈尔大学的 Frank Rosenblatt 提出了感知机 (Perceptron) 模型。这是人工神经网络和现代深度学习的早期雏形。
- ▶ 1969 年，Marvin Minsky 和 Seymour Papert 指出，感知机模型难以解决简单的异或 (XOR) 操作，使人工智能的发展进入了冬天。

人工智能的终极目标是尝试模仿人类的智慧，尝试模拟人类的逻辑推理。而机器学习从 20 世纪 80 年代开始被提出，则采取了从底向上的模式，即从求解一些具体的实际问题入手，基于的是统计学习与优化技术，而不是逻辑推理，得到了快速的发展。并由于计算机科学与统计学的交叉，使得机器学习领域进一步转向数据驱动。随着信息化时代的到来，获得大规模数据后，科学家们开始构建能够分析和学习海量数据的智能系统。1994 年，IBM 公司的 Gerry Tesauro 尝试教神经网络学习一种棋类博弈 (BackGammon)，该神经网络通过自我对弈 10 万多次后，击败了当时的世界冠军 [3]。随后，Tesauro 将此技术用于训练下象棋的程序。1997 年，IBM 公司的深蓝 (Deep Blue) 系统击败了当时的世界国际象棋冠军。

现在，机器学习技术的应用几乎无处不在，如搜索引擎、垃圾邮件过滤、推荐系统、打车系统、手机拍照等。2010 年以后，作为机器学习的一个分支，深度学习得到迅猛的发展，使得人工智能进入到一个繁荣期，深度学习相关的工业应用如智能监控、自动驾

驶、智能机器人等也在快速的发展中。

1.3 机器学习算法的类型

机器学习根据数据训练方式的不同，可大致分为以下三种学习算法。同时，根据输入数据为连续还是离散，又分为多种子类型。

- ▶ 监督学习：样本数据包含预期的正确输出即标记/标签 (Label)，通过学习使得模型对新的数据能产生正确的输出，如图像数据的分类。
- ▶ 无监督学习：仅提供了数据而没有标注，需要从中挖掘数据模式、结构或子空间等。如对相似特征的数据进行聚类、对高维数据的降维等。
- ▶ 强化学习：通过并从反馈中进行学习。如机器人的行走学习、游戏 AI、实时决策系统等。

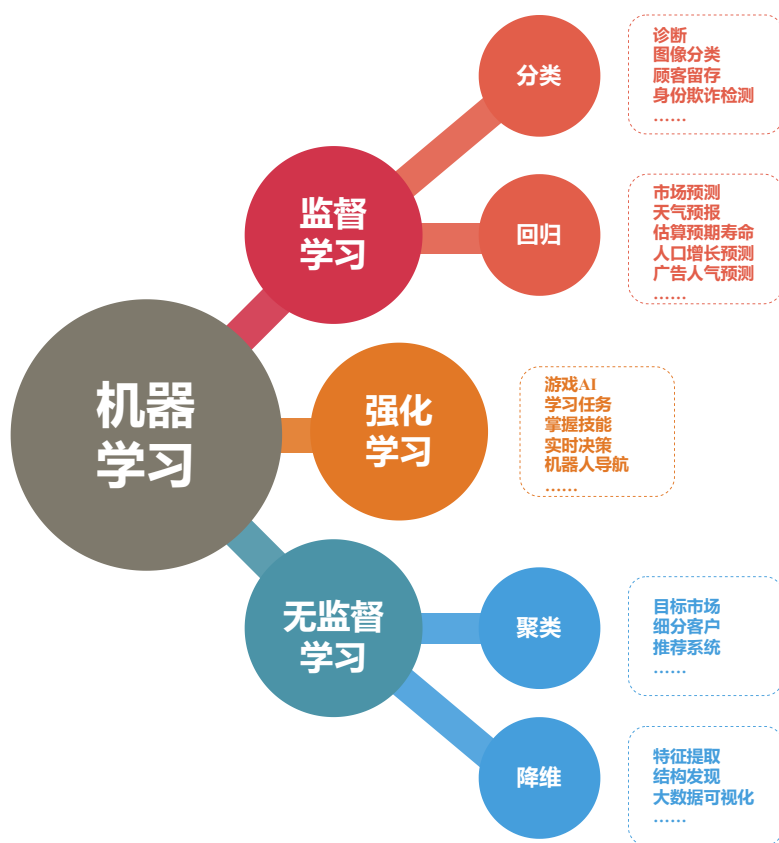


图 1-2 机器学习的类型

Figure 1-2 Machine learning category

1.3.1 监督学习算法

在监督学习中，每个数据都标记有正确的标记或标签 (label)，模型训练的目标是很好地近似映射函数，使得当有新的输入数据时，可以准确预测该数据的输出。监督学习在连续数据场景主要包括回归、决策树、随机森林等，在离散数据场景主要包括分类任务，典型算法包括 KNN、逻辑回归、朴素贝叶斯、支持向量机等。监督学习最典型的任务包括：

- ▶ 分类：输出变量是离散的类别，例如对各种图像的分类，包括“汽车”、“飞机”、“自行车”等；或对手写数字的识别，包括“0”、“1”、“2”等；或对文档的分类，如将邮件分为“垃圾邮件”和“非垃圾邮件”。
- ▶ 回归：输出变量是实数值，例如“身高”、“重量”、“温度”、“气压”等。可用于预测如明天的气温和湿度、根据照片判断人的年龄等。

1.3.2 无监督学习算法

在无监督学习场景，只有输入数据，没有标签，需要从输入数据中学习“隐藏的模式”，因此也称为知识发现 (Knowledge Discovery)。无监督学习在连续数据场景主要包括聚类和降维两类任务，在离散数据场景主要包括关联分析和隐马尔科夫模型 (Hidden Markov Model)。

无监督学习最典型的任务包括：

- ▶ 聚类：聚类问题希望发现数据中的内在组别，例如通过购买行为对客户进行分组。
- ▶ 降维：对于高维的数据，保留其主要特征，映射到低维空间中。典型的降维方法包括主成分分析 (Principal Components Analysis, PCA)。
- ▶ 数据补全：对缺失数据进行预测和补全。数据多以矩阵的形式出现，因此也称为矩阵补全 (Matrix Completion)，包括图像修复、协同过滤、关联分析等。

1.3.3 强化学习算法

强化学习算法通过与其环境交互得到的反馈来进行学习。正确执行而获得奖励，错误执行而受到处罚。强化学习通过最大化其奖励并最小化其惩罚而在没有人干预的情况下学习。强化学习是一种动态编程，它使用奖励和惩罚系统训练算法，通过改善其环境知识来选择下一个动作。

本书将重点讲述各类典型的监督学习模型与算法。

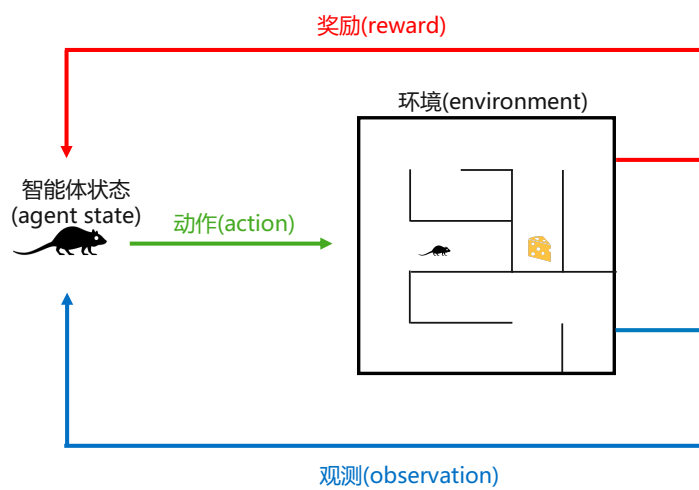


图 1-3 强化学习示意图

Figure 1-3 Illustration for reinforcement learning

2

监督学习

2.1 介绍

监督学习的目标是根据已有的数据与标签拟合模型。监督学习在人类生产生活中都有广泛的应用，其中一个流行应用是电子邮件的垃圾邮件过滤。按照传统计算机的算法设计思想，我们会去编写一个精心设计的程序，该程序遵循一些规则来决定电子邮件是否是垃圾邮件。虽然这样的程序可能会在一段时间内运行正常。随着垃圾邮件的迭代，它必须不断被重写，垃圾邮件发送者可能会尝试对软件进行逆向工程并设计绕过它的方法。机器学习则通过垃圾邮件数据生成可以预测垃圾邮件模式的程序，不是手动编程，而是从过去的学习中学习。如果有积累的数据实例，且确切地知道正确的标签是什么，那么通过机器学习方法就得到有效过滤垃圾邮件的算法。例如，过去的数据已经被用户标记为垃圾邮件或非垃圾邮件，机器学习算法可以利用这样的数据来训练分类器，以预测新的数据实例的正确标签。

2.2 模型

我们的训练数据是形如 (\mathbf{x}, y) 的输入对，其中 $\mathbf{x} \in R^d$ 是输入实例， y 是标签。完整的训练数据可以表示为：

$$D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\} \subseteq R^d \times c. \quad (2.1)$$

其中：

- ▶ R^d 是 d 维特征空间
- ▶ \mathbf{x}_i 是第 i 个样本的特征向量
- ▶ y_i 是第 i 个样本的标签
- ▶ c 是标签空间

数据点 (\mathbf{x}_i, y_i) 来源于某个分布 $P(X, Y)$ 。我们想要学习函数 h ，使得对于新的数据点 $(\mathbf{x}, y) \sim P$ ，有较高的概率使得 $h(\mathbf{x}) = y$ 或者 $h(\mathbf{x}) \approx y$ 。监督学习的整体模型如下：

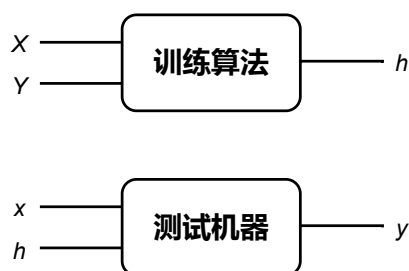


图 2-1 监督学习

Figure 2-1 Supervised learning

由于我们不知道真实的映射函数 h ，因而只能根据经验来假设一个函数集合，这个函数集合叫做假设空间。假设空间可以包括多种映射函数，如线性分类函数、决策树、人工神经网络、支持向量机 (SVM) 等等。

下面，对于数据 X 与标签 Y 的不同形式举几个例子。

2.2.1 标签空间实例

对于标签空间 c 有以下几种情形：

二分类	$c = \{0, 1\}$ 或 $c = \{-1, +1\}$	如垃圾邮件过滤，一封邮件是垃圾邮件 (+1) 或者不是 (-1)
多分类	$c = \{0, 1, \dots, K\} (K \geq 2)$	如人脸分类器，人脸图像属于 K 个人中的某个 (如，“1”表示张三，“2”表示李四)
回归	$c = R$	如预测未来某时的温度或人的身高

2.2.2 特征向量实例

我们称 \mathbf{x}_i 为特征向量，每个特征表示 \mathbf{x}_i 有 d 维，其中每一维表示第 i 个样本的相应特征，以下举几个例子。

病人数据： $\mathbf{x}_i = \{x_i^1, x_i^2, \dots, x_i^d\}$ ，其中 $x_i^1=0$ 或 1，表示第 i 个病人的性别， x_i^2 表示第 i 个病人的身高， x_i^3 表示第 i 个病人的年龄，等等。在这种情况下，一般有 $d \leq 100$ 并且特征向量比较密集，即， \mathbf{x}_i 中的非零值的数量相对于 d 是较大的。

文本文档: $\mathbf{x}_i = \{x_i^1, x_i^2, \dots, x_i^d\}$, 其中 x_i^α 是第 i 篇文档中第 α 个单词出现的次数。在这种情况下, $d \sim 10000 - 10\mathbf{M}$ 并且特征向量经常是稀疏的, 即 \mathbf{x}_i 主要由零组成。避免使用字典的一种常用方法是使用特征散列来直接将任何字符串散列到维度索引。

图片: 这里特征代表像素值。 $\mathbf{x}_i = \{x_i^1, x_i^2, \dots, x_i^{3k}\}$, 其中 x_i^{3j-2} , x_i^{3j-1} , x_i^{3j} 表示在图片中第 j 个像素的红、绿、蓝的值。 $d \sim 10000$ 甚至更大, 且特征向量是密集的。

2.3 损失函数

通常需要两步去学习一个假设函数 $\mathbf{h}(\cdot)$ 。首先, 我们选择适合这个学习问题的机器学习算法。这定义了假设空间 \mathcal{H} , 即我们可能学习的函数集合。第二步则是找到这个类中的最佳函数, $\mathbf{h} \in \mathcal{H}$ 。第二步实际上是一个学习训练的过程, 一般来说会涉及到优化问题。本质上, 我们试图在假设类中找到一个函数 h , 使得它在我们的训练数据中得到。(如果出现多个函数满足条件, 我们通常会通过一些类似“奥卡姆剃刀”的方式来选择“最简单”的函数——但是我们将在后面更详细地讨论这个问题。) 如何找到最好的函数? 为此, 我们需要某种方法来评估一个函数的效果。这就是损失函数 (又称风险函数) 的作用。对于训练集, 一个损失函数对假设函数, $\mathbf{h} \in \mathcal{H}$ 进行评估, 用来量化模型预测和真实标签之间的差异。损失越大, 则预测的准确率就越低——损失为零意味着对每一个样本, 假设函数都可以做出完美的预测。通常, 我们用损失函数除以用训练样本的总数 n , 这样输出即为每个样本的平均损失 (与 n 无关)。

2.3.1 常见损失函数

0-1 损失函数: 最简单的损失函数是 0-1 损失函数。它计算假设函数 h 在训练集上的错误率。对于每一个样本, 如果预测错误, 则损失为 1, 否则损失为 0。0-1 损失函数常用于多分类或二分类环境下的分类器评估。但很少用于优化过程, 因为 0-1 损失函数不可微的且非连续的, 因此常常难以优化。形式上, 零一损失可以表述为:

$$\mathcal{L}_{0/1}(\mathbf{h}) = \frac{1}{n} \sum_{i=1}^n \delta_{\mathbf{h}(x)_i \neq y_i}, \text{ where } \delta_{\mathbf{h}(x)_i \neq y_i} = \begin{cases} 1 & \text{if } \mathbf{h}(x)_i \neq y_i \\ 0 & \text{o.w.} \end{cases}$$

平方损失函数: 平方损失函数通常用于回归问题中。它遍历所有的样本, 并且以 $(\mathbf{h}(x)_i - y_i)^2$ 为损失。平方损失有两个性质: 首先损失是非负的, 其次损失以预测的绝对误差的平方增长。后一种性质则使得预测值距离实际值不会太远。另一方面, 如果一个预测非常接近于正确值, 那么这个平方误差就会很小, 这个样本的误

差则很难得到更新。例如，如果 $|(\mathbf{h}(x)_i - \mathbf{y}_i)| = 0.001$ ，那么平方损失函数会更小，0.000001，并且很可能永远都无法纠正。如果给定一个输入 \mathbf{x} 和标签 \mathbf{y} ，根据分布概率 $\mathbf{P}(\mathbf{x}|\mathbf{y})$ ，平方损失最小化得到的最优预测函数是其概率分布下的期望值，即 $\mathbf{h}(x) = \mathbf{E}_{\mathbf{P}(\mathbf{x}|\mathbf{y})}[\mathbf{y}]$ ，那么平方损失为：

$$\mathcal{L}_{sq}(\mathbf{h}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{h}(x)_i - \mathbf{y}_i)^2$$

绝对损失函数：与平方损失类似，绝对损失函数也通常用于回归问题。它对样本预测做出 $|\mathbf{h}(x)_i - \mathbf{y}_i|$ 的处罚。由于损失与错误预测呈线性增长，因此更适合于噪声数据（当一些错误预测不可避免且不应主导损失时）。如果给定一个输入 \mathbf{x} 和标签 \mathbf{y} ，根据分布 $\mathbf{P}(\mathbf{x}|\mathbf{y})$ 的概率。那么为使绝对损失最小化，最优预测函数是预测中间值，即 $\mathbf{h}(x) = \text{MEDIAN}_{\mathbf{P}(\mathbf{x}|\mathbf{y})}[\mathbf{y}]$ ，那么绝对损失可表示为：

$$\mathcal{L}_{abs}(\mathbf{h}) = \frac{1}{n} \sum_{i=1}^n |\mathbf{h}(x)_i - \mathbf{y}_i|$$

2.4 泛化

给定一个损失函数，我们尝试训练得到使损失最小化的函数 \mathbf{h} ：

$$\mathbf{h} = \arg \min_{\mathbf{h} \in \mathcal{H}} \mathcal{L}(\mathbf{h})$$

我们在训练集上训练模型，但往往我们想知道我们训练的模型并非只针对该训练集，而希望它在其他训练集有类似的效果。比如我们发现在数据集 \mathcal{D} 上有一个使得损失函数较低的函数 $\mathbf{h}(\cdot)$ ，如何知道它是否可以在其他数据集的推断能力。

错误的例子 “存储器” $\mathbf{h}(\cdot)$

$$\mathbf{h}(x) = \begin{cases} \mathbf{y}_i, & \text{if } \exists (\mathbf{x}_i, \mathbf{y}_i) \in \mathcal{D}, s.t., \mathbf{x} = \mathbf{x}_i \\ 0, & o.w. \end{cases}$$

对于这样的 $\mathbf{h}(\cdot)$ ，仅仅对数据集进行“记忆”，对于训练集 \mathcal{D} 上我们可以得到 100% 的正确率，但如果在训练集 \mathcal{D} 以外的数据集测试上，那么情况就很糟糕，几乎失去了效果。那么我们可以说这个函数存在**过拟合问题**，也就是说这个模型有着很差的**泛化**。

2.5 训练集/测试集划分

为解决过拟合问题，我们将数据集 \mathcal{D} 划分 为三个子集：其中 \mathcal{D}_{TR} 为训练集， \mathcal{D}_{VA} 为验证集， \mathcal{D}_{TE} 为测试集，通常我们以 80%，10%，10% 的比例来划分。然后，通过

\mathcal{D}_{TR} 数据集来训练 $\mathbf{h}(\cdot)$ ，根据 \mathcal{D}_{TE} 来测试 $\mathbf{h}(\cdot)$ 。

思考? : \mathcal{D}_{VA} 的作用是?

\mathcal{D}_{VA} 是用来验证从 \mathcal{D}_{TR} 获得的 $\mathbf{h}(\cdot)$ 是否存在过拟合的问题。 $\mathbf{h}(\cdot)$ 需要在 \mathcal{D}_{VA} 上被验证，如果 $\mathbf{h}(\cdot)$ 严重过拟合，我们则需要根据 \mathcal{D}_{TR} 重新训练选择 $\mathbf{h}(\cdot)$ ，然后再次在 \mathcal{D}_{VA} 上验证。这个过程会反复进行，直到在 \mathcal{D}_{VA} 也能预测较好为止。在 \mathcal{D}_{TR} 和 \mathcal{D}_{VA} 的数据集大小选择上有一个平衡：对于较大的 \mathcal{D}_{TR} ，训练结果会更好，但是如果 \mathcal{D}_{VA} 更大，验证会更可靠 (噪音更小)。

2.5.1 如何划分数据

在训练、验证和测试集中划分数据时必须非常小心。测试集必须模拟真实的测试场景，即模拟的是在现实生活中的事物。例如，如果想训练一个电子邮件垃圾邮件过滤器，可以根据过去数据来训练一个系统来预测未来接收到的邮件是否为垃圾邮件。在这里，将训练集/测试集以时间为基准分开是非常重要的——这样就可以利用过去的数据预测未来。如果不存在时间分量，通常最好的是随机均匀地划分。绝对不要按字母或特征值进行拆分，这样会引入额外信息。

如果是时间序列数据，那么根据时间是一个好的方法。

一致随机，当且仅当数据是独立同分布的。

测试集误差 (或测试集损失) 应该近似于真实的泛化误差/损失。

2.6 总结

我们通过最小化在训练集上的损失函数来训练分类器:

$$\text{Learning : } \mathbf{h}^*(\cdot) = \arg \min_{\mathbf{h}(\cdot) \in \mathcal{H}} \frac{1}{|\mathcal{D}_{TR}|} \sum_{(x,y) \in \mathcal{D}_{TR}} \ell(x, y | \mathbf{h}(\cdot))$$

其中 \mathcal{H} 是假设空间 (即所有分类器 $\mathbf{h}(\cdot)$ 的集合)。换句话说，我们试图找到一个假设函数 \mathbf{h} ，使得它在过去/已知数据上运行良好。然后我们对分类器在测试集的损失进行评估:

$$\text{Evaluation : } \epsilon_{TE} = \frac{1}{|\mathcal{D}_{TE}|} \sum_{(x,y) \in \mathcal{D}_{TE}} \ell(x, y | \mathbf{h}^*(\cdot))$$

如果样本从相同的分布 \mathcal{P} 中独立抽取的, 则测试损失是**泛化损失**的无偏估计:

$$\textbf{Generaliazation} : \epsilon = \mathbb{E}_{(x,y) \sim \mathcal{P}} [\ell(x, y | \mathbf{h}^*(\cdot))]$$

思考: 为什么当 $|\mathcal{D}_{TE}| \rightarrow +\infty$ 时 $\epsilon_{TE} \rightarrow \epsilon$? 这是因为弱大数定理指出从数据中经验地得到的平均值收敛于其分布的平均值。

没有免费的午餐: 每一个机器学习算法都带有对 \mathcal{H} 的假设。这个假设是取决于数据的, 并因此对数据集/分布 \mathcal{P} 的假设编码。显然, 没有一个完美的 \mathcal{H} 可以预先解决所有数据的问题。

例子: 假设 $(\mathbf{x}_1, \mathbf{y}_1) = (1, 1)$, $(\mathbf{x}_2, \mathbf{y}_2) = (2, 2)$, $(\mathbf{x}_3, \mathbf{y}_3) = (3, 3)$, $(\mathbf{x}_4, \mathbf{y}_4) = (4, 4)$, $(\mathbf{x}_5, \mathbf{y}_5) = (5, 5)$ 。

提问: 当 $\mathbf{x} = 2.5$ 时 \mathbf{y} 的值会是多少? 没有对 \mathcal{H} 的假设是不可能知道答案的。为保证模型的鲁棒性, 机器学习算法最常见的假设是拟合的函数的局部光滑性。

3

k 近邻和维数灾难

3.1 k 近邻算法

3.1.1 概念介绍

k 近邻算法 (k -Nearest Neighbor Method, 简称 k NN) 的基本假设是相似的输入可以得到相似的输出。简单来说, 对于一个测试输入 \mathbf{x} , 在训练数据集中找到与 \mathbf{x} 最邻近的 k 个实例 (也就是 k 近邻所指的 k 个邻居), 将这 k 个实例出现最多的标签, 设为 \mathbf{x} 的标签。

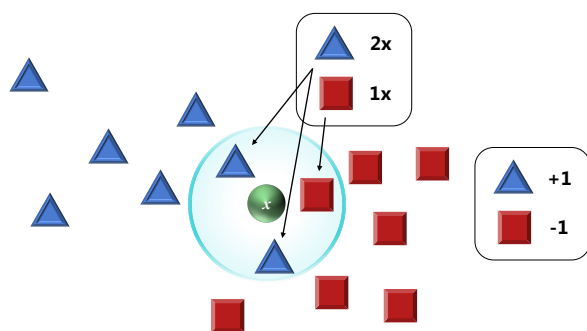


图 3-1 邻居数量设置为 $k = 3$ 的二分类 k 近邻分类器示例。中间的绿色数据点是测试样本, 因为 k 近邻中蓝色点个数多于红色点数, 所以判断该点属于蓝色点类别 ($+1$)

3.1.2 正式定义

给定一组测试点: \mathbf{x} , 设最靠近 \mathbf{x} 的 k 个邻居组成的集合为 S_x 。 S_x 的正式定义是 $S_x \subseteq D$ s.t. $|S_x| = k$ 且 $\forall (x', y') \in D \setminus S_x$,

$$\text{dist}(\mathbf{x}, \mathbf{x}') \geq \max_{(\mathbf{x}'', y'') \in S_x} \text{dist}(\mathbf{x}, \mathbf{x}''),$$

换言之, 对于在 D 中且不在 S_x 中的每一个点, 它和 \mathbf{x} 的距离大于等于它和 S_x 中其他点的最大距离。

我们定义分类器 $h(\cdot)$, 它将返回 S_x 中最常见类别的标签:

$$h(\mathbf{x}) = \text{mode}(\{y'' : (\mathbf{x}'', y'') \in S_x\}),$$

其中 $\text{mode}(\cdot)$ 意思是选择出现最多的标签。这样的分类器叫做 k 近邻分类器, 也称为 k NN 分类器。

3.1.3 k 值的选取

k 值的选取和数据本身的特征有关。一般而言, k 值的增加可以减小噪声对分类的影响, 但是使得分类的边界不再明显。特殊地, 当 $k = 1$ 时, 选取离测试点最近的邻居标签作为测试点的标签。当 $k = n$ 时 (n 为测试点的个数), 选取所有数据点中频数最高的标签。

3.1.4 距离函数的选择

k NN 分类器从根本上依赖于距离度量的标准。度量的标准越是能反映标签的相似性, 分类的效果越好。最常见的选择是闵可夫斯基距离:

$$\text{dist}(\mathbf{x}, \mathbf{z}) = \left(\sum_{r=1}^d |x_r - z_r|^p \right)^{1/p}.$$

特殊地, 当 $p = 1$, $\text{dist}(\mathbf{x}, \mathbf{z})$ 是曼哈顿距离。

当 $p = 2$, $\text{dist}(\mathbf{x}, \mathbf{z})$ 是欧几里得距离。

当 $p \rightarrow \infty$, $\text{dist}(\mathbf{x}, \mathbf{z})$ 是切比雪夫距离。

3.2 最近邻分类器

最近邻分类器即 $k = 1$ 时的 k 近邻分类器, 也叫做 1-NN 分类器。

3.2.1 贝叶斯最优分类器

我们先了解一下贝叶斯分类器（以贝叶斯定理为基础的分类算法的总称）。对于一个贝叶斯分类器来说，假设我们已知所有条件概率 $P(y|\mathbf{x})$ (虽然在大多数情况下不是这样)，我们可以简单预测出最有可能的标签。

$$\text{贝叶斯最优分类器预测: } y^* = h_{\text{opt}}(\mathbf{x}) = \underset{y}{\operatorname{argmax}} P(y|\mathbf{x})$$

虽然贝叶斯分类器很好，但是它也会犯错。比如当样本没有可能性最大达到 1 的标签时，贝叶斯分类器总会犯错。我们可以准确计算出这种错误发生的概率：

$$\epsilon_{\text{BayesOpt}} = 1 - P(h_{\text{opt}}(\mathbf{x})|\mathbf{x}) = 1 - P(y^*|\mathbf{x})$$

举例来说，假设一封邮件 \mathbf{x} 可能被归类为垃圾邮件 (+1) 或非垃圾邮件 (-1)。这两者的条件概率分别为：

$$P(+1|\mathbf{x}) = 0.8$$

$$P(-1|\mathbf{x}) = 0.2$$

在这种情况下贝叶斯最优分类器会预测最有可能的标签为 $y^* = +1$ ，而它的错误率是 $\epsilon_{\text{BayesOpt}} = 0.2$ 。

虽然贝叶斯分类器无法应用到实践中，为什么它依然非常有价值？原因在于它给出了理论上最低错误率。在相同的特征表示下，没有分类器能够获得更低的错误率。我们将利用这一性质来分析 k 近邻分类器的错误率。

3.2.2 最近邻分类器的收敛性

我们可以证明：当 $n \rightarrow \infty$ ，最近邻分类器的错误率不大于贝叶斯最优分类器错误率的两倍。

设 \mathbf{x}_{NN} 是距离我们测试点 \mathbf{x}_t 最近的邻居。当 $n \rightarrow \infty$ ， $\text{dist}(\mathbf{x}_{\text{NN}}, \mathbf{x}_t) \rightarrow 0$ ，即 $\mathbf{x}_{\text{NN}} \rightarrow \mathbf{x}_t$ 。（这意味着最近的邻居等同于 \mathbf{x}_t ），你得到了 \mathbf{x}_{NN} 的标签，那么 \mathbf{x}_t 的标签和它不一样的概率为：

$$\begin{aligned} \epsilon_{\text{NN}} &= P(y^*|\mathbf{x}_t)(1 - P(y^*|\mathbf{x}_{\text{NN}})) + P(y^*|\mathbf{x}_{\text{NN}})(1 - P(y^*|\mathbf{x}_t)) \\ &\leq (1 - P(y^*|\mathbf{x}_{\text{NN}})) + (1 - P(y^*|\mathbf{x}_t)) = 2(1 - P(y^*|\mathbf{x}_t)) = 2\epsilon_{\text{BayesOpt}}, \end{aligned}$$

好消息是：当 $n \rightarrow \infty$ ，最近邻分类器的预测结果错误率不会超过理论最好的分类器的两倍。

坏消息是：维数灾难是我们不可避免的问题。

3.3 维数灾难

3.3.1 点和点之间的距离

k NN 分类器假定相近的点具有相似的标签。遗憾的是，在高维空间，从概率分布中采样的点，往往永远不会彼此靠近。我们可以在一个简单的例子中阐释这一点。在一个单位立方体中随机均匀地画点，然后研究对于立方体中的一个测试点，它的 k 个最近的邻居会占用多少空间。

记单位立方体为 $[0, 1]^d$ 。 n 个训练数据在这个立方体中均匀分布，即 $\forall i, x_i \in [0, 1]^d$ 。我们考虑 $k=10$ 的情况，即距离测试点最近的 10 个邻居。

设 ℓ 是包含测试点的 k 个最近邻居的最小超立方体的边长。显然， $\ell^d \approx \frac{k}{n}$ ，即 $\ell \approx (\frac{k}{n})^{1/d}$ 。假如取 $n = 1000$ ，我们可以计算出 ℓ 的大小：

当 $d=2$, $\ell=0.1$;

当 $d=10$, $\ell=0.63$;

当 $d=100$, $\ell=0.955$;

当 $d=1000$, $\ell=0.9954$ 。

可以发现，当 $d \gg 0$ 几乎需要搜索整个空间才能发现最靠近测试点的 10 个邻居。这打破了 kNN 的基本假设，因为距离测试点最近的 k 个邻居并不比训练集中的其他点更加接近。既然这 k 个邻居并不比其他点更相似于测试点，那么它们就没有理由和测试点具有相同的标签了。

有人可能认为一种补救措施是增加训练集的数量，也就是增大 n 的值，直到测试点的邻居真正接近于测试点。然而，约束 $\ell = 0.1$ ，可以注意到 $n = \frac{k}{\ell^d} = k \cdot 10^d$ 是指数级增长。当 $d > 100$ 时，需要的数据点数量就比整个宇宙的电子都要多了。

3.3.2 点和超平面之间的距离

可以发现，两个随机点之间的距离随着它们维数的增加而剧烈增长。考虑图3-2。图中有两个蓝点和一张红色的超平面。左边的图表示的是二维场景，右边的图表示的是三维场景。当维数 $d = 2$ 时，两点之间的距离为 $\sqrt{\Delta x^2 + \Delta y^2}$ 。当加入第三个维数时，距离扩展为 $\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}$ 这再次证明了成对距离在高维会增长。

另一方面，在第三个维度加入后，它们距离红色超平面的距离并没有增加。原因在于超平面的法线和新的维度正交。这是一个至关重要的结果。在 d 维空间里， $d - 1$ 维和任意给定的超平面的法线正交。点在 $d - 1$ 维空间里的运动不会增加和减少它们和超平面之间的距离——点仅仅是和超平面保持着相同距离进行平移。由于在高维空间下，

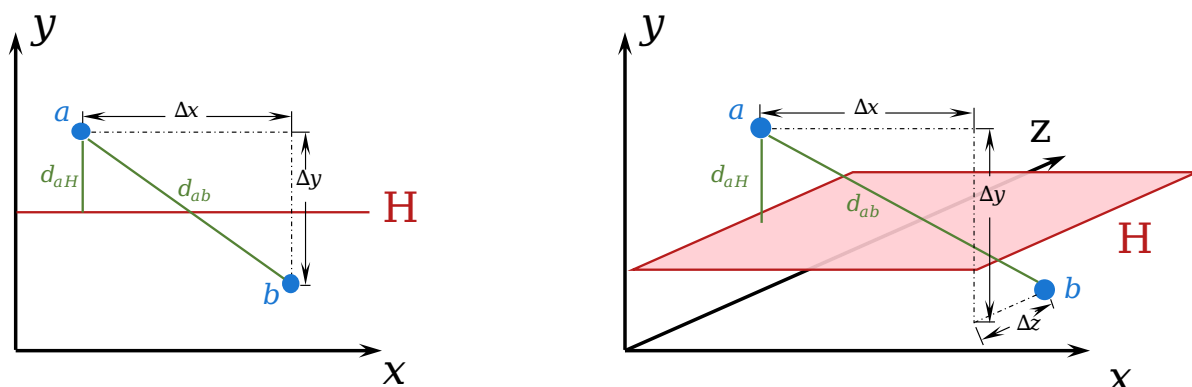


图 3-2 维数灾难对点与点之间的距离 (d_{ab}) 和点与超平面之间的距离 (d_{aH}) 有不同的效果

成对点之间的距离非常大，点和超平面之间的距离相对就显得小了。机器学习算法和这一点密切相关。在接下来的内容中可以看到，许多分类器（比如感知机和支持向量机）在不同类别之间放置分类超平面。

维数灾难的一个结果是大多数数据点分布于超平面附近，这使得我们可以给输入一个扰动，以便改变分类的结果。近期这种现象随着对抗样本的出现而广为人知，这种现象常常被错误归因为神经网络的复杂性。

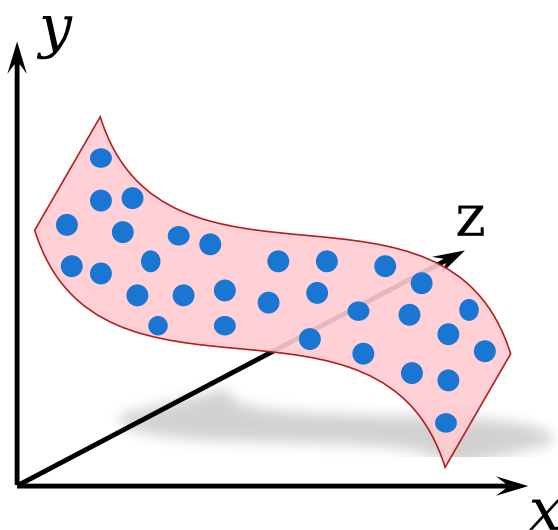


图 3-3 数据集取自嵌入在三维空间的二维流形。
蓝色的点属于粉色的平面区域，而粉色的平面则嵌入在三维空间。

3.3.3 低维结构的数据流形

上述结论来看，维数灾难的一个直接解决方案就是降低数据的维度，可能有人会担心这样会导致数据特征的丢失。然而情况可能不是这样的，真实数据可能分布在一个低维子空间或子流形上。比如，自然图像。数据的真实维数可能比它所在环境空间的维数要小得多。图3-3所表示的数据集取自嵌入在三维空间的一个二维流形。人脸就是低维数据集的典型例子。虽然一张人脸图片可能有 18M 个像素且千差万别，一个人却可以用少于 50 个特征属性 (男/女，金发/黑发.....) 来表述和识别这张人脸。

3.4 k -均值算法

k -均值算法（英文： k -means clustering）源于信号处理中的一种向量量化方法，现在则更多地作为一种聚类分析方法广泛应用于数据挖掘领域。 k -均值聚类的目的是：把 n 个点（可以是样本的一次观察或一个实例）划分到 k 个聚类中，使得每个点都属于离它最近的均值（此即聚类中心）对应的聚类，以之作为聚类的标准。

需要注意区分 k 近邻算法和 k -均值算法。 k 近邻是有标记的监督学习， k -均值则是无监督聚类算法。

4

概率估计与朴素贝叶斯

4.1 概率分布

概率分布 (probability distribution) 用来描述随机变量或一簇随机变量在一系列状态取值的可能性大小。我们描述概率分布有离散和连续两种不同的方式。

4.1.1 离散型变量和概率质量函数

离散型变量的概率分布采用概率质量函数进行描述，我们通常用大写字母 P 来表示概率质量函数。在描述随机变量时，我们不再利用不同的函数名称来区分不同的概率质量函数，而是认为不同的变量一般都有不同的概率质量函数。例如 $P(x)$ 和 $P(y)$ 所表示的概率分布是不同的，因为两个不同变量的取值状态一般是不同的。

概率质量函数将随机变量可以取到的值映射到随机变量取得该状态的概率，且概率质量函数可以同时作用于多个随机变量，这种作用于多个随机变量的概率分布被称为联合概率分布 (joint probability distribution)。例如 $P(\mathbf{x} = x, \mathbf{y} = y)$ 表示 $\mathbf{x} = x$ 和 $\mathbf{y} = y$ 同时发生的概率，简写为 $P(x, y)$ 。对于离散随机变量的概率质量函数 P ，需要满足以下几个条件：

- (1) P 的定义域必须是 \mathbf{x} 所有可能状态的集合。
- (2) $\forall x \in \mathbf{x}, 0 \leq P(x) \leq 1$ 。不可能的取值概率为 0，并且不存在比这个概率更低的状态。同样类似的，能够确保一定发生的事件概率为 1，并且不存在比这个概率更高的状态。

(3) $\sum_{x \in \mathbf{x}} P(x) = 1$ 。随机变量具有归一性，如果没有这条性质，会出现事件发生的概率大于 1 的情况，这是错误的。

例如，考虑一个离散型随机变量 x 有 k 个不同的状态，我们可以假设 x 是均匀分布 (uniform distribution) 的，它的概率质量函数表示为：

$$P(\mathbf{x} = x_i) = \frac{1}{k}$$

对于所有的 i 都成立。我们可以看出这满足概率质量函数的条件，因为 k 是一个正整数，所以 $\frac{1}{k}$ 是正的。同时我们也发现：

$$\sum_i P(\mathbf{x} = x_i) = \sum_i \frac{1}{k} = \frac{k}{k} = 1$$

因此也满足归一化条件。

4.1.2 连续型变量和概率密度函数

在研究连续型变量时，我们使用概率密度函数来描述它的概率分布。如果一个函数 p 是概率密度函数，那么它必须满足下面几个条件：

- (1) p 的定义域必须是 x 所有可能状态的集合。
- (2) $\forall x \in \mathbf{x}, p(x) \geq 0$ 。注意，我们并不要求 $p(x) \leq 1$ 。
- (3) $\int_0^1 p(x) dx = 1$ 概率密度函数 $p(x)$ 并没有要求直接对特定的状态给出概率，相对的，它给出了落在区间为 dx 的概率大小 $p(x)dx$ 。

我们可以对概率密度函数求积分获得点集的真实概率质量。例如在单一随机变量的例子里， x 落在区间 $[a, b]$ 中的概率为 $\int_a^b p(x) dx$ 。

为了给出一个连续型随机变量的例子，我们考虑一个简单的实例，实数区间上的均匀分布。我们用符号 $u(x; a, b)$ 表示随机变量 x 在区间 $[a, b]$ 上的均匀分布。对于这个分布函数而言，我们观察到在区间 $[a, b]$ 内，所有变量可能取值的概率都是 $\frac{1}{b-a}$ ，对于区间外的所有变量，我们定义其取值的概率大小为 0，即不可能取到对应的状态。此外 $\int_a^b \frac{1}{b-a} dx = 1$ ，满足归一化条件。通常记 $x \sim U(a, b)$ 表示 x 在 $[a, b]$ 上均匀分布。

4.1.3 边缘概率

有时我们知道了一组变量的联合概率分布，但是我们需要使用其中一个随机变量单独的概率分布。这种定义在子集上的概率分布被称为边缘概率分布 (marginal probability distribution)。对于离散型随机变量我们使用求和法来计算 $P(x)$ ：

$$\forall x \in \mathbf{x}, P(\mathbf{x} = x) = \sum_y P(\mathbf{x} = x, \mathbf{y} = y)$$

“边缘概率”的直观含义就是当 $P(x, y)$ 的每个值被写在由每行表示不同的 x 值，每列表示不同的 y 值形成的网格中时，对网格中的每行求和是自然的事情，然后将求和完成后的结果 $P(x)$ 写在每行右边的纸边缘处。

对于连续型变量，我们需要用积分替代求和：

$$p(x) = \int p(x, y) dy$$

4.1.4 条件概率

在许多场景中，我们需要讨论某个事件在给定其他事件发生时出现的概率，我们定义这种概率为条件概率。我们将给定 $\mathbf{x} = x$ ， $\mathbf{y} = y$ 发生的条件概念记为：

$$P(\mathbf{y} = y | \mathbf{x} = x) = \frac{P(\mathbf{y} = y, \mathbf{x} = x)}{P(\mathbf{x} = x)}$$

条件概率只在 $P(\mathbf{x} = x) > 0$ 时有定义。我们不能计算给定在永远不会发生概率事件上的条件概率，这也是符合直觉的，因为如果永远不会发生，那么这个前提成立本身就是矛盾的。

这里需要注意的是，不要把条件概率和计算当采用某个动作后会发生什么混淆。假定某个人说当地的方言，那么他是当地人的概率是非常高的，但是如果选择一个人，那么他说当地方言的概率并不会变高。

4.2 最大似然估计 (MLE, Maximum Likelihood Estimation)

最大似然估计提供了一种给定观察数据来评估模型参数的方法。由最大似然估计的定义可知，根据数据分布，找出一个模型参数 θ 使得事件发生的概率尽可能大。以抛硬币为例，极大似然估计法希望找到一个 θ ，使结果出现 X 的概率最大（已知模型和数据，推测参数）。

假设 m 个样本的数据集 $X = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_m\}$ 是独立同分布于 $p_{data}(\mathbf{x})$ 的数据集， X_c 表示数据集中所有分类为 c 的样本集合，由于这些样本是独立同分布的，因此参数 θ_c 对于数据集 X_c 的最大似然估计为如下：

$$\begin{aligned} \hat{\theta}_{MLE} &= \operatorname{argmax}_{\theta_c} P(X_c | \theta_c) \\ &= \operatorname{argmax}_{\theta_c} \prod_{i=1}^m P(\mathbf{x}_i | \theta_c) \end{aligned} \tag{4.1}$$

$P(X_c | \theta_c)$ 为最大似然函数表达式, MLE 的目标为找到参数 θ_c 使最大似然函数取到最大值。最大似然估计中由于存在连乘操作, 通常使用对数似然进行计算, 对数最大似然估计表达式为:

$$\begin{aligned} \log(\hat{\theta}_{MLE}) &= \underset{\theta_c}{\operatorname{argmax}} \log(P(X_c | \theta_c)) \\ &= \underset{\theta_c}{\operatorname{argmax}} \sum_{i=1}^m \log P(\mathbf{x}_i | \theta_c) \end{aligned} \quad (4.2)$$

假设似然函数满足联系可导的条件, 则最大似然估计就是如下方程的解:

$$\nabla_{\theta} \log(\hat{\theta}_{MLE}) = \sum_{i=1}^m \nabla_{\theta_c} \log P(\mathbf{x}_i | \theta_c) = 0 \quad (4.3)$$

方程的解只是一个估计值, 只有在样本数趋于无限多的时候, 它才会接近于真实值。假设样本服从正态分布 $N(\mu, \sigma^2)$, 则似然函数为:

$$P(\mu, \sigma^2) = \prod_{i=1}^N \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x_i - \mu)^2}{2\sigma^2}} = (2\pi\sigma^2)^{-\frac{n}{2}} e^{-\frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2}$$

对 $P(\mu, \sigma^2)$ 自然函数取对数得到:

$$\ln P(\mu, \sigma^2) = -\frac{n}{2} \ln(2\pi) - \frac{n}{2} \ln(\sigma^2) - \frac{1}{2\sigma^2} \sum_{i=1}^n (x_i - \mu)^2$$

对上面的表达式求偏导, 令偏导数为 0 得到方程组:

$$\begin{cases} \frac{\partial \ln P(\mu, \sigma^2)}{\partial \mu} = \frac{1}{\sigma^2} \sum_{i=1}^n (x_i - \mu) = 0 \\ \frac{\partial \ln P(\mu, \sigma^2)}{\partial \sigma^2} = -\frac{n}{2\sigma^2} + \frac{1}{2\sigma^4} \sum_{i=1}^n (x_i - \mu)^2 = 0 \end{cases}$$

联合解得:

$$\begin{cases} \mu^* = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i \\ \sigma^{*2} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2 \end{cases}$$

由上述求解过程可以知道, 通过最大似然估计得到的正态分布均值就是样本的均值, 方差就是样本的方差, 这和我们的直觉符合, 在离散的条件也可以使用类似的方法进行参数估计。

4.3 最大后验估计 (MAP, Maximum A Posteriori Probability Estimate)

在实际任务中，采集大量的样本数据往往代价很高，这时可以引入先验分布来解决这个问题。以抛硬币问题举例，因为硬币一般来说是匀称的，所以抛硬币两面朝上的概率一样大的可能性比较高，即 $\theta = 0.5$ 的可能性比较大，而 $\theta = 0.3$ 或 $\theta = 0.8$ 的可能性比较小。在这种情况下，最大后验估计 (MAP, Maximum A Posteriori) 求得的 θ 不但要使得似然函数最大， θ 自己出现的先验概率也要大，根据上述表达，MAP 定义如下：

$$\hat{\theta}_{MAP} = \arg \max_{\theta} P(\theta|X) \quad (4.4)$$

根据贝叶斯公式，上式可展开为：

$$\hat{\theta}_{MAP} = \arg \max_{\theta} P(\theta|X) = \arg \max_{\theta} \frac{P(X|\theta)P(\theta)}{P(X)}$$

根据表达式可知，分母中的 $P(X)$ 只与数据集的数据样本相关，因此原问题等价于求解如下表达式：

$$\hat{\theta}_{MAP} = \arg \max_{\theta} P(\theta|X) = \arg \max_{\theta} P(X|\theta)P(\theta) \quad (4.5)$$

以抛硬币为例，数据是由多次独立同分布实验形成的伯努利随机变量，假设参数 θ 有一个先验估计，它服从 Beta 分布，即：

$$P(\theta) = \text{Beta}(\beta_0, \beta_1) = \frac{\theta^{\beta_1-1}(1-\theta)^{\beta_0-1}}{B(\beta_0, \beta_1)} \quad (4.6)$$

这里的 β_0 和 β_1 是参数，我们必须事先指定这两个参数的值才能确定特定的 β_0 。

$$\begin{aligned} \hat{\theta} &= \arg \max_{\theta} P(D|\theta)P(\theta) \\ &= \arg \max_{\theta} \theta^{\alpha_1}(1-\theta)^{\alpha_0} \frac{\theta^{\beta_1-1}(1-\theta)^{\beta_0-1}}{B(\beta_0, \beta_1)} \\ &= \arg \max_{\theta} \frac{\theta^{\alpha_1+\beta_1-1}(1-\theta)^{\alpha_0+\beta_0-1}}{B(\beta_0, \beta_1)} \\ &= \arg \max_{\theta} \theta^{\alpha_1+\beta_1-1}(1-\theta)^{\alpha_0+\beta_0-1} \end{aligned} \quad (4.7)$$

因为 $B(\beta_0, \beta_1)$ 独立于 θ ，所以最后一行可以略去分母。

对上述表达式求对数，求导后令导数等于 0，易得：

$$\hat{\theta}_{MAP} = \arg \max_{\theta} P(D|\theta)P(\theta) = \frac{(\alpha_1 + \beta_1 - 1)}{(\alpha_1 + \beta_1 - 1) + (\alpha_0 + \beta_0 - 1)} \quad (4.8)$$

4.4 朴素贝叶斯分类器

朴素贝叶斯法 (Naive Bayes) 是基于贝叶斯定理与特征条件强 (朴素) 独立假设的分类方法。对于给定的训练数据集, 首先基于特征条件独立假设学习输入/输出的联合概率分布; 然后基于此模型, 对给定的输入 \mathbf{x} , 利用贝叶斯定理求出后验概率最大的输出 y 。

假设输入空间 $\mathcal{X} \subseteq \mathbb{R}^d$ 为 d 维向量的集合, 输出空间 $\mathcal{Y} \subseteq \mathbb{R}$ 为 1 维向量的集合, 训练数据集为 $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, 由某个联合概率 $P(X, Y)$ 独立同分布采样得到。因为所有的数据点都服从独立同分布, 所以有:

$$P(D) = P((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) = \prod_{\alpha=1}^n P(\mathbf{x}_\alpha, y_\alpha) \quad (4.9)$$

假设数据样本数量充足, 则可以使用类似抛硬币的例子估计联合分布 $P(X, Y)$ 。我们将联合分布 $P(X, Y)$ 中所有特征组合得到的数据样本数量进行统计, 就可以通过频率计算来估计一个特定特征组合出现的概率:

$$\hat{P}(\mathbf{x}, y) = \frac{\sum_{i=1}^n I(\mathbf{x}_i = \mathbf{x} \wedge y_i = y)}{n}, \quad (4.10)$$

其中 $I()$ 为指示函数, 如果 $\mathbf{x}_i = \mathbf{x}$ 且 $y_i = y$ 那么 $I(\mathbf{x}_i = \mathbf{x} \wedge y_i = y)$ 为 1, 否则为 0。

如果只是想通过特征 \mathbf{x} 来预测标签 y 的概率, 我们可以直接估计 $P(y|\mathbf{x})$ 而不是 $P(\mathbf{x}, y)$ 。这里可以用贝叶斯最优分类器 (Bayes Optimal Classifier) 的方法。对给定的输入 \mathbf{x} , 先计算特定后验概率 $\hat{P}(y|\mathbf{x})$, 再将后验概率最大的对应标签作为输入 \mathbf{x} 的输出分类标签。那么我们该如何计算 $\hat{P}(y|\mathbf{x})$ 呢? 之前我们已经通过极大似然估计 (Maximum Likelihood Estimation) 得到 $\hat{P}(y) = \frac{\sum_{i=1}^n I(y_i=y)}{n}$ 。同理, 可以得到 $\hat{P}(\mathbf{x}) = \frac{\sum_{i=1}^n I(\mathbf{x}_i=\mathbf{x})}{n}$ 和 $\hat{P}(y, \mathbf{x}) = \frac{\sum_{i=1}^n I(\mathbf{x}_i=\mathbf{x} \wedge y_i=y)}{n}$, 将这两者用贝叶斯公式结合计算可以得到:

$$\hat{P}(y|\mathbf{x}) = \frac{\hat{P}(y, \mathbf{x})}{\hat{P}(\mathbf{x})} = \frac{\sum_{i=1}^n I(\mathbf{x}_i = \mathbf{x} \wedge y_i = y)}{\sum_{i=1}^n I(\mathbf{x}_i = \mathbf{x})}. \quad (4.11)$$

如图4-1所示, 可以将极大似然估计 (MLE) 表示为集合之间的关系, 并得到 $\hat{P}(y|\mathbf{x})$ 的估计值:

$$\hat{P}(y|\mathbf{x}) = \frac{|C|}{|B|}. \quad (4.12)$$

但用极大似然估计会出现一个问题, 对于给定数据量不足, 在高维特征空间或者特征 \mathbf{x} 是连续的情况下, 会使得 $|B| \rightarrow 0$, 导致无法根据定义进行计算。

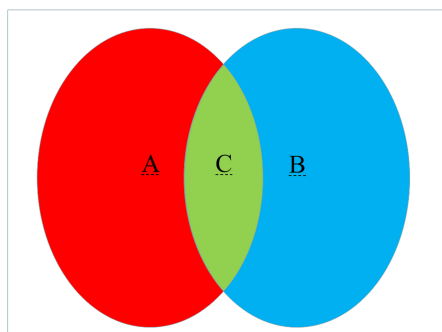


图 4-1 MLE 的韦恩图表示。

Figure 4-1 The Venn diagram illustrates the MLE method.

4.5 朴素贝叶斯法

用极大似然估计可能会出现无法计算的情况，可以通过朴素贝叶斯法解决上一节的问题。

对于上一节的表达式 (4.11)，利用贝叶斯公式进行变换得到：

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \quad (4.13)$$

估计 $P(y)$ 是容易的，只需要计算每一个标签类在总数据集中出现的比例。比如，如果 Y 取的是离散的二分类值，那么估计 $P(y)$ 的值就与抛硬币问题是一样的。对于多分类问题，我们只需计算对应类别出现的频率，这就相当于是该类别标签频率的统计：

$$P(y = c) = \frac{\sum_{i=1}^n I(y_i = c)}{n} = \hat{\pi}_c. \quad (4.14)$$

而直接估计 $P(\mathbf{x}|y)$ 比较难，为了降低估计的难度并比较准确的进行估计，需要对其做一定的假设，称为朴素贝叶斯假设 (Naive Bayes Assumption)，对应的定义式为：

$$P(\mathbf{x}|y) = \prod_{\alpha=1}^d P(x_{\alpha}|y), \quad (4.15)$$

其中 $x_{\alpha} = [\mathbf{x}]_{\alpha}$ 是样本 \mathbf{x} 第 α 特征维度上的值。

朴素贝叶斯假设是一个强（独立）假设，其表达的含义是所有用于分类的特征信息在给定条件下都是互相独立的。这一假设使朴素贝叶斯法在概率和分布估计上变得十分简单，但也会牺牲一定的分类正确率。

贝叶斯分类器的一个实际应用场景是进行垃圾邮件过滤。此时，概率模型需要训练数据的特征就是邮件内容本身，而邮件的分类标签只有两类，即垃圾邮件和非垃圾邮

件。在这个特定的条件下朴素贝叶斯假设表达的具体含义是：邮件内容中的每个单词在已知邮件是否为垃圾邮件的情况下都是条件独立的。显然这个假设直观理解是不成立的，因为在写邮件时，句子中的单词并不是从词库随机挑选组成的，而是根据上下文的关联性来组织的。尽管朴素贝叶斯假设与实际场景是违背的，但是使用朴素贝叶斯法进行计算时，训练得到的概率模型给出的分类结果却有着相当不错的效果。

如图4-2所示，这里用图示来表达朴素贝叶斯法的思想。图例中，特征有两个维度，类别标签也只有两个。根据图例中的数据样本，单独使用第一个或者第二个特征都无法将两种样本完全分开，而朴素贝叶斯法可以容易地将两者区分。

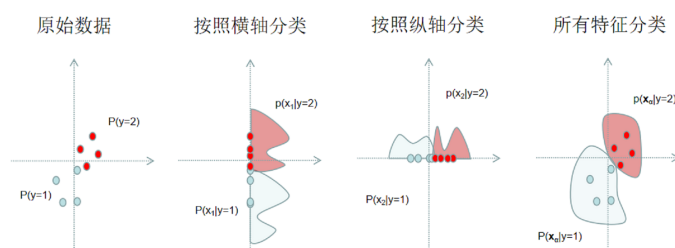


图 4-2 朴素贝叶斯法思想的图示。我们可以在每个输入特征维度上独立地估计 $P(x_\alpha|y)$ 的值（如中间两幅图所示），通过朴素贝叶斯假设 $P(\mathbf{x}|y) = \prod_\alpha P(x_\alpha|y)$ 可以进一步估计整体数据的分布（如最右边图所示）。

Figure 4-2 Illustration behind the Naive Bayes algorithm.

我们假定朴素贝叶斯假设成立，由此贝叶斯分类器可以由下式来重新定义：

$$\begin{aligned}
 \mathbf{h}(\mathbf{x}) &= \operatorname{argmax}_y P(y|\mathbf{x}) \\
 &= \operatorname{argmax}_y \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \\
 &= \operatorname{argmax}_y P(\mathbf{x}|y)P(y) && (P(\mathbf{x}) \text{ 与 } y \text{ 无关}) \\
 &= \operatorname{argmax}_y \prod_{\alpha=1}^d P(x_\alpha|y)P(y) && (\text{由朴素贝叶斯假设可得}) \\
 &= \operatorname{argmax}_y \sum_{\alpha=1}^d \log(P(x_\alpha|y)) + \log(P(y)) && (\text{因为 } \log \text{ 是一个单调函数}).
 \end{aligned}$$

因为只需要考虑一个特征维度上的问题，所以估计 $\log(P(x_\alpha|y))$ 的值是相对于直接估计 $\log(P(\mathbf{x}_\alpha|y))$ 是容易的，并且 $P(y)$ 的估计不受该假设影响。接下来我们引入几个实例介绍如何利用朴素贝叶斯法进行参数估计。

4.6 $P([\mathbf{x}]_\alpha|y)$ 参数估计

用上一节介绍的朴素贝叶斯假设可以很方便地得到 $P(y|\mathbf{x})$ ，下面介绍三个典型实例，运用朴素贝叶斯法对参数进行估计。

4.6.1 实例 1: 离散特征

本实例中所有的数据样本特征均为离散特征：

$$[\mathbf{x}]_\alpha \in \{f_1, f_2, \dots, f_{K_\alpha}\}.$$

每一个特征维度 α 都在 K_α 个离散类型中取值。（注意：二类型特征只是这种情况的一个特例，即 $K_\alpha = 2$ 。）满足这一情况的现实生活的例子有很多，比如医院体检数据中某一特征可以是性别，有男性和女性两种离散类型；另一特征可以是婚姻状况，有未婚、已婚和离婚三种离散类型，类别标签可以是非常健康、良好和不健康等。

在这种特征情况下， $P(x_\alpha | y)$ 的估计很容易确定：

$$P(x_\alpha = j | y = c) = [\theta_{jc}]_\alpha \quad (4.16)$$

$$\sum_{j=1}^{K_\alpha} [\theta_{jc}]_\alpha = 1, \quad (4.17)$$

其中 $[\theta_{jc}]_\alpha$ 是特征维度 α 在分类标签为 c 的情况下取值为 j 的概率。

由此可以得到 $[\theta_{jc}]_\alpha$ 参数的估计：

$$[\hat{\theta}_{jc}]_\alpha = \frac{\sum_{i=1}^n I(y_i = c) I(x_{i\alpha} = j) + l}{\sum_{i=1}^n I(y_i = c) + l K_\alpha}, \quad (4.18)$$

其中 $x_{i\alpha} = [\mathbf{x}_i]_\alpha$ ， l 是平滑参数。如果取 $l = 0$ 那么式 (4.18) 即为 MLE，如果 $l > 0$ 那么式 (4.18) 变成 MAP (Maximum a posteriori estimation)，如果取 $l = +1$ 那么式 (4.18) 为拉普拉斯平滑 (Laplace Smoothing)。

如果忽略平滑参数 l ，这就等价于：

$$\frac{\text{数据中分类标签为 } c \text{ 且特征维度 } \alpha \text{ 取值为 } j \text{ 的个数}}{\text{数据中分类标签为 } c \text{ 的个数}}. \quad (4.19)$$

本质上，这个离散特征的模型和一枚带有特征和类别标签的特殊硬币比较类似。对每一个可能的分类标签（例如前面提到的体检类别标签：非常健康），它都有 d 个骰子，输入特征的每个维度都对应一个骰子。数据生成的时候，按照先后次序投掷每个骰子，并将得到的值作为相应特征维度上的特征值。由此，如果有 C 个可能的标签，那么我们

就需要有 $d \times C$ 个骰子，每个骰子（即输入特征的每个维度） α 都有 K_α 个可能的面（即取值）。当然，这里所描述的并非真正的数据对生成方式，这只是基于朴素贝叶斯法的假设模型。

所以，我们可以得到预测方法：

$$\operatorname{argmax}_y P(y = c \mid \mathbf{x}) \propto \operatorname{argmax}_y \hat{\pi}_c \prod_{\alpha=1}^d [\hat{\theta}_{jc}]_\alpha. \quad (4.20)$$

4.6.2 实例 2: 多项式特征

若特征值不是代表着简单的离散类型（如男性、女性），而是一种计数形式的特殊类型，那么朴素贝叶斯法又该如何应用呢？在文本分类问题中，特征维度 $x_\alpha = j$ 表示该文本 \mathbf{x} 中的第 α 个单词在词典中出现过 j 次。回到垃圾邮件分类的问题中，假设邮件文本中的第 α 个单词是“spam”。如果 $x_\alpha = 10$ 那么这封邮件可能是垃圾邮件，因为“spam”这个词在邮件中出现了 10 次；如果一封邮件它的 $x'_\alpha = 20$ ，相比前者，这一封更可能是垃圾邮件，但是离散特征模型不能保证这一性质。所以我们需要一个适用于文本特征分类的朴素贝叶斯模型，这正是将要介绍的多项式分布模型。

该实例中的数据特征表现为多项式特征：

$$x_\alpha \in \{0, 1, 2, \dots, m\} \text{ 且 } m = \sum_{\alpha=1}^d x_\alpha. \quad (4.21)$$

此时每一个特征维度 α 代表着一个计数， m 是整个文本中单词的个数（文本中可能有重复单词）， d 表示词典的大小，即词典中单词的个数（词典中没有重复单词）。用多项式分布来表征 $P(\mathbf{x} \mid y)$ ：

$$P(\mathbf{x} \mid m, y = c) = \frac{m!}{x_1! \cdot x_2! \cdot \dots \cdot x_d!} \prod_{\alpha=1}^d (\theta_{\alpha c})^{x_\alpha}, \quad (4.22)$$

其中 $\theta_{\alpha c}$ 是选择特征维度 x_α 的概率，且 $\sum_{\alpha=1}^d \theta_{\alpha c} = 1$ 。所以我们可以通过这一模型生成一封垃圾邮件。根据分布 $P(\mathbf{x} \mid y = \text{spam})$ ，从大小为 d 的词典中独立随机地取出 m 个单词，从而组成分类类别为 $y = \text{spam}$ 的文本 \mathbf{x} ，即一封垃圾邮件。

由此可以得到 $\hat{\theta}_{\alpha c}$ 参数的估计：

$$\hat{\theta}_{\alpha c} = \frac{\sum_{i=1}^n I(y_i = c) x_{i\alpha} + l}{\sum_{i=1}^n I(y_i = c) m_i + l \cdot d}, \quad (4.23)$$

其中 $m_i = \sum_{\beta=1}^d x_{i\beta}$ 表示文本 i 的总单词个数。

若忽略平滑参数 l ，那么上式中分子表示分类类别为 c 的文本中含有单词 α 的个数，分母表示所有分类类别为 c 的文本中所有单词的总个数。若以垃圾邮件为例，那么可以表述为：

$$\frac{\text{所有垃圾邮件中含有单词}\alpha\text{的个数}}{\text{所有垃圾邮件的单词总数}}. \quad (4.24)$$

所以，我们可以得到预测方法：

$$\operatorname{argmax}_c P(y = c | \mathbf{x}) \propto \operatorname{argmax}_c \hat{\pi}_c \prod_{\alpha=1}^d \hat{\theta}_{\alpha c}^{x_{\alpha}} \quad (4.25)$$

4.6.3 实例 3: 连续特征（高斯朴素贝叶斯法）

该实例中的数据特征表现为连续特征：

$$x_{\alpha} \in \mathbb{R} \quad (\text{每一维的特征都取实数值}). \quad (4.26)$$

用高斯分布来表征 $P(x_{\alpha} | y)$ ：

$$P(x_{\alpha} | y = c) = \mathcal{N}(\mu_{\alpha c}, \sigma_{\alpha c}^2) = \frac{1}{\sqrt{2\pi}\sigma_{\alpha c}} e^{-\frac{1}{2}\left(\frac{x_{\alpha}-\mu_{\alpha c}}{\sigma_{\alpha c}}\right)^2}. \quad (4.27)$$

值得注意的是，上述模型是基于每一维特征 α 都服从一个独立的条件高斯分布的假设而成立的，而数据的整体分布将服从一个高维的高斯分布，即 $P(\mathbf{x}|y) \sim \mathcal{N}(\mu_y, \Sigma_y)$ ，其中 Σ_y 是一个对角的协方差矩阵，且 $[\Sigma_y]_{\alpha, \alpha} = \sigma_{\alpha, y}^2$ 。

因为高斯分布只有两个参数，均值和方差。参数均值 $\mu_{\alpha c}$ 和方差 $\sigma_{\alpha c}^2$ 的估计可由所有分类标签为 c 的数据中特征维度 α 的平均值和方差得到：

$$\mu_{\alpha c} \leftarrow \frac{1}{n_c} \sum_{i=1}^n I(y_i = c) x_{i\alpha} \quad (\text{其中 } n_c = \sum_{i=1}^n I(y_i = c)) \quad (4.28)$$

$$\sigma_{\alpha c}^2 \leftarrow \frac{1}{n_c} \sum_{i=1}^n I(y_i = c) (x_{i\alpha} - \mu_{\alpha c})^2. \quad (4.29)$$

4.7 朴素贝叶斯法是一种线性分类器

假设 $y_i \in \{-1, +1\}$ ，并且特征是多项式形式的，那么可以证明：

$$h(\mathbf{x}) = \operatorname{argmax}_y P(y) \prod_{\alpha=1}^d P(x_{\alpha} | y) = \operatorname{sign}(\mathbf{w}^{\top} \mathbf{x} + b). \quad (4.30)$$

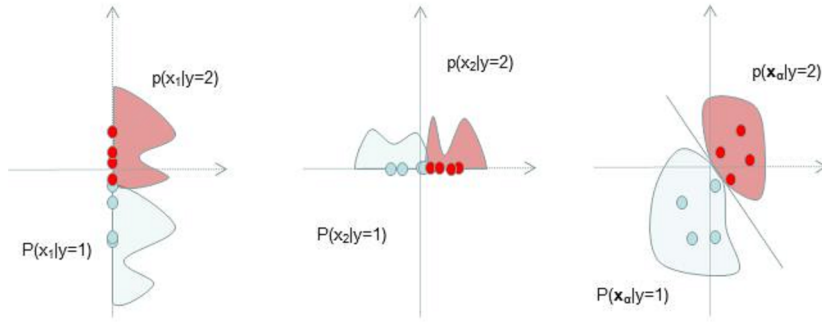


图 4-3 在很多情况下，朴素贝叶斯法将生成一个线性决策边界。此处假定 $P(x_\alpha | y)$ 满足高斯分布且对每种类别 c 标准差 $\Sigma_{\alpha,c}$ 都是理想的（尽管可能在不同的 α 维度上值不相同）。最右侧图中直线表示决策边界，即 $P(y = 1|\mathbf{x}) = P(y = 2|\mathbf{x})$ 。

Figure 4-3 Naive Bayes leads to a linear decision boundary in many common cases.

也就是说朴素贝叶斯法是一种线性分类器，图4-3形象地描述了这一性质，具体表达为：

$$\mathbf{w}^T \mathbf{x} + b > 0 \iff h(\mathbf{x}) = +1. \quad (4.31)$$

下面来证明这一结论。首先定义 $P(x_\alpha | y = +1) \propto \theta_{\alpha+}^{x_\alpha}$ ， $P(Y = +1) = \pi_+$ 和 $P(Y = -1) = \pi_-$ ，则有：

$$[\mathbf{w}]_\alpha = \log(\theta_{\alpha+}) - \log(\theta_{\alpha-}) \quad (4.32)$$

$$b = \log(\pi_+) - \log(\pi_-). \quad (4.33)$$

如果用式 (4.32) 和式 (4.33) 去做分类, 则可以直接计算 $\mathbf{w}^\top \cdot \mathbf{x} + b$, 即:

$$\begin{aligned}
 \mathbf{w}^\top \mathbf{x} + b > 0 &\iff \sum_{\alpha=1}^d [\mathbf{x}]_{\alpha} \overbrace{(\log(\theta_{\alpha+}) - \log(\theta_{\alpha-}))}^{[\mathbf{w}]_{\alpha}} + \overbrace{\log(\pi_+) - \log(\pi_-)}^b > 0 \quad (\text{代入 } \mathbf{w}, b) \\
 &\iff \exp \left(\sum_{\alpha=1}^d [\mathbf{x}]_{\alpha} (\log(\theta_{\alpha+}) - \log(\theta_{\alpha-})) + \log(\pi_+) - \log(\pi_-) \right) > 1 \quad (\text{同取指数}) \\
 &\iff \prod_{\alpha=1}^d \frac{\exp(\log \theta_{\alpha+}^{[\mathbf{x}]_{\alpha}} + \log(\pi_+))}{\exp(\log \theta_{\alpha-}^{[\mathbf{x}]_{\alpha}} + \log(\pi_-))} > 1 \quad (a \log(b) = \log(b^a) \text{ 且 } e^{a-b} = \frac{e^a}{e^b}) \\
 &\iff \prod_{\alpha=1}^d \frac{\theta_{\alpha+}^{[\mathbf{x}]_{\alpha}} \pi_+}{\theta_{\alpha-}^{[\mathbf{x}]_{\alpha}} \pi_-} > 1 \quad (e^{\log(a)} = a \text{ 以及 } e^{a+b} = e^a e^b) \\
 &\iff \frac{\prod_{\alpha=1}^d P([\mathbf{x}]_{\alpha} | Y = +1) \pi_+}{\prod_{\alpha=1}^d P([\mathbf{x}]_{\alpha} | Y = -1) \pi_-} > 1 \quad (P([\mathbf{x}]_{\alpha} | Y = -1) = \theta_{\alpha-}^{[\mathbf{x}]_{\alpha}}) \\
 &\iff \frac{P(\mathbf{x} | Y = +1) \pi_+}{P(\mathbf{x} | Y = -1) \pi_-} > 1 \quad (\text{朴素贝叶斯假设}) \\
 &\iff \frac{P(Y = +1 | \mathbf{x})}{P(Y = -1 | \mathbf{x})} > 1 \quad (\text{贝叶斯定理, 因子 } P(\mathbf{x}) \text{ 上下抵消}) \\
 &\iff P(Y = +1 | \mathbf{x}) > P(Y = -1 | \mathbf{x}) \\
 &\iff \operatorname{argmax}_y P(Y = y | \mathbf{x}) = +1
 \end{aligned}$$

所以, 当且仅当朴素贝叶斯法预测结果为 +1 的时候, 点 \mathbf{x} 落在分割超平面的正侧。对于连续特征 (高斯朴素贝叶斯) 的情形, 可以证明:

$$P(y | \mathbf{x}) = \frac{1}{1 + e^{-y(\mathbf{w}^\top \mathbf{x} + b)}} \quad (4.34)$$

这一模型也叫逻辑回归 (Logistic Regression), 这将在下一讲中介绍。

4.8 思考题

- 为什么在高维特征空间或者特征 \mathbf{x} 是连续的情况下, 用 MLE 方法可能无法进行估计?
- 试由表 4-1 的训练数据学习一个朴素贝叶斯分类器并预测 $\mathbf{x} = (3, S)^T$ 的类标记 y 。表中 $[\mathbf{x}]_1$ 、 $[\mathbf{x}]_2$ 为特征在两个维度上的取值, 取值的集合分别为 $A_1 = \{1, 2, 3, 4\}$ 、 $A_2 = \{S, M, L\}$, y 为类标记, $y \in C = \{1, -1\}$ 。如果按照拉普拉斯平滑估计概率, 即取平滑参数 $l=1$, 此时该如何预测 $\mathbf{x} = (2, S)^T$ 的类标记 y ?

表 4-1 训练数据

Table 4-1 Training data

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
$[\mathbf{x}]_1$	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3	4	4	4	4	4
$[\mathbf{x}]_2$	<i>S</i>	<i>M</i>	<i>M</i>	<i>S</i>	<i>S</i>	<i>S</i>	<i>M</i>	<i>M</i>	<i>L</i>	<i>L</i>	<i>L</i>	<i>M</i>	<i>M</i>	<i>L</i>	<i>L</i>	<i>L</i>	<i>S</i>	<i>M</i>	<i>M</i>	<i>S</i>
y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1	1	1	1	-1	-1

5

线性模型

给定由 d 个属性描述的示例 $\mathbf{x} = (x_1; x_2; \cdots; x_d)$, 其中 x_i 是 \mathbf{x} 在第 i 个属性上的取值, 线性模型 (linear model) 试图学得一个通过属性的线性组合来进行预测的函数, 即

$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + \cdots + w_dx_d + b$$

一般用向量形式写成

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

其中 $\mathbf{w} = (w_1; w_2; \cdots; w_d)$ 。 \mathbf{w} 和 b 学得之后, 模型就得以确定。

线性模型形式简单、易于建模, 但却蕴涵着机器学习中一些重要的基本思想。许多功能更为强大的非线性模型 (nonlinear model) 可在线性模型的基础上通过引入层级结构或高维映射而得。此外, 由于它直观表达了各属性在预测中的重要性, 因此线性模型有很好的可解释性 (comprehensibility)。

5.1 感知机

在感知机 (Perceptron) 模型中, 假设数据是二分类且线性可分的, 数据标签是 +1 (正样本) 和 -1 (负样本), 模型的优化目标是寻找一个超平面将数据分离开来。

5.1.1 感知机分类模型

感知机分类模型如图5-1所示。给定正负两个类别的线性可分数据, 感知机需要寻找一个超平面 $\mathcal{H} = \{\mathbf{x} \mid \mathbf{w}^T \mathbf{x} + b = 0\}$, 通过该平面, 将正负数据划分在两个半空间中。

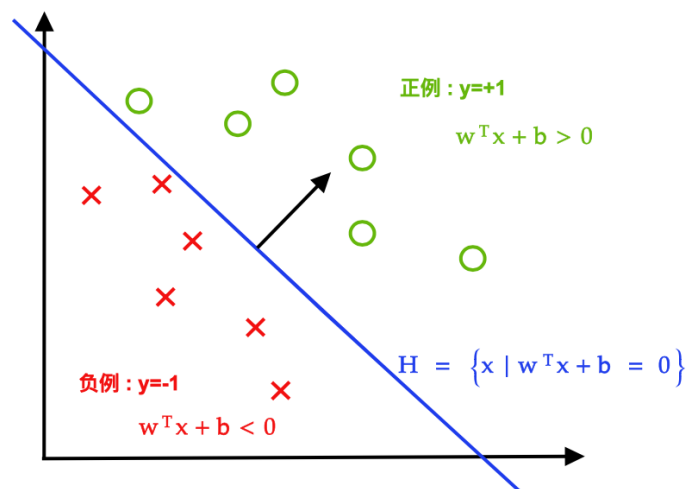


图 5-1 感知机分类示例

其中 b 是一个偏置，如果去掉则超平面经过原点。对每一个样本点来说，都可以计算 $h(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T \mathbf{x}_i + b)$ 。

为了方便处理，我们可以将 b 放入到权重向量 \mathbf{w} 当中，即：

$$\mathbf{x} \text{ 改写为 } \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}, \mathbf{w} \text{ 改写为 } \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$$

很容易验证处理后的超平面与原平面等价，即：

$$\begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix}^T \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} = \mathbf{w}^T \mathbf{x} + b$$

至此，感知机模型可以表示为：

$$h(\mathbf{x}_i) = \text{sign}(\mathbf{w}^T \mathbf{x}_i) \quad (5.1)$$

观察图5-1可以发现，当 $y_i (\mathbf{w}^T \mathbf{x}_i) > 0$ 时，表示 \mathbf{x}_i 被正确分类，这就是感知机算法的判别准则。从另一方面也可以看出，感知机算法只有当标签为 $y_i \in \{-1, +1\}$ 时才有效，当 $y_i \in \{0, +1\}$ 时是无效的。

5.1.2 感知机算法

感知机的模型参数是 \mathbf{w} （该参数定义了一个超平面），该超平面最初为 $\mathbf{w} = \mathbf{0}$ ，每一次选择一个分类错误的点，用其对参数 \mathbf{w} 进行更新，学习参数 \mathbf{w} 的算法如算法1所示。

Algorithm 1 感知机算法

```

Initialize  $\mathbf{w} = \mathbf{0}$ 
while True do
     $m = 0$ 
    for  $(\mathbf{x}_i, y_i) \in D$  do
        if  $y_i(\mathbf{w}^\top \mathbf{x}_i) \leq 0$  then
             $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ 
             $m \leftarrow m + 1$ 
        end if
    end for
    if  $m = 0$  then
        break
    end if
end while

```

图5-2是该算法的几何解释。在左图中，由 \mathbf{w}_t 定义的超平面没有成功区分一个红点 (-1) 和一个绿点 (+1)。在中图中，选择误分的红点用作更新，他的标签是-1，按照算法中定义的更新方式 $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ ，对超平面进行更新。更新后的超平面如右图所示， $\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{x}$ 成功将数据点区分。

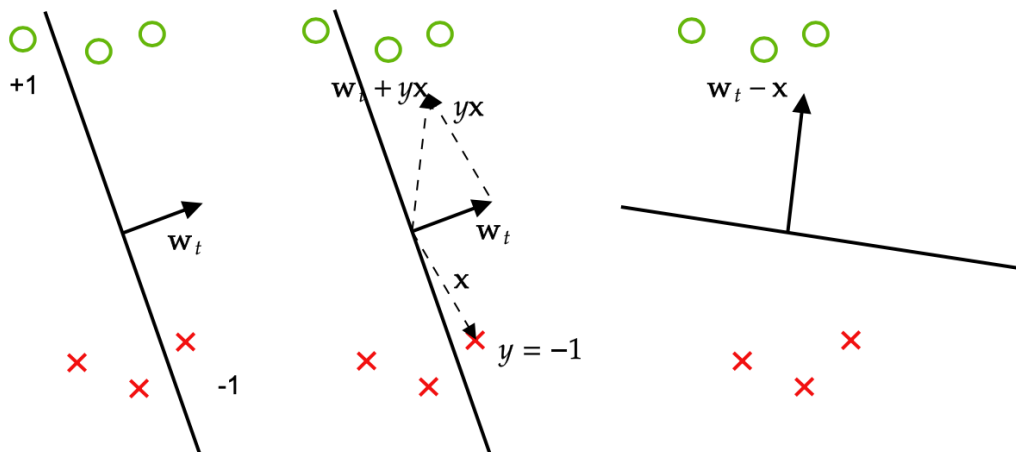


图 5-2 感知机算法几何解释

5.1.3 感知机收敛性证明

本节将根据感知机模型定义和更新过程，给出其收敛性证明。

假设对于数据集 D ， $\exists \mathbf{w}^*$ 使得对于 $\forall (\mathbf{x}_i, y_i) \in D$ ，都有 $y_i(\mathbf{x}_i^\top \mathbf{w}^*) > 0$ ，即该数据集是线性可分的。

将 \mathbf{w}^* 处理为单位向量并对每一个数据点 \mathbf{x}_i 进行放缩，使得

$$\|\mathbf{w}^*\| = 1 \text{ 并且 } \|\mathbf{x}_i\| \leq 1 \quad \forall \mathbf{x}_i \in D$$

定义 $\gamma = \min_{(\mathbf{x}_i, y_i) \in D} |\mathbf{x}_i^\top \mathbf{w}^*|$ ，表示所有数据点到超平面的最小距离。（ \mathbf{x}_i 在 \mathbf{w}^* 上的投影长度可以表示为 $\frac{\mathbf{x}_i^\top \mathbf{w}^*}{\|\mathbf{w}^*\|}$ ）容易看出，对于所有 \mathbf{x} ， $y(\mathbf{x}^\top \mathbf{w}^*) = |\mathbf{x}^\top \mathbf{w}^*| \geq \gamma$ 。

数据归一化结果如图5-3所示，我们有：

- 所有的数据点都在单位球内
- \mathbf{w}^* 落在单位球上（ $\|\mathbf{w}^*\| = 1$ ）
- γ 是超平面（蓝色线）距离最近的数据点的距离

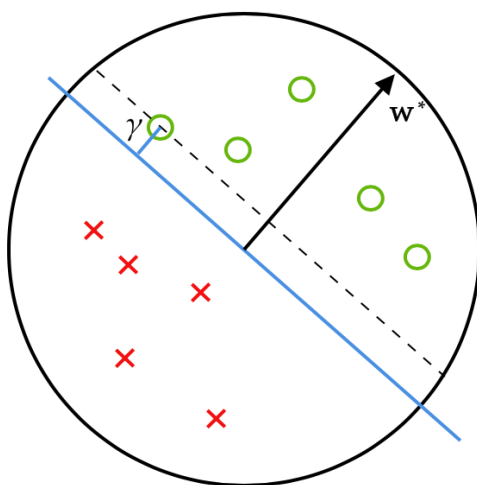


图 5-3 数据归一化

定理 1 满足上述三个条件的情况下，感知机算法最多需要 $\frac{1}{\gamma^2}$ 步可结束。

证明. 已知，一次更新发生时， \mathbf{w} 更新为 $\mathbf{w} + y\mathbf{x}$ 。我们已经有以下两个事实：

1. $y(\mathbf{x}^\top \mathbf{w}) \leq 0$ ：不等式成立因为当发生更新的时候该 \mathbf{x} 一定是被分类错误
2. $y(\mathbf{x}^\top \mathbf{w}^*) > 0$ ：不等式成立因为最优参数 \mathbf{w}^* 一定能使所有数据点分类正确

我们考虑每次更新对于 $\mathbf{w}^\top \mathbf{w}^*$ 和 $\mathbf{w}^\top \mathbf{w}$ 的影响。对于前者我们有：

$$(\mathbf{w} + y\mathbf{x})^\top \mathbf{w}^* = \mathbf{w}^\top \mathbf{w}^* + y(\mathbf{x}^\top \mathbf{w}^*) \geq \mathbf{w}^\top \mathbf{w}^* + \gamma$$

注意上述不等式满足是因为被 \mathbf{w}^* 定义的超平面距离 \mathbf{x} 的距离至少是 γ ，即 $y(\mathbf{x}^\top \mathbf{w}^*) = |\mathbf{x}^\top \mathbf{w}^*| \geq \gamma$ 。因此，对于每一次更新， $\mathbf{w}^\top \mathbf{w}^*$ 至少增加了 γ 。

此外，根据不等式 $2y(\mathbf{w}^\top \mathbf{x}) < 0$ 和 $y^2(\mathbf{x}^\top \mathbf{x}) \leq 1$ ，我们可以得到 $\mathbf{w}^\top \mathbf{w}$ 的变化满足：

$$(\mathbf{w} + y\mathbf{x})^\top (\mathbf{w} + y\mathbf{x}) = \mathbf{w}^\top \mathbf{w} + 2y(\mathbf{w}^\top \mathbf{x}) + y^2(\mathbf{x}^\top \mathbf{x}) \leq \mathbf{w}^\top \mathbf{w} + 1$$

因此，对于每一次更新， $\mathbf{w}^\top \mathbf{w}$ 至多增加了 1。

因为我们初始化 $\mathbf{w} = \mathbf{0}$ 为零向量，设我们已经更新 M 次，可以得到如下不等式：

$$1. \mathbf{w}^\top \mathbf{w}^* \geq M\gamma$$

$$2. \mathbf{w}^\top \mathbf{w} \leq M$$

因此，我们可以给出更新次数 M 的上界：

$$\begin{aligned} M\gamma &\leq \mathbf{w}^\top \mathbf{w}^* \\ &\leq |\mathbf{w}^\top \mathbf{w}^*| \\ &\leq \|\mathbf{w}\| \|\mathbf{w}^*\| && \text{柯西不等式} \\ &= \|\mathbf{w}\| && \|\mathbf{w}^*\| = 1 \\ &= \sqrt{\mathbf{w}^\top \mathbf{w}} \\ &\leq \sqrt{M} \end{aligned}$$

由此可以得到不等式 $M\gamma \leq \sqrt{M}$ ，求解该不等式可以得到更新次数 M 的上界。

$$\begin{aligned} M\gamma &\leq \sqrt{M} \\ \Rightarrow M^2\gamma^2 &\leq M \\ \Rightarrow M &\leq \frac{1}{\gamma^2} \end{aligned}$$

□

5.2 线性回归

线性回归 (Linear Regression, LR) 是对给定的训练数据集进行线性拟合，从而找到一条能使得大多数样本点都尽可能被准确预测的拟合线，如5-4所示，样本点均匀分布在拟合曲线两侧。

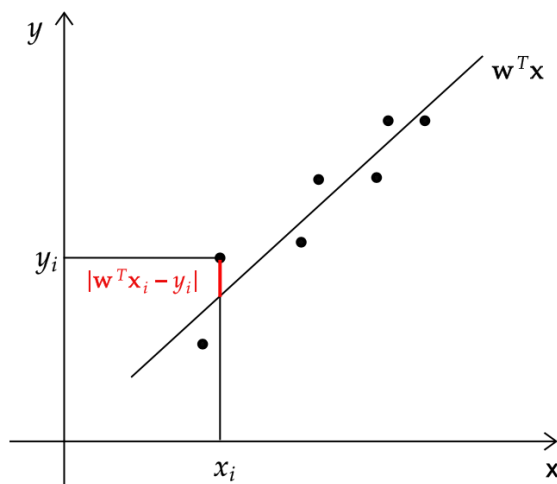


图 5-4 线性回归

5.2.1 假设

给定数据集 $D = \{\mathbf{x}_i, y_i\}, i = 1, \dots, n$, 其中 $\mathbf{x}_i \in \mathbb{R}^d, y_i \in \mathbb{R}$ 为了使用直线对数据进行拟合, 对样本点的分布进行了一定的假设。

$$\begin{aligned}
 y_i &= \mathbf{w}^\top \mathbf{x}_i + \epsilon_i \text{ where } \epsilon_i \sim N(0, \sigma^2) \\
 \Rightarrow y_i | \mathbf{x}_i &\sim N(\mathbf{w}^\top \mathbf{x}_i, \sigma^2) \\
 \Rightarrow P(y_i | \mathbf{x}_i, \mathbf{w}) &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\mathbf{x}_i^\top \mathbf{w} - y_i)^2}{2\sigma^2}}
 \end{aligned}$$

通俗地说, 我们假定数据分布在一条过原点的直线 $\mathbf{w}^\top \mathbf{x}$ 附近 (与感知机类似, 将偏置扩充进 \mathbf{w} 中)。对于特征为 \mathbf{x}_i 的数据点, y 中存在着均值为 $\mathbf{w}^\top \mathbf{x}_i$ 且方差为 σ^2 的高斯噪音。我们的任务便是通过数据估计直线的斜率 \mathbf{w} 。下面, 将分别通过最大似然估计和最大后验估计两种方法对直线的斜率 \mathbf{w} 进行估计。

5.2.2 最大似然估计

已知数据集为 $D = \{\mathbf{x}_1, y_1, \dots, \mathbf{x}_n, y_n\}$, 该问题相当于确定合适的参数 \mathbf{w} , 使得数据集上的点出现的概率最大, 即 $\mathbf{w} = \operatorname{argmax}_{\mathbf{w}} P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w})$ 。因为 D 上所有数据

独立分布，故 \mathbf{w} 的估计如下：

$$\begin{aligned}
 \mathbf{w} &= \operatorname{argmax}_{\mathbf{w}} P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^n P(y_i, \mathbf{x}_i | \mathbf{w}) && \text{数据点之间相互独立} \\
 &= \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i | \mathbf{w}) && \text{链式法则} \\
 &= \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i) && \mathbf{x}_i \text{ 与 } \mathbf{w} \text{ 无关} \\
 &= \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) && P(\mathbf{x}_i) \text{ 是常数} \\
 &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^n \log[P(y_i | \mathbf{x}_i, \mathbf{w})] && \log \text{ 是单调函数，使用 } \log \text{ 变相乘为相加} \\
 &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^n \left[\log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) + \log \left(e^{-\frac{(\mathbf{x}_i^T \mathbf{w} - y_i)^2}{2\sigma^2}} \right) \right] && \text{插入概率分布} \\
 &= \operatorname{argmax}_{\mathbf{w}} -\frac{1}{\sqrt{2\pi\sigma^2}} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 && \text{第一项是常数，且 } \log(e^z) = z \\
 &= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 && \text{最小化}
 \end{aligned}$$

由此可以得到 \mathbf{w} 的估算式，从几何意义上看，如图5-4所示，该式表示了使得 $f(\mathbf{x})$ 与 y 之间的均方误差最小化，即样本点到直线上的欧式距离。

通过优化损失函数 $l(\mathbf{w}) = (\mathbf{x}^T \mathbf{w} - \mathbf{y}^T)^T (\mathbf{x}^T \mathbf{w} - \mathbf{y}^T)$ 。最小化该损失函数的过程又被称为最小二乘 (OLS) 参数估计。OLS 可以通过梯度下降或牛顿法进行优化，也可以给出解析解。

$$\begin{aligned}
 \frac{\partial L}{\partial \mathbf{w}} &= \frac{\partial (\mathbf{w}^T \mathbf{X} \mathbf{X}^T \mathbf{w} - \mathbf{w}^T \mathbf{X} \mathbf{y}^T - \mathbf{y} \mathbf{X}^T \mathbf{w} + \mathbf{y} \mathbf{y}^T)}{\partial \mathbf{w}} \\
 &= (\mathbf{w}^T \mathbf{X} \mathbf{X}^T)^T + \mathbf{X} \mathbf{X}^T \mathbf{w} - \mathbf{X} \mathbf{y}^T - (\mathbf{y} \mathbf{X}^T)^T \\
 &= 0
 \end{aligned}$$

解得： $\mathbf{w} = (\mathbf{X} \mathbf{X}^T)^{-1} \mathbf{X} \mathbf{y}^T$ 。

补充:

$$\begin{aligned}\frac{\partial \mathbf{A}\mathbf{x}}{\partial \mathbf{x}} &= \mathbf{A}^T \\ \frac{\partial \mathbf{x}\mathbf{A}}{\partial \mathbf{x}} &= \mathbf{A} \\ \frac{\partial \mathbf{A}^T \mathbf{x} \mathbf{B}}{\partial \mathbf{x}} &= \mathbf{A} \mathbf{B}^T \\ \frac{\partial \mathbf{A}^T \mathbf{x}^T \mathbf{B}}{\partial \mathbf{x}} &= \mathbf{B} \mathbf{A}^T\end{aligned}$$

5.2.3 最大后验估计

额外模型假设: $P(\mathbf{w}) = \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{\mathbf{w}^T \mathbf{w}}{2\tau^2}}$, 即假设 \mathbf{w} 符合 $(0, \tau^2)$ 的正态分布。

$$\begin{aligned}\mathbf{w} &= \operatorname{argmax}_{\mathbf{w}} P(\mathbf{w}|D) \\ &= \operatorname{argmax}_{\mathbf{w}} \frac{P(D|\mathbf{w})P(\mathbf{w})}{P(D)} && \text{忽略常量 } P(D) \\ &= \operatorname{argmax}_{\mathbf{w}} P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w}) P(\mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i, \mathbf{x}_i | \mathbf{w}) \right] P(\mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i | \mathbf{w}) \right] P(\mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i) \right] P(\mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) \right] P(\mathbf{w}) \\ &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^n \log P(y_i | \mathbf{x}_i, \mathbf{w}) + \log P(\mathbf{w}) \\ &= \operatorname{argmin}_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 + \frac{1}{2\tau^2} \mathbf{w}^T \mathbf{w} \\ &= \operatorname{argmin}_{\mathbf{w}} \sum_{i=1}^n (\mathbf{x}_i^T \mathbf{w} - y_i)^2 + \lambda \mathbf{w}^T \mathbf{w} && \lambda = \frac{\sigma^2}{\tau^2}\end{aligned}$$

按照第5.2.2节中的计算方法，通过最小化损失函数 $l(\mathbf{w}) = (\mathbf{x}^T \mathbf{w} - \mathbf{y})^T (\mathbf{x}^T \mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}^T \mathbf{w}$ ，得到 \mathbf{w} 的取值 $\mathbf{w} = (\mathbf{X}\mathbf{X}^T + \lambda \mathbf{I})^{-1} \mathbf{X}\mathbf{y}^T$ 。

5.3 逻辑回归

第5.2节中讨论了如何使用线性模型进行回归学习，但是对于分类任务来说，线性回归在部分问题上面没有办法进行求解。下面以一个简单的例子进行说明。

图5-5是一个乳腺癌分类问题，横轴代表肿瘤的大小，纵轴代表预测值，预测阈值设置为 0.5，当大于 0.5 时认为是恶性肿瘤，小于 0.5 时表示是良性肿瘤。通过线性回归得到拟合曲线，可以看出当前情况下分类均是正确的。

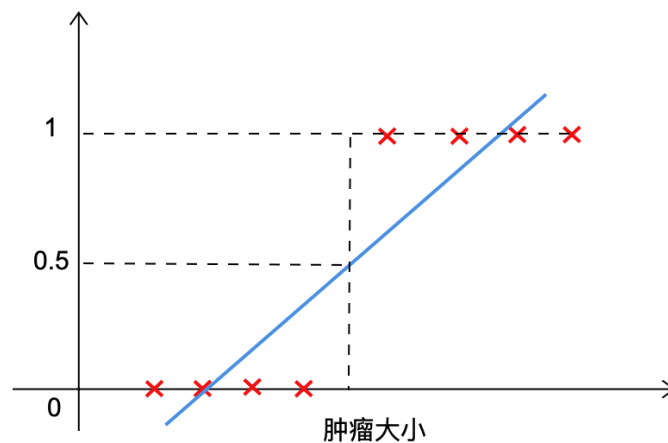


图 5-5 线性回归局限性说明 1

但是当训练集中出现一个体积非常大的恶性肿瘤时，这个样本点使得之前的拟合曲线发生变化，如图5-6所示，可以看出 0.5 的阈值点发生了变化，使得原来预测为 1 的样本预测为 0。由此看出线性回归模型做分类问题往往不能够得到好的结果。与此同时，线性回归模型的预测值为实值，因此需要找一个单调可微函数将分类任务的真实标记 y 与线性回归模型的预测值联系起来。

在本章中，我们为参数 \mathbf{w} 添加一个维度来将参数 \mathbf{b} 合并到参数 \mathbf{w} 中。

5.3.1 逻辑回归

逻辑回归 (Logistic Regression, LR) 就是在线性回归的基础上，引入 Sigmoid 函数，也就是将原来的预测值 $\mathbf{w}^T \mathbf{x}$ 作为自变量填充到 Sigmoid 函数中。对数几率函数就是一

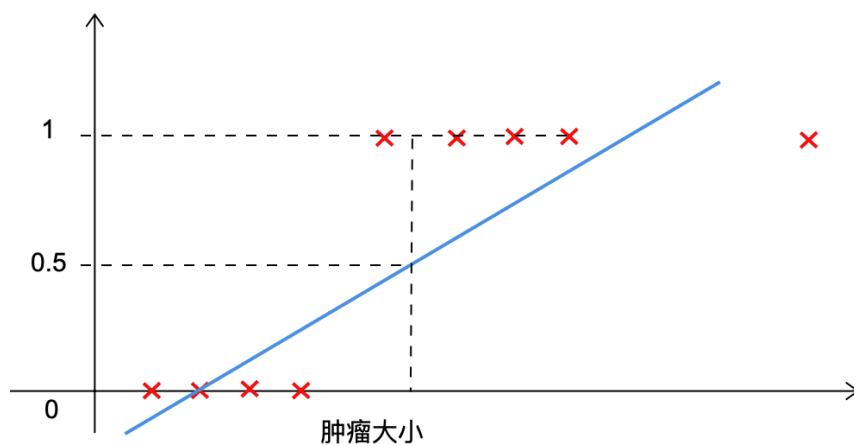


图 5-6 线性回归局限性说明 2

种常见的 Sigmoid 函数：

$$y = \frac{1}{1 + e^{-z}}$$

其函数图像如图5-7所示。

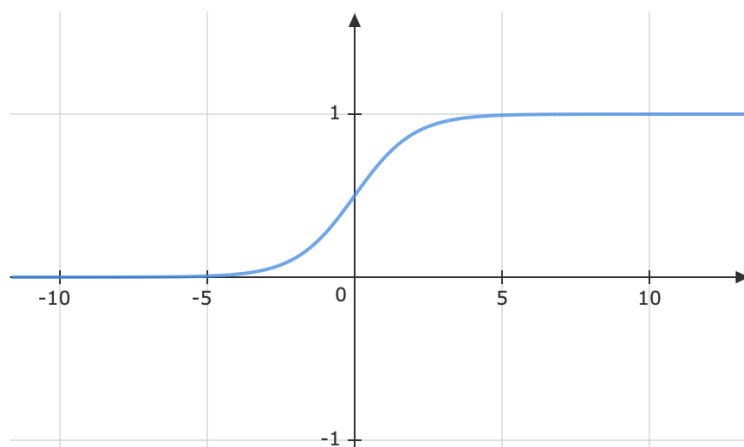


图 5-7 对数几率函数图像

我们需要引入一些新的概念：我们把一个事件的几率指代为该事件发生的概率与该事件不发生的概率比值；如果事件发生的概率是 p ，那么该事件的几率是 $\frac{p}{1-p}$ ，该事件

的对数几率是:

$$\text{logit}(p) = \log \frac{p}{1-p}$$

为了建立起上面两个式子的联系, 我们将 y 视为样本 \mathbf{x} 作为正例的可能性, 也就是 p , $1-y$ 视为样本 \mathbf{x} 作为反例的可能性, 也就是 $1-p$, 所以我们可以通过 $y = \frac{1}{1+e^{-\mathbf{w}^T \mathbf{x}}}$ 变化为如下形式:

$$\ln \frac{y}{1-y} = \mathbf{w}^T \mathbf{x}$$

为了对逻辑回归中的参数 \mathbf{w} 进行估计, 仿照第5.2节中的方法, 将给出最大似然估计和最大后验估计两种估计方法。

5.3.2 最大似然估计

在 MLE 中, 我们最大化条件似然函数来选择参数。条件似然函数 $P(y|X, \mathbf{w})$ 是在训练数据中以特征 x_i 为条件的观测值 $y \in \mathbb{R}^n$ 的概率函数。注意 $X = [x_1, \dots, x_i, \dots, x_n] \in \mathbb{R}^{d \times n}$ 。我们需要选择最优参数来最大化这个概率函数, 并且我们假设给定输入特征 x_i 和 \mathbf{w} , y_i 是相互独立的, 因此:

$$P(y|X, \mathbf{w}) = \prod_{i=1}^n P(y_i|x_i, \mathbf{w})$$

其中 $P(y_i|x_i, \mathbf{w}) = \frac{1}{1+e^{-y_i \mathbf{w}^T x_i}}$ 。将等式右边取对数, 得到:

$$\log\left(\prod_{i=1}^n P(y_i|x_i, \mathbf{w})\right) = -\sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i})$$

$$\begin{aligned} \hat{\mathbf{w}}_{MLE} &= \arg \max_{\mathbf{w}} -\sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i}) \\ &= \arg \min_{\mathbf{w}} \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i}) \end{aligned}$$

为了找到最优的参数 $\hat{\mathbf{w}}_{MLE}$, 我们可以求解 $\nabla_{\mathbf{w}} \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i}) = 0$ 。这个方程没有解析解, 所以我们在负对数似然函数 $\ell(\mathbf{w}) = \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i})$ 上应用梯度下降寻找最优的参数。

5.3.3 最大后验估计

在 MAP 中，我们将 \mathbf{w} 看作一个随机变量，并且假设它服从某种先验概率，例如： $\mathbf{w} \sim N(0, \sigma^2 I)$ ，这就是对逻辑回归的高斯近似。

MAP 的目标是在给定数据的条件下，寻找可能性最大的模型参数，即那些使得后验概率最大的参数。

$$P(\mathbf{w}|D) = P(\mathbf{w}|X, y) \propto P(y|X, \mathbf{w})P(\mathbf{w})$$

$$\hat{\mathbf{w}}_{MAP} = \arg \max_{\mathbf{w}} \log(P(y|X, \mathbf{w})P(\mathbf{w})) = \arg \min_{\mathbf{w}} \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i}) + \lambda \mathbf{w}^T \mathbf{w}$$

其中 $\lambda = \frac{1}{2\sigma^2}$ 。这个函数依然没有解析解，所以我们在负对数后验函数 $l(\mathbf{w}) = \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i}) + \lambda \mathbf{w}^T \mathbf{w}$ 上应用梯度下降来寻找最优的参数 \mathbf{w} 。

5.3.4 逻辑回归与朴素贝叶斯

机器学习模型大致可以分为两类：

- ▶ 生成模型：估计 $P(x_i, y)$ ，通常分别对 $P(x_i|y)$ 和 $P(y)$ 建模。
- ▶ 判别模型：对 $P(y|x_i)$ 建模。

朴素贝叶斯是一个生成模型，它对 $P(x_i|y)$ 建模，并且为这个概率分布做出显示的假设，例如：多项分布、高斯分布等，最后用 MLE 或 MAP 估计模型的参数。在后续的章节中，我们将详细介绍朴素贝叶斯，高斯朴素贝叶斯 $P(y|x_i) = \frac{1}{1 + e^{-y(\mathbf{w}^T x_i + b)}}$ ，其中 $y \in \{+1, -1\}$ ，它是由 $P(x_i|y)$ 唯一决定的。

通常认为，逻辑回归是与朴素贝叶斯对应的判别模型。在逻辑回归中，我们对 $P(y|x_i)$ 建模，并且假设它的形式为：

$$P(y|x_i) = \frac{1}{1 + e^{-y(\mathbf{w}^T x_i + b)}}$$

逻辑回归对 $P(x_i|y)$ 只做了极少的假设，它可以是高斯分布，也可以是多项分布，这无关紧要，因为我们直接通过 MLE 或 MAP 最大化条件似然函数 $\prod_i P(y_i|x_i; \mathbf{w}, b)$ 。

5.4 本章总结

本章分别介绍了感知机模型、线性回归模型、逻辑回归模型。感知机模型主要是假设数据是二分类线性可分的，并通过一个超平面将数据分离开来；线性回归模型主要是通过直线拟合数据，找到使得大部分样本点能够被准确预测的拟合线；逻辑回归在线性回归的基础上加入 sigmoid 函数，能够对分类问题进行求解。

在本章中，我们不仅介绍了主要的线性模型原理，还对线性模型的参数进行了解析意义上的估计。本章中涉及到最大似然估计，和最大后验估计，感兴趣的读者可以自行查阅相关资料进行了解。

6

梯度下降

在机器学习训练模型的过程中，我们使用损失函数来度量模型拟合程度。因此不仅需要选择一个合适的损失函数进行度量，还需要选择合适的方法最小化损失函数。本节主要介绍基本的模型优化算法。

6.1 数学基础: 泰勒展开

在正式介绍梯度下降之前，首先回顾一阶泰勒展开式。对于任意函数 f ，记函数 f 在点 x_0 处的导数为 $\nabla f(x_0)$ 。则函数 $f(x)$ 的一阶泰勒展开式为

$$f(x) \approx f(x_0) + \nabla f(x_0)(x - x_0)$$

一阶泰勒展开式对函数 $f(x)$ 进行逼近，利用函数在 x_0 处的值和对应的一阶梯度 $\nabla f(x_0)$ 进行线性近似。

除对 $f(x)$ 使用一阶泰勒展开近似外，还可以利用二阶微分对 $f(x)$ 进行二阶泰勒展开近似：

$$f(x + \Delta x) \approx f(x) + \nabla f(x)\Delta x + \frac{1}{2}\Delta x^T H(x)\Delta x$$

二阶泰勒展开相比于一阶泰勒展开式，对函数进行近似精度提升，但需要更多的计算开销。因此在实际场景中，往往对损失函数进行一阶泰勒近似。

6.2 梯度下降法

我们使用损失函数的一阶梯度进行梯度下降，并且为提升计算速度，我们每次只随机抽取数据集的一部分做梯度计算。一阶泰勒展开损失函数的表达式为

$$\ell(\mathbf{w}) \approx \ell(\mathbf{w}_0) + \nabla \ell(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) + O((\mathbf{w} - \mathbf{w}_0)^2)$$

由上式可知一阶泰勒展开表达式的近似误差是二阶的。根据梯度定义可知，梯度方向是函数值增长最快的方向，因此最小化损失则只需沿梯度的反方向更新。当损失函数 l 在 w_0 附近时，则近似于线性表达式

$$\ell(\mathbf{w}) = \ell(\mathbf{w}_0) + \nabla \ell(\mathbf{w}_0)(\mathbf{w} - \mathbf{w}_0) + O((\mathbf{w} - \mathbf{w}_0)^2)$$

根据一阶泰勒展开表达式可知，这个等号成立的条件为 $w - w_0$ 足够小，使二阶无穷小量误差可以忽略。我们取 $w = w_0 - \alpha \nabla l(w_0)$ ，其中 $\alpha > 0$ 是一个接近于 0 的小量，我们有：

$$\underbrace{\ell(\mathbf{w}_0 - \alpha \nabla \ell(\mathbf{w}_0))}_{\text{after one update}} \approx \ell(\mathbf{w}_0) - \underbrace{\alpha \nabla \ell(\mathbf{w}_0)^T \nabla \ell(\mathbf{w}_0)}_{>0} + O((\alpha \nabla \ell(\mathbf{w}_0))^2) < \ell(\mathbf{w}_0)$$

在梯度下降的更新过程中，我们需要设定一个大小合适的学习率 $\alpha > 0$ ，来帮助模型参数收敛。设定合适的学习速率是困难的，如果学习率过大，算法很容易发散（如图 8-1 所示）；而学习率过大，可能得到局部极小点。一般来说，初始状态下由于模型效果较差，因此可以通过较大的学习率更新参数；但随着模型不断得到更新，损失函数逐渐平缓，这时较小的学习率可以使得模型收敛。因此一种保证模型收敛的方法是设定 $\alpha = \frac{t_0}{t}$ ，这样动态地更新学习率的方式，可以兼顾训练速度与避免模型发散。

上述的梯度下降，在每一次迭代的过程，对所有样本数据的梯度都进行计算。虽然这种方式下梯度下降方向准确，但计算所有样本的梯度会产生巨大的时间和计算资源开销。在此算法的基础上对样本梯度的运算过程改进，则有下面两种主流的梯度下降算法：随机梯度下降（Stochastic GD）和小批量样本梯度下降（Mini Batch GD）。

随机梯度下降算法：随机梯度下降算法只随机抽取数据集中的一个小样本进行梯度更新。由于每次梯度下降迭代只计算一次梯度，因此相应的运算时间和计算资源则对应单个样本开销。但每次训练过程用于梯度计算的数据量太小（只有一个），因此训练过程中的梯度信息相比于全批量梯度更新，随机性和方差也更大，模型收敛也需要更多的迭代步数。

小批量样本梯度下降算法：随机梯度下降算法只随机使用一个小样本计算梯度，计算得到的梯度随机性较大，因此一种自然的想法就是使用一部分随机数据样本计算平均梯

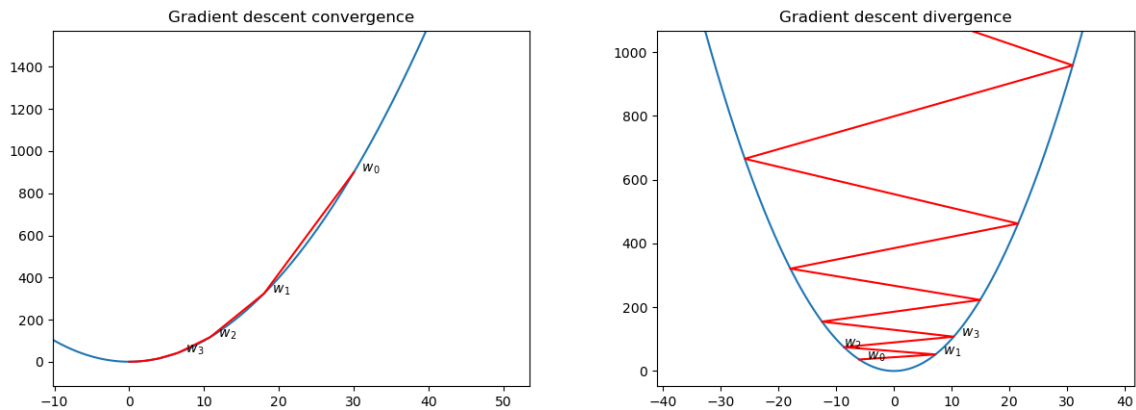


图 6-1 梯度下降收敛和发散过程示意图

度，这样既减小了梯度的随机性，又不会有太高的时间和计算开销，故提出小批量样本梯度下降算法。每次梯度下降，只选取一部分的样本数据计算梯度，比如整体样本 1/100 的数据。在数据量较大的情况，可以明显地减少梯度更新的时间。

6.3 基本一阶算法

下面对基本一阶优化算法进行介绍。

6.3.1 小批量样本梯度下降：使用一阶近似

小批量样本梯度下降及其变种可能是机器学习中应用最多的优化算法。根据上一节的介绍，从给定的数据集中抽取 m 个小批量 (独立同分布的) 样本，通过计算它们的梯度均值，我们可以得到梯度的无偏估计。

Algorithm 2 小批量样本梯度下降法

Require: 学习率 α ，初始参数 θ

while 停止准则未满足 **do**

 从数据集中随机选 m 个样本 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ 作为当前轮次梯度计算的数据集

 计算梯度估计: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i \mathbf{L}(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 参数更新: $\theta \leftarrow \theta - \alpha \hat{\mathbf{g}}$

end while

6.3.2 动量梯度更新

虽然随机梯度下降被广泛应用，但其优化过程速度较慢，并且经常随机梯度下降会更新到局部极小值。动量方法在加速高曲率、小但一致梯度、或是带噪声的梯度有较好的表现。动量算法积累了之前梯度的移动平均，并且沿着积累的梯度方向更新。从形式上看，动量算法中的动量和物理中的动量有一样的形式，都是对一段时间的信息进行积累，从而防止随机扰动对梯度更新的影响。

Algorithm 3 使用动量的小批量样本梯度下降法

Require: 学习率 ϵ ，动量参数 α ，初始参数 θ ，初始速度 \mathbf{v}

while 停止准则未满足 **do**

 从数据集中随机选 m 个样本 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ 作为当前轮次梯度计算的数据集

 计算梯度估计: $\hat{\mathbf{g}} \leftarrow +\frac{1}{m} \nabla_{\theta} \sum_i \mathbf{L}(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 计算速度更新: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$

 参数更新: $\theta \leftarrow \theta + \mathbf{v}$

end while

随机梯度下降下，步长是梯度的范数乘以学习率。而动量更新方法，步长不仅与当前的梯度有关，还与历史梯度有关。当之前的梯度都有大致相同的方向时，步长最大。如果动量算法的梯度总是 \mathbf{g} 时，那么会在方向 $-\mathbf{g}$ 上不断加速，直到最终速度，对应更新的步长为

$$\frac{\epsilon \|\mathbf{g}\|}{1 - \alpha}$$

其中， $\frac{1}{1-\alpha}$ 表示对应积累了 $\frac{1}{1-\alpha}$ 次梯度信息。例如常见的 α 值有 0.5, 0.9, 0.99 分别表示最大速度 2, 10 或 100 倍于梯度值。在实际参数更新过程中， α 随迭代轮数的增加而增加，更新越长，梯度的经验积累就更重要。

我们可以用牛顿动力学下石头来进行直观的理解。假设有一个石头 (参数) 在斜坡上，石头由于受到重力的作用则会滚向山谷 (凸优化) 的谷底 (局部最优)。但如果重力 (代价函数的梯度) 是唯一存在的力，那么当落入谷底的时候，则停止滚动，于是陷入局部极小点。但当采取动量方式时，由于惯性的存在石头在经过最低点时会继续向前运动直到动量衰竭至 0，从而突破局部极小点。

6.3.3 Nesterov 动量方法

Nesterov 动量受 Nesterov 加速梯度算法的启发，对动量更新的方法进行改进

$$\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\epsilon}{m} \nabla_{\theta} \sum_{i=1}^m \mathbf{L}(f(\mathbf{x}^{(i)}; \theta + \alpha \mathbf{v}), \mathbf{y}^{(i)})$$

$$\theta \leftarrow \theta + \mathbf{v}$$

改进后的表达式， α 和 ϵ 发挥的作用和标准动量方法中的作用类似。Nesterov 动量可以解释为在标准动量的方法中添加了一个校正因子。

Algorithm 4 使用 Nesterov 动量的小批量样本梯度下降法

Require: 学习率 ϵ ，动量参数 α ，初始参数 θ ，初始速度 \mathbf{v}

while 停止准则未满足 **do**

 从数据集中随机选 m 个样本 $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}$ 作为当前轮次梯度计算的数据集

 参数临时更新: $\hat{\theta} \leftarrow \theta + \alpha \mathbf{v}$

 计算梯度估计: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\hat{\theta}} \sum_i \mathbf{L}(f(\mathbf{x}^{(i)}; \hat{\theta}), \mathbf{y}^{(i)})$

 速度更新: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \hat{\mathbf{g}}$

 参数更新: $\theta \leftarrow \theta + \mathbf{v}$

end while

在凸批量优化的情况下，使用 Nesterov 动量可以将额外误差收敛率从 $O(1/k)$ 改进到 $O(1/k^2)$ ，但是如果在随机梯度的情况下，Nesterov 动量则无法改进收敛率。

6.4 二阶近似方法

与一阶方法相比，二阶方法使用二阶导数改进优化。本节对常用的二阶近似方法进行介绍。

6.4.1 牛顿法

牛顿法假定代价函数 f 二阶可微，因此使用 Hessian 矩阵（二阶微分）来近似。Hessian 矩阵即为二阶偏导组成的矩阵，Hessian 矩阵的表达式为：

$$\begin{bmatrix} \frac{\partial^2 \mathbf{f}}{\partial x_1^2} & \frac{\partial^2 \mathbf{f}}{\partial x_1 x_2} & \dots & \frac{\partial^2 \mathbf{f}}{\partial x_1 x_n} \\ \frac{\partial^2 \mathbf{f}}{\partial x_2 x_1} & \frac{\partial^2 \mathbf{f}}{\partial x_2^2} & \dots & \frac{\partial^2 \mathbf{f}}{\partial x_2 x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 \mathbf{f}}{\partial x_n x_1} & \frac{\partial^2 \mathbf{f}}{\partial x_n x_2} & \dots & \frac{\partial^2 \mathbf{f}}{\partial x_n^2} \end{bmatrix} \quad (6.1)$$

因为函数 ℓ 具有凸性，所以我们的 Hessian 矩阵既是对称平方矩阵，也是半正定矩阵。

注：如果矩阵 \mathbf{M} 特征值非负，则称矩阵 \mathbf{M} 为半正定矩阵。换言之，对于任意向量 \mathbf{x} ，都满足 $\mathbf{x}^T \mathbf{M} \mathbf{x} \geq 0$ 。

使用牛顿法进行的梯度下降算法损失函数 ℓ 可以用以下方法近似：

$$\ell(\mathbf{w} + \Delta\mathbf{w}) \approx \ell(\mathbf{w}) + g(\mathbf{w})^\top \Delta\mathbf{w} + \frac{1}{2} \Delta\mathbf{w}^\top H(\mathbf{w}) \Delta\mathbf{w}$$

为了得到损失函数 ℓ 的最小值，使用上述近似对原等式进行变换，原问题变为求解近似表达式的最小值。近似表达式是一个凸的抛物线，所以我们可以通过求解以下优化方程来找到它的最小值：

$$\arg \min_{\Delta\mathbf{w}} \{ \ell(\mathbf{w}) + g(\mathbf{w})^\top \Delta\mathbf{w} + \frac{1}{2} \Delta\mathbf{w}^\top H(\mathbf{w}) \Delta\mathbf{w} \}$$

为了找到目标函数的最小值，我们对变量 $\Delta\mathbf{w}$ 求一阶导数，令一阶导数等于 0，来得到 $\Delta\mathbf{w}$ 的极小值：

$$g(\mathbf{w}) + H(\mathbf{w})\Delta\mathbf{w} = 0 \quad (6.2)$$

$$\Rightarrow \Delta\mathbf{w} = -[H(\mathbf{w})]^{-1}g(\mathbf{w}). \quad (6.3)$$

Algorithm 5 牛顿法

while 没有达到停止准则 **do**

 计算梯度 $g := \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} \ell(y^{(i)}, y^{(i)})$

 计算 Hessian 矩阵 $H := \frac{1}{m} \sum_{i=1}^m \nabla_{\theta}^2 \ell(y^{(i)}, y^{(i)})$

 计算 Hessian 逆矩阵 H^{-1}

 参数更新 $\theta := \theta - H^{-1}g$

end while

对于局部二次函数，牛顿法会直接跳到极小值处。但牛顿法经常伴随梯度更新扩散问题。即如果函数“平坦”或者在某些维度上近似“平坦”，在这种情况下，二阶导数接近 0，而二阶导数的倒数近似于无穷大，则会带来梯度爆炸。不同于梯度下降，牛顿法没有限制步长的方法。由于泰勒近似要求逼近限制在当前点附近，爆炸的步长可能会导致当前的估计移动到泰勒近似不再精确的区域。

牛顿法的优点在于利用了二阶信息，相较梯度下降法，下降速度更快，但缺点在于利用二阶矩阵信息需要计算逆，算法复杂度为 $O(k^3)$ ，其中 k 为参数的个数。

6.4.2 共轭梯度法

共轭梯度是一种通过利用迭代下降的共轭方向以避免 Hessian 矩阵求逆的计算方法。这种方法的灵感来自对当前牛顿法弱点的改进。

在共轭梯度法中，我们寻求一个和当前线搜索方法共轭的搜索方向，即它不会抵消在该方向上的搜索。在训练迭代 t 时，下一步搜索方向 \mathbf{d}_t 的形式如下：

$$\mathbf{d}_t = \nabla_{\theta} \mathbf{J}(\theta) + \beta_t \mathbf{d}_{t-1}$$

其中系数 β_t 控制先前方向 \mathbf{d}_{t-1} 对搜索的影响。

如果 $\mathbf{d}^T \mathbf{H} \mathbf{d}_{t-1} = 0$ ，其中 \mathbf{H} 是 Hessian 矩阵，则两个方向 \mathbf{d} 和 \mathbf{d}_{t-1} 可被称为共轭的。两个计算 β_t 的流行方法表达式分别是

(1) Fletcher-Reeves:

$$\beta_t = \frac{\nabla_{\theta} \mathbf{J}(\theta_t)^T \nabla_{\theta} \mathbf{J}(\theta_t)}{\nabla_{\theta} \mathbf{J}(\theta_{t-1})^T \nabla_{\theta} \mathbf{J}(\theta_{t-1})}$$

(2) Polak-Ribiere:

$$\beta_t = \frac{\nabla_{\theta} \mathbf{J}(\theta_t)^T \nabla_{\theta} \mathbf{J}(\theta_t)}{\nabla_{\theta} \mathbf{J}(\theta_{t-1})^T \nabla_{\theta} \mathbf{J}(\theta_{t-1})}$$

对于二次曲面，梯度方向保证梯度沿着前一方向大小不变。因此，共轭梯度至多需要 k 次线性搜索就可以达到极小值。

Algorithm 6 共轭梯度方法

Require: 初始参数 θ_0 ，包含 m 个样本的训练集初始化 $t = 1, g_0 = 0, \rho_0 = 0$

while 没有达到停止准则 **do**

 初始化梯度 $\mathbf{g}_t = 0$

 计算梯度： $\mathbf{g}_t \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

 计算 $\beta_t = \frac{(\mathbf{g}_t - \mathbf{g}_{t-1})^T \mathbf{g}_t}{\mathbf{g}_{t-1}^T \mathbf{g}_{t-1}}$

 计算搜索方向： $\rho_t = -\mathbf{g}_t + \beta_t \rho_{t-1}$

 执行线搜索寻找： $\epsilon^* = \operatorname{argmin}_{\epsilon} \frac{1}{m} \sum_{i=1}^m L(f(\mathbf{x}^{(i)}; \theta_t + \epsilon \rho_t), \mathbf{y}^{(i)})$

 参数更新： $\theta_{t+1} = \theta_t + \epsilon^* \rho_t$

$\mathbf{t} \leftarrow \mathbf{t} + 1$

end while

6.5 自适应学习率算法

6.5.1 AdaGrad 算法

使用梯度下降法对模型参数更新，需要选择合适的学习率。但对于不同的任务合适的学习率也各不相同，因此我们可以考虑尝试根据训练数据动态更新学习率。

在学习率固定的参数更新基础上，提出了自适应梯度下降法 AdaGrad (Adaptive Gradient)，该方法对每一个特征自适应设定更新步长，是梯度下降优化算法的扩展。

AdaGrad 的主要优点之一是它不需要手动调整学习率。在迭代过程中，AdaGrad 动态调整学习率，让目标函数中的每个参数都有相适应的学习率。大多数实现中对学习率默认值为 0.01，设置一个较大的初始学习率。AdaGrad 的缺点之一是它在分母中累积平方梯度：由于每个添加项都是正数，因此在训练过程中累积和不断增大。这反过来会导致学习率不断变小并最终变得无限小，使得模型停止更新。

Algorithm 7 Adagrad 算法

```

学习率  $\eta$ , 模型参数  $\theta$ , 梯度积累变量  $\mathbf{r} = 0$ 
while 没有达到停止准则 do
    从训练集中随机采样  $m$  个数据样本  $\{x^1, \dots, x^m\}$ 
    初始化设置  $\mathbf{g} = \mathbf{0}$ 
    for  $i = 1$  to  $m$  do
        计算梯度:  $\mathbf{g} \leftarrow \mathbf{g} + \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(f(x^i, \theta), y^i)$ 
    end for
    计算梯度:  $\mathbf{r} \leftarrow \mathbf{r} + \mathbf{g}^2$ 
    计算更新:  $\Delta \theta \leftarrow -\frac{\eta}{\sqrt{\mathbf{r}}} \odot \mathbf{g}$ 
    参数更新:  $\theta \leftarrow \theta + \Delta \theta$ 
end while
  
```

6.5.2 RMSProp 算法

RMSProp 算法相较于 AdaGrad，在非凸优化下的有更佳表现，对梯度积累更改为指数加权的移动平均。AdaGrad 在训练时，参数更新轨迹可能穿过了多种不同的结构，最终到达一个局部是凸碗的结构。但由于 AdaGrad 算法根据所有历史平方梯度缩减学习率，使得在没有到达最优解附近的区域时，学习率已经变得很小，没有办法收敛到最终区域。而 RMSProp 算法使用指数平均的方式以丢弃遥远的历史信息，从而使其可以更新至最终区域。

同样，RMSProp 算法也可以应用 Nesterov 动量进行加速。

6.5.3 Adam

Adam 则是另一种学习率自适应的优化算法。

Algorithm 8 RMSProp 算法

学习率 η , 模型参数 θ , 梯度积累变量 $\mathbf{r} = 0$, 衰减系数 ρ

while 没有达到停止准则 **do**

从训练集中随机采样 m 个数据样本 $\{x^1, \dots, x^m\}$

初始化 $\mathbf{g} = \mathbf{0}$

for $i = 1$ to m **do**

计算梯度: $\mathbf{g} \leftarrow \mathbf{g} + \frac{1}{m} \sum_{i=1}^m \nabla_{\theta} L(f(x^i, \theta), y^i)$

end for

计算梯度: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g}^2$

计算更新: $\Delta \theta \leftarrow -\frac{\eta}{\sqrt{\mathbf{r}}} \odot \mathbf{g}$

参数更新: $\theta \leftarrow \theta + \Delta \theta$

end while

Algorithm 9 使用 Nesterov 的 RMSProp 算法

学习率 η , 模型参数 θ , 梯度积累变量 $\mathbf{r} = 0$, 衰减系数 ρ , 动量系数 α , 初始参数 \mathbf{v}

while 没有达到停止准则 **do**

从训练集中随机采样 m 个数据样本 $\{x^1, \dots, x^m\}$

初始化 $\mathbf{g} = \mathbf{0}$

计算临时更新 $\hat{\theta} \leftarrow \theta + \alpha \mathbf{v}$

for $i = 1$ to m **do**

计算梯度: $\mathbf{g} \leftarrow \mathbf{g} + \frac{1}{m} \sum_{i=1}^m \nabla_{\hat{\theta}} L(f(x^i, \hat{\theta}), y^i)$

end for

计算梯度: $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho) \mathbf{g}^2$

累计速度 $\mathbf{v} \leftarrow \alpha \mathbf{v} - \frac{\epsilon}{\sqrt{\mathbf{r}}} \odot \mathbf{g}$

计算更新: $\Delta \theta \leftarrow -\frac{\eta}{\sqrt{\mathbf{r}}} \odot \mathbf{g}$

参数更新: $\theta \leftarrow \theta + \mathbf{v}$

end while

Algorithm 10 Adam 算法

步长 ϵ , 模型初始参数 θ , 矩估计的指数衰减速率 ρ_1 和 ρ_2

初始化一阶和二阶矩变量 $\mathbf{s} = \mathbf{0}$ 和 $\mathbf{r} = \mathbf{0}$

while 没有达到停止准则 **do**

 从训练集中随机采样 m 个数据样本 $\{x^1, \dots, x^m\}$

for $i = 1$ to m **do**

 计算梯度: $\mathbf{g} \leftarrow \mathbf{g} + \frac{1}{m} \sum_{i=1}^m \nabla_{\hat{\theta}} L(f(x^i, \hat{\theta}), y^i)$

end for

$t \leftarrow t + 1$

 更新有偏一阶矩估计: $\mathbf{s} \leftarrow \rho_1 \mathbf{s} + (1 - \rho_1) \mathbf{g}$

 更新有偏二阶矩估计: $\mathbf{r} \leftarrow \rho_2 \mathbf{r} + (1 - \rho_2) \mathbf{g} \odot \mathbf{g}$

 修正一阶矩的偏差: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}}{1 - \rho_1^t}$

 修正二阶矩的偏差: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}}{1 - \rho_2^t}$

 计算更新: $\Delta \theta = -\frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}} + \delta}}$

 参数更新: $\theta \leftarrow \theta + \Delta \theta$

end while

在 Adam 中, 将梯度的一阶矩 (指数加权) 估计加入动量。可以看出, 将动量加入 RMSProp 最直观的方法是将动量应用于缩放后的梯度。其次在 Adam 加入偏置修正, 通过从原点初始化的一阶矩 (动量项) 和二阶矩 (非中心的) 的估计修正。RMSProp 也可以看成缺失修正因子 (非中心的) 二阶矩估计。因此, 相比 Adam, RMSProp 二阶矩估计可能在训练初期有很高的偏置。

6.5.4 思考题

- (1) 如何验证目标函数梯度的准确性?
- (2) 在大训练数据量时, 当前主流梯度下降法各存在什么问题, 需要如何改进?
- (3) 随机梯度下降法失效的原因?
- (4) 如何改进随机梯度下降?

7

支持向量机

7.1 线性分类问题

考虑输入空间 X 为 \mathbb{R}^N ($N \geq 1$) 的一个子集, 输出空间为 $Y = \{-1, +1\}$, 目标函数为 $f: X \rightarrow Y$ 。二分类任务可以表述为: 利用根据未知分布 D 从样本空间 X 独立同分布采样得到的样本集合 $S = ((x_1, y_1), (x_2, y_2), \dots, (x_m, y_m))$, 并且 $f(x_i) = y_i, \forall i \in [1, m]$, 我们希望从假设函数空间 H 中找出一个最优的 $h \in H$, 使得如下泛化误差最小:

$$R_D(h) = \Pr_{x \sim D} [h(x) \neq f(x)]. \quad (7.1)$$

一个自然而简单的假设是目标函数为线性分类器, 或者说是 N 维空间中的超平面:

$$H = \{\mathbf{x} \mapsto \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}. \quad (7.2)$$

具有上述形式的函数 h 将超平面 $\mathbf{w} \cdot \mathbf{x} + b = 0$ 两侧的点标注为 $+1$ 和 -1 , 因而上述学习问题被称为“线性分类问题”。其中对于最大化到两个类中最近数据点的距离, 我们说它是最大边距的超平面。

7.2 SVM——线性可分情形

假设样本集合 S 是线性可分的, 即存在超平面完美地将样本集合中的正负样本分开。由于参数空间是连续的, 必然存在无穷多个在样本集上完成分类任务的超平面。因此, 我们可以选择一个最“好”的超平面, 使得正负样本与超平面之间具有最大边界。

7.2.1 主优化问题

注意到对超平面的参数 (\mathbf{w}, b) 等比例放缩并不改变超平面的几何性质，因此可以通过适当的放缩使得 $\min_{(\mathbf{x}, y) \in S} |\mathbf{w} \cdot \mathbf{x} + b| = 1$ ，放缩后的超平面称为**规范超平面**。由定义可得，对样本中的任意 $x_i (i \in [1, m])$ ，有 $|\mathbf{w} \cdot \mathbf{x}_i + b| \geq 1$ 。

从解析几何的结论可知，空间中任意一点 $\mathbf{x}_0 \in \mathbb{R}^N$ 到超平面 $\mathbf{w} \cdot \mathbf{x} + b = 0$ 的距离为：

$$\frac{|\mathbf{w} \cdot \mathbf{x}_0 + b|}{\|\mathbf{w}\|}. \quad (7.3)$$

对于规范超平面，定义其边界大小 ρ 为：

$$\rho = \min_{(\mathbf{x}, y) \in S} \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \quad (7.4)$$

因此，求取具有最大边界的超平面可以表述为如下优化问题：

$$\begin{aligned} \max_{\mathbf{w}, b} \quad & \frac{1}{\|\mathbf{w}\|} \\ \text{subject to:} \quad & y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1, \forall i \in [1, m] \end{aligned}$$

等价于：

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to:} \quad & y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1, \forall i \in [1, m] \end{aligned} \quad (7.5)$$

显而易见，该优化问题的目标函数为凸函数，约束条件为线性不等式组，因而是典型的二次规划问题（QP），可以用成熟的商业求解器求解。

7.2.2 对偶优化问题

对优化问题（7.5）引入拉格朗日乘子 $\alpha_i \geq 0, i \in [1, m]$ ，构造拉格朗日函数

$$L(\mathbf{w}, b, \alpha) = \frac{1}{2} \|\mathbf{w}\|^2 - \sum_{i=1}^m \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1],$$

得到原目标函数的对偶函数为

$$\begin{aligned} g(\alpha) &= \inf_{\mathbf{w}, b} L(\mathbf{w}, b, \alpha) \\ &= \sum_{i=1}^m \alpha_i + \inf_{\mathbf{w}} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - \left(\sum_{i=1}^m \alpha_i y_i \mathbf{x} \right) \cdot \mathbf{w} \right\} + \inf_b \left\{ - \sum_{i=1}^m \alpha_i y_i b \right\} \\ &= \begin{cases} \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) & \text{如果 } \sum_{i=1}^m \alpha_i y_i = 0 \\ -\infty & \text{其他情况} \end{cases} \end{aligned} \quad (7.6)$$

因而，对偶问题可以表述为：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{subject to :} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, \dots, m \end{aligned} \quad (7.7)$$

对偶优化问题的目标函数为凸函数，约束条件为线性不等式组，也是典型的二次规划问题（QP）。

7.2.3 支持向量

由于原问题和对偶问题的不等式约束条件都是线性的，强对偶条件成立，故最优解满足 KKT 条件：

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \quad (7.8)$$

$$\nabla_b L = - \sum_{i=1}^m \alpha_i y_i = 0 \implies \sum_{i=1}^m \alpha_i y_i = 0 \quad (7.9)$$

$$\forall i, \alpha_i [y_i ((\mathbf{w} \cdot \mathbf{x}_i + b) - 1)] = 0 \implies \alpha_i = 0 \vee y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1. \quad (7.10)$$

从互补松弛条件（7.10）可知，若 \mathbf{x}_i 不是距离分离超平面最近的点，即 $|\mathbf{w} \cdot \mathbf{x}_i + b| \neq 1$ ，则必有 $\alpha_i = 0$ 。反映在等式（7.8）中，说明上述 \mathbf{x}_i 对超平面方向的选择没有影响。因此，决定分离超平面方向 \mathbf{w} 的只有那些使 $y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1$ 成立的样本点，称这些点为**支撑向量**。

7.3 SVM——线性不可分情形

在数据线性不可分时，无法找到参数 (\mathbf{w}, b) 使得 $\mathbf{w} \cdot \mathbf{x}_i + b$ 与 y_i 的符号一致，也就是说，对任意超平面 $\mathbf{w} \cdot \mathbf{x} + b = 0$ ，存在 $\mathbf{x}_i \in S$ 使得

$$y_i [\mathbf{w} \cdot \mathbf{x}_i + b] < 1. \quad (7.11)$$

为此，我们引入松弛变量 $\xi_i \geq 0$ ，使得

$$y_i [\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1 - \xi_i. \quad (7.12)$$

这样一来，主优化问题改写成：

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i^p \\ \text{subject to :} \quad & y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i \in [1, m] \end{aligned} \quad (7.13)$$

下面只考虑 $p = 1$ 时的情形。

7.3.1 对偶优化问题

对优化问题 (7.13) 引入拉格朗日乘子 $\alpha_i, \beta_i \geq 0, i \in [1, m]$ ，构造拉格朗日函数

$$L(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x} + b) - 1 + \xi_i] - \sum_{i=1}^m \beta_i \xi_i,$$

得到原目标函数的对偶函数为

$$\begin{aligned} g(\alpha, \beta) &= \inf_{\mathbf{w}, b, \xi} L(\mathbf{w}, b, \xi, \alpha, \beta) \\ &= \sum_{i=1}^m \alpha_i \\ &\quad + \inf_{\mathbf{w}} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - \left(\sum_{i=1}^m \alpha_i y_i \mathbf{x} \right) \cdot \mathbf{w} \right\} \\ &\quad + \inf_b \left\{ - \sum_{i=1}^m \alpha_i y_i b \right\} \\ &\quad + \inf_{\xi} \left\{ \sum_{i=1}^m (C - \alpha_i - \beta_i) \xi_i \right\} \\ &= \begin{cases} \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j), & \text{如果 } \sum_{i=1}^m \alpha_i y_i = 0 \text{ 且 } \alpha_i + \beta_i = C, \\ -\infty, & \text{其他情况。} \end{cases} \end{aligned} \quad (7.14)$$

因而，对偶问题可以表述为：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{subject to :} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \\ & \alpha_i + \beta_i = C, \\ & \alpha_i \geq 0, \\ & \beta_i \geq 0, \quad i = 1, \dots, m \end{aligned}$$

等价于：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{subject to :} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \end{aligned} \quad (7.15)$$

7.3.2 KKT 条件

线性不可分问题的最优解满足如下 KKT 条件：

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \quad (7.16)$$

$$\nabla_b L = - \sum_{i=1}^m \alpha_i y_i = 0 \implies \sum_{i=1}^m \alpha_i y_i = 0 \quad (7.17)$$

$$\nabla_{\xi_i} L = C - \alpha_i - \beta_i = 0 \implies \alpha_i + \beta_i = C \quad (7.18)$$

$$\forall i, \alpha_i [y_i ((\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i)] = 0 \implies \alpha_i = 0 \vee y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1 - \xi_i. \quad (7.19)$$

$$\forall i, \beta_i \xi_i = 0 \implies \beta_i = 0 \vee \xi_i = 0. \quad (7.20)$$

结合 (7.18) 和 (7.19), (7.20) 可改写成：

$$\alpha_i = C \vee y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1. \quad (7.21)$$

这说明，对于 $\alpha_i \neq 0$ 对应的支撑向量 \mathbf{x}_i ，要么正好落在边界平面上： $y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1$ ，要么其对应的 α_i 正好等于 C 。

7.4 核函数

从对偶问题 (7.7) 和 (7.15) 发现，优化问题的目标函数只和样本点的内积 $(\mathbf{x}_i, \mathbf{x}_j)$ 有关。由此可以设想，把单纯的内积用某种二元函数 $K(\mathbf{x}_i, \mathbf{x}_j)$ 替换，使其等效于在另一种维度的空间（称之为**特征空间**）里做内积运算，这样或许能将原空间中的线性不可分问题变换成特征空间中的线性可分问题。

7.4.1 多项式核函数

对常数 $c > 0$ ，定义 d 维多项式核函数为：

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d \quad (7.22)$$

7.4.2 高斯核函数

对常数 $\sigma > 0$, 定义 高斯核函数 或者 径向基函数 为:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}' - \mathbf{x}\|^2}{2\sigma^2}\right) \quad (7.23)$$

7.4.3 sigmoid 核函数

对常数 $a, b \geq 0$, 定义 sigmoid 核函数 为:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, K(\mathbf{x}, \mathbf{x}') = \tanh(a(\mathbf{x} \cdot \mathbf{x}') + b) \quad (7.24)$$

因此, 基于核函数的 SVM 表述为如下问题:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to :} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \end{aligned} \quad (7.25)$$

8

学习准则与模型选择

8.1 学习准则

机器学习的目标在于通过训练样本，从假设空间中学习最优模型，使得函数对输入的估计与实际的输出之间的损失函数最小化。在这个过程中包含机器学习方法的几个基本要素：模型、学习准则、优化算法。本章主要介绍其中的学习准则，以及给定学习准则之后进行模型调试的一些通用策略。

8.1.1 损失函数和经验风险最小化准则

给定样本 x 及其标签 y ，假设存在模型 f 对 x 的标签进行预测，将其记为 $\hat{y} = f(x)$ 。我们知道一个好的模型应该在所有样本的预测值 \hat{y} 都与其标签 y 一致，然而实际情况是对于不同样本可能一致也可能不一致，所以我们需要用一个损失函数来量化模型预测值和真实标签之间的差距，记为 $L(y, \hat{y})$ 。以常见的平方损失函数为例可以表示为：

$$L(y, \hat{y}) = (y - \hat{y})^2$$

损失函数越小，意味着模型预测和标签差距越小，模型也就越好。然而损失函数只是度量模型预测的好坏，我们希望能够度量模型在一般情况下的好坏，就需要关心损失函数的期望。设模型的输入、输出是随机变量 (X, Y) ，他们满足联合分布 $P(X, Y)$ ，我们可以得到损失函数的期望如下：

$$R_{\text{exp}}(f) = E_p[L(Y, f(X))] = \int L(y, f(y))P(x, y)dx dy$$

这是模型 f 理论上关于联合分布 $P(X, Y)$ 的预测误差，称为风险函数。

机器学习算法的目标就是降低上式表示的期望误差。由于现实中 (X, Y) 的联合概率分布并不是已知的，因此 $R_{\text{exp}}(f)$ 只是理论上的一个概念，无法直接求得。虽然无法直接求得期望风险 $R_{\text{exp}}(f)$ ，但是可以使用样本对其进行估计，于是就有了经验风险。

假设给定训练集 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$ ，那么在 T 上预测的模型 f 的经验风险为

$$R_{\text{emp}}(f) = \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i))$$

根据大数定律，当 $N \rightarrow \infty$ 时， $R_{\text{emp}}(f) \rightarrow R_{\text{exp}}(f)$ 。

在模型、损失函数以及训练数据集确定的情况下，就可以确定经验风险函数。一种优化准则是找到确认模型的一组参数使得经验损失函数取得最小值。这就是经验风险最小化（Empirical Risk Minimization, ERM）准则。

8.1.2 过拟合和欠拟合

当我们拥有足够多的训练样本时，可以认为经验风险最小化可以保证很好的学习效果。然而在大多数情况下，训练样本往往只是真实数据一个很小的子集，并且训练样本包含一定的噪声数据。另外，假如模型过于简单，可能也无法捕捉到训练样本足够的信息，即使使用经验风险最小化准则也无法取得很好的效果。简单来说，模型在某个数据集上学习可能会出现两种有问题的情况——过拟合和欠拟合。

欠拟合：从训练集上学习到的分类器没有足够的表达能力，无法捕捉训练样本所提供的信息。在这种情况下，由于分类器没有充分考虑到训练集中的主要相关信息，训练误差和测试误差都会很高。

过拟合：在训练集上学习到的分类器用有过强的表达能力，以至于能够表达噪声等无关信息，无法用于准确推断未知数据。尽管训练误差会不断降低，但是由于分类器在学习到主要信息之后，还会依赖仅出现于训练集中而非广泛存在的关系进行决策，测试误差将再次开始增大。

8.1.3 结构风险最小化

从上一节可以看到，仅仅使用经验风险最小化准则学习模型参数，不能很好地反映全部数据的真实分布。为了解决过拟合问题，我们可以在经验风险最小化的基础上引入表示模型结构复杂度的惩罚项（正则化项）来限制模型的表达能力，使其关注我们希望

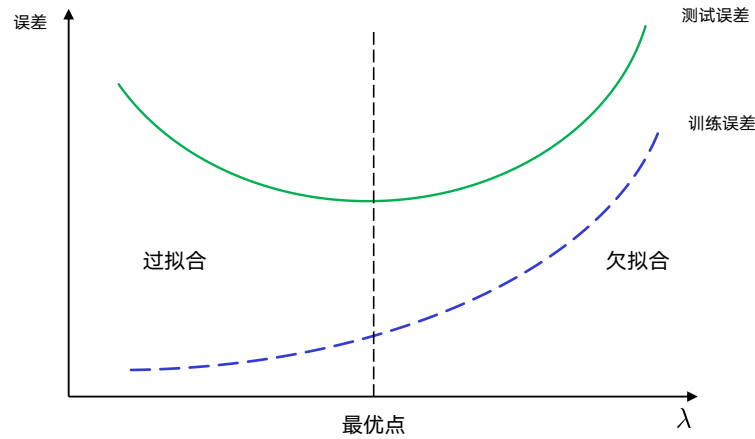


图 8-1 过拟合和欠拟合

其学习的特征信息。从而优化目标变为如下所示：

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f)$$

其中，模型 f 的复杂度越大， $J(f)$ 的值就越大。 $\lambda > 0$ 是系数，表示模型复杂度的重要性。这就是结构风险最小化（Structural Risk Minimization, SRM）准则。经验风险越来越小的时候，意味着模型的复杂度越来越大。随着经验风险（第一项）的减少，结构风险（第二项）会增大，从而抑制模型太过复杂，从而避免过拟合。对于超参数 λ 的选择需要根据实际使用的模型情况进行确定，需要进行多次尝试才可以最终确定。

8.2 偏差-方差分析

模型的泛化能力指的是经过数据集中的样本训练之后，在从未见过的数据上的表现准确度，而衡量这个表现准确度的高低用的是泛化误差。在统计学和机器学习中，分析模型的泛化误差有一个很有力的工具：偏差-方差分析。

偏差 (bias) 度量了模型的期望预测（样本内和样本外）与真实结果的偏离程度，高偏差意味着模型不能很好地将特征和目标关联起来。

方差 (variance) 描述的是模型预测结果的变化程度，也就是离预测值期望值的距离，方差越大，数据的分布越分散，模型会越关注随机噪声而不是特征本身。

经过一些简单的数学推导，模型的泛化误差可以被分解为三项：偏差，方差和系统噪声。通过拆解三个误差来源，我们可以更好地分析模型的预测结果，并设计相应的改进思路。

8.2.1 偏差-方差分解

与前面的假设相同，我们有训练样本 x 及其标签 y ，假设存在模型 f 对 x 的标签进行预测： $y = f(x) + \epsilon$ ，其中 ϵ 是均值为 0 方差为 δ^2 的噪声。在一个机器学习任务中，我们希望通过学习算法找到一个模型 $\hat{f}(x)$ ，使其在训练样本内以及样本外都能尽量拟合上述模型 $f(x)$ 。举例来说，我们用均方误差来衡量泛化误差，即 $(y - \hat{f}(x))^2$ 最小。为了记号上的方便，我们令 $f = f(x)$ ， $\hat{f} = \hat{f}(x)$ 。

首先，对于任意的随机变量 X ，我们有：

$$\text{Var}[X] = E[X^2] - (E[X])^2.$$

真实模型 f 是确定的： $E[f] = f$ 。

由于给定了 $y = f(x) + \epsilon$ 和 $E[\epsilon] = 0$ ，我们可以得到： $E[y] = E[f + \epsilon] = E[f] = f$ 。

同理也有 $\text{Var}[\epsilon] = \delta^2$ ，因为 ϵ 和 \hat{f} 是独立的，我们可以作如下分解：

$$\begin{aligned} E[(y - \hat{f})^2] &= E[y^2 + \hat{f}^2 - 2y\hat{f}] \\ &= E[y^2] + E[\hat{f}^2] - E[2y\hat{f}] \\ &= \text{Var}[y] + E[y]^2 + \text{Var}[\hat{f}] + (E[\hat{f}])^2 - 2fE[\hat{f}] \\ &= \text{Var}[y] + \text{Var}[\hat{f}] + (f^2 - 2fE[\hat{f}] + (E[\hat{f}])^2) \\ &= \text{Var}[y] + \text{Var}[\hat{f}] + (f - E[\hat{f}])^2 \\ &= \sigma^2 + \text{Var}[\hat{f}] + \text{Bias}[\hat{f}]^2 \\ &= \text{Bias}[\hat{f}]^2 + \text{Var}[\hat{f}] + \sigma^2 \end{aligned}$$

其中第三项为标签自带的噪声，也叫系统误差，他是在当前任务上任何模型所能达到的期望泛化误差的下界。

我们可以用偏差来衡量学习算法的能力，偏差越小代表学习算法能力越强；可以用方差衡量样本数据自身的稳定性，即相同类型的数据分布是否更集中，当数据足够充分的时候，我们可以认为方差足够小；而可以用系统误差衡量问题本身的复杂程度，问题越复杂，样本就越难刻画，系统误差也就越大。因此，偏差-方差分解说明我们的泛化误差是由模型自身的能力，数据的质量以及问题的复杂度所共同决定的。

8.2.2 偏差-方差窘境

还需要指出的一个情况是，偏差和方差一般是有冲突的，即越低的偏差往往带来越高的方差，这称为偏差-方差困境 (bias-variance dilemma)。通常来说，当我们使用更复杂的模型时，学习算法都能让模型对数据拟合的更好，即偏差越低。但是对数据拟合的越好就意味着模型对于数据的微小变化更加敏感，从而导致预测的方差变高。而二者的相对变化速度，决定了总误差是在升高还是降低，如下图所示。一开始随着模型复杂度的升高，偏差急剧下降，但是方差只是缓缓上升，所以我们的总误差也在下降。当模型复杂度到某个临界点的时候，再增加模型复杂度只会导致偏差缓缓下降，而方差开始急剧上升，从而总误差也开始上升。因此，我们在设计模型的时候也需要考虑这个因素。

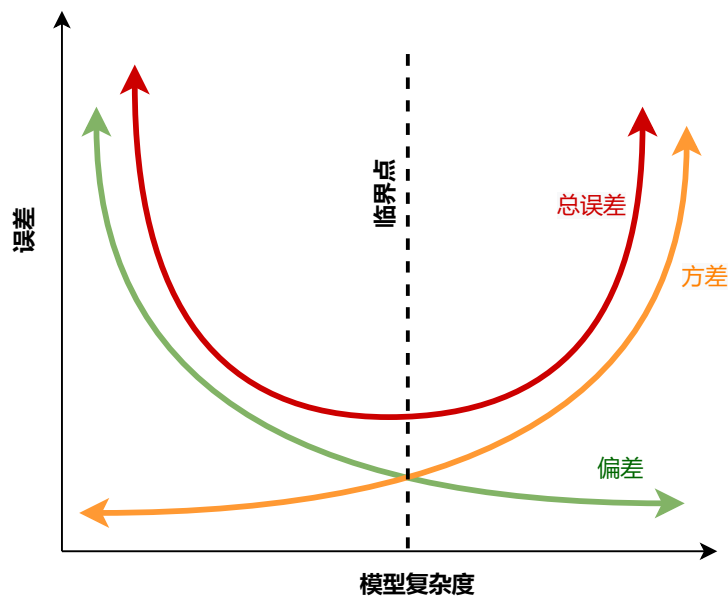


图 8-2 偏差-方差困境

8.2.3 模型选择

对于结构风险最小化 (SRM) 模型

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i)) + \lambda J(f)$$

我们可以把 λ 看作权衡偏差和方差的变量，当 λ 越高，正则化强度越大，偏差越高方差越低，反之亦然。那么，如何确定 λ 使得模型正好处于偏差-方差困境中的最优位置呢？

数据划分：首先为了能近似评估模型的泛化误差，我们需要将数据集划分为训练集 (training set) 和验证集 (validation set)。对于不同的 λ ，我们在训练集上训练模型，并且在验证集上进行验证，将验证集的误差当作泛化误差。

有许多划分数据集的方式，其中最常见的是 K-折交叉验证。

将数据集分成 k 个集合，并且在其中 $k - 1$ 个训练集上进行训练，并且保留一个集合作为验证集。总共会运行 k 次（即每个集合都会当做验证集运行一次），之后再平均每次运行的验证误差，这样可以很好的估计验证错误（甚至是标准差）。在极端情况下，可以令 $k = n$ ，即每次只会剩下一个单独的数据点（也就是留一法）。留一法对于小的数据集有很好的表现。

参数搜索：现在我们可以对泛化误差进行评估了，怎么确定最优的 λ 呢？有以下三个常见的搜索策略：

- ▶ 迭代搜索。进行两步搜索：首先，寻找 λ 的最佳数量级。其次，在刚刚寻找到的 λ 的“附近”进行搜索。比如：第一步我们尝试 $\lambda = 0.01, 0.1, 1, 10, 100$ 。得到当 $\lambda = 10$ 的时候表现最佳。第二步就是搜索 $\lambda = 10$ 的“周围”也就是 $\lambda = 5, 10, 15, 20, 25, \dots, 95$ 。
- ▶ 网格搜索。如果我们有多个参数来调控 λ ，我们需要确认每个参数的最佳位置。一个简单的方式是确定每个参数的搜索集合，从而遍历测试每一种参数组合，从而找到最优的组合。然而这样做会导致搜索空间随参数数量的增加而指数级增长，而且容易对一些不重要的参数浪费过多计算资源。
- ▶ 随机搜索。相对于网格搜索对每个参数的搜索空间有一个固定的间隔，随机搜索可以在搜索空间中任意选择某个值进行验证。

另外，在优化过程还没有完全收敛时结束训练，即早停 (early stopping)，也可以控制学习算法的复杂度，权衡偏差和方差。

9

核方法

线性分类器的效果很好，但是如果不存在线性决策边界呢？事实上，有一种可以将非线性合并到大多数线性分类器中的好方法。

9.1 手动扩展特征

通过在输入特征向量上应用基函数 (特征变换)，可以使线性分类器非线性化。对数据向量 $\mathbf{x} \in \mathbb{R}^d$ ，我们应用变换 $\mathbf{x} \rightarrow \phi(\mathbf{x}), \phi(\mathbf{x}) \in \mathbb{R}^D$ 。因为我们是通过添加维度的方式捕捉非线性特性之间的相关性，所以通常 $D \gg d$ 。

这种方法的优点是它的操作简单，并且可以保证问题仍然是凸优化问题。所以我们仍然可以使用普通的梯度下降代码，只不过处理的表征维度更高了。缺点也很明显， $\phi(\mathbf{x})$ 的维度可能过高。我们考虑以下这个例子：

$$\text{有 } \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{pmatrix}, \text{ 并且定义 } \phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}$$

通过将 $\phi(\mathbf{x})$ 映射到高维空间，把低维空间的非线性可分问题，转化为高维空间的线性可分问题，这种方法就叫做核方法。

思考：在这个例子中， $\phi(\mathbf{x})$ 的维度是多少？

对于 \mathbf{x} 中的每个元素，只有选择和不选择两个选项，所以 $\phi(\mathbf{x})$ 的维度是 2^d 。

拥有更多维度的新向量 $\phi(\mathbf{x})$ 的表达能力非常强，使得复杂的非线性边界可以存在。但过多的维度也会使得算法的效率严重下降，这是我们所不能接受的。

9.2 核技巧

核技巧是权衡表达能力和计算复杂度的一种方式，它是基于以下观察：如果我们使用任意标准的损失函数进行梯度下降，得到的梯度会是输入样本的线性组合。我们来具体看一个例子。

9.2.1 平方损失的梯度下降

以平方损失为例：

$$\ell(\mathbf{w}) = \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$$

我们用步长为 $s > 0$ 的梯度下降更新 \mathbf{w} ：

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - s \left(\frac{\partial \ell}{\partial \mathbf{w}} \right) \text{ where: } \frac{\partial \ell}{\partial \mathbf{w}} = \sum_{i=1}^n \underbrace{2(\mathbf{w}^\top \mathbf{x}_i - y_i) \mathbf{x}_i}_{\gamma_i \text{ 是 } \mathbf{x}_i, y_i \text{ 的函数}} = \sum_{i=1}^n \gamma_i \mathbf{x}_i$$

现在我们要证明我们可以把 \mathbf{w} 表示成所有输入向量的线性组合，

$$\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$$

因为损失函数是凸的，且最后的结果不依赖于初始值，所以我们可以将 \mathbf{w}_0 初始化为任意我们需要的。为了方便，我们初始化 $\mathbf{w}_0 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$ 。对于 \mathbf{w}_0 来说 $\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$ 的线性组合明显为 $\alpha_1 = \dots = \alpha_n = 0$ 。现在我们证明在整个梯度下降的过程中，系数 $\alpha_1, \dots, \alpha_n$ 始终存在，因为我们可以根据更新系数 α_i 重写整个梯度下降过程：

$$\begin{aligned}
 \mathbf{w}_1 &= \mathbf{w}_0 - s \sum_{i=1}^n 2 (\mathbf{w}_0^\top \mathbf{x}_i - y_i) \mathbf{x}_i = \sum_{i=1}^n \alpha_i^0 \mathbf{x}_i - s \sum_{i=1}^n \gamma_i^0 \mathbf{x}_i = \sum_{i=1}^n \alpha_i^1 \mathbf{x}_i & (\text{with } \alpha_i^1 &= \alpha_i^0 - s\gamma_i^0) \\
 \mathbf{w}_2 &= \mathbf{w}_1 - s \sum_{i=1}^n 2 (\mathbf{w}_1^\top \mathbf{x}_i - y_i) \mathbf{x}_i = \sum_{i=1}^n \alpha_i^1 \mathbf{x}_i - s \sum_{i=1}^n \gamma_i^1 \mathbf{x}_i = \sum_{i=1}^n \alpha_i^2 \mathbf{x}_i & (\text{with } \alpha_i^2 &= \alpha_i^1 - s\gamma_i^1) \\
 \mathbf{w}_3 &= \mathbf{w}_2 - s \sum_{i=1}^n 2 (\mathbf{w}_2^\top \mathbf{x}_i - y_i) \mathbf{x}_i = \sum_{i=1}^n \alpha_i^2 \mathbf{x}_i - s \sum_{i=1}^n \gamma_i^2 \mathbf{x}_i = \sum_{i=1}^n \alpha_i^3 \mathbf{x}_i & (\text{with } \alpha_i^3 &= \alpha_i^2 - s\gamma_i^2) \\
 \dots & \dots & \dots & \\
 \mathbf{w}_t &= \mathbf{w}_{t-1} - s \sum_{i=1}^n 2 (\mathbf{w}_{t-1}^\top \mathbf{x}_i - y_i) \mathbf{x}_i = \sum_{i=1}^n \alpha_i^{t-1} \mathbf{x}_i - s \sum_{i=1}^n \gamma_i^{t-1} \mathbf{x}_i = \sum_{i=1}^n \alpha_i^t \mathbf{x}_i & (\text{with } \alpha_i^t &= \alpha_i^{t-1} - s\gamma_i^{t-1})
 \end{aligned}$$

α_i^t 更新方法为:

$$\alpha_i^t = \alpha_i^{t-1} - s\gamma_i^{t-1}$$

考虑到 $\alpha_1 = \dots = \alpha_n = 0$, 我们有

$$\alpha_i^t = -s \sum_{r=0}^{t-1} \gamma_i^r$$

换句话说, 我们可以在不显式表达 \mathbf{w} 的情况下执行整个梯度下降更新算法。既然 \mathbf{w} 可以写成所有输入向量的线性组合, 我们也可以用所有输入向量之间的内积来表示 \mathbf{w} 与任意输入向量 \mathbf{x}_i 的内积:

$$\mathbf{w}^\top \mathbf{x}_j = \sum_{i=1}^n \alpha_i \mathbf{x}_i^\top \mathbf{x}_j$$

因此, 我们可以将平方损失函数 $\ell(\mathbf{w}) = \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$ 以输入向量内积的形式重写为:

$$\ell(\alpha) = \sum_{i=1}^n \left(\sum_{j=1}^n \alpha_j \mathbf{x}_j^\top \mathbf{x}_i - y_i \right)^2$$

在测试时, 我们也只需要这些系数就可以对任意测试输入向量 \mathbf{x}_t 进行预测, 因此可以用测试点和训练点之间的内积来表示整个分类器:

$$h(\mathbf{x}_t) = \mathbf{w}^\top \mathbf{x}_t = \sum_{j=1}^n \alpha_j \mathbf{x}_j^\top \mathbf{x}_t$$

此时我们可以发现, 如果要建立具有平方损失的超平面分类器, 我们需要的唯一信息是所有训练数据向量之间的内积。

9.2.2 内积计算

回到刚开始的例子 $\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1x_2 \\ \vdots \\ x_{d-1}x_d \\ \vdots \\ x_1x_2 \cdots x_d \end{pmatrix}$, 内积 $\phi(\mathbf{x})^\top \phi(\mathbf{z})$ 可以被形式化为:

$$\phi(\mathbf{x})^\top \phi(\mathbf{z}) = 1 \cdot 1 + x_1z_1 + x_2z_2 + \cdots + x_1x_2z_1z_2 + \cdots + x_1 \cdots x_dz_1 \cdots z_d = \prod_{k=1}^d (1 + x_kz_k)$$

可以看到, 这里把 2^d 项的和变成了 d 项的乘积, 从而把计算内积的时间从 $O(2^d)$ 变成了 $O(d)$! 事实上, 我们可以定义核函数:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

我们可以预先计算所有训练数据的核函数并将它们存储在一个核矩阵中:

$$K_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

利用这个核矩阵 \mathbf{K} , 我们在整个梯度下降算法中只需要进行简单的内积查找和在低维度空间上进行计算就好。最终的分类器可以表示成:

$$h(\mathbf{x}_t) = \sum_{j=1}^n \alpha_j k(\mathbf{x}_j, \mathbf{x}_t)$$

那么如何通过训练估计参数呢? 我们考虑通过利用高维空间中的 $\phi(\mathbf{x})$ 来计算 γ_i , 进而避免计算任何 $\phi(\mathbf{x}_i)$ 甚至是 \mathbf{w} 。因为通过上一节的推导我们已经得到 $\mathbf{w} = \sum_{j=1}^n \alpha_j \phi(\mathbf{x}_j)$ 和 $\gamma_i = 2(\mathbf{w}^\top \phi(\mathbf{x}_i) - y_i)$ 。可以容易得到 $\gamma_i = 2(\sum_{j=1}^n \alpha_j K_{ij} - y_i)$, 所以第 $t+1$ 步的梯度更新可以表示为:

$$\alpha_i^{t+1} \leftarrow \alpha_i^t - 2s(\sum_{j=1}^n \alpha_j^t K_{ij} - y_i).$$

因为有 n 个样本, 所以我们需要像这样进行 n 次计算, 每步计算时的复杂度是 $O(n^2)$, 远优于之前的 $O(2^d)$ 。像这样利用核函数直接计算, 从而加速核方法计算的技巧的就叫做核技巧。

9.3 常见核函数

下面介绍一些常见的核函数:

- ▶ 线性核: $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$ 。线性核函数等价于使用一个普通的线性分类器, 但是如果数据的维数 d 很高, 使用核函数矩阵会更快。
- ▶ 多项式核: $k(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^\top \mathbf{z})^d$ 。
- ▶ RBF 核 (径向基函数, 也称作高斯核): $k(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{\sigma^2})$ 。RBF 核是最常用的核, 它是一个通用逼近器, 因为其对应的特征向量是无限维的。
- ▶ 指数核: $k(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\|\mathbf{x}-\mathbf{z}\|}{2\sigma^2})$ 。
- ▶ 拉普拉斯核: $k(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\|\mathbf{x}-\mathbf{z}\|}{\sigma})$ 。
- ▶ Sigmoid 核: $k(\mathbf{x}, \mathbf{z}) = \tanh(\mathbf{a}\mathbf{x}^\top + c)$ 。

思考: $K(\cdot, \cdot)$ 可以是任意矩阵吗?

不, 这个矩阵 $K(x_i, x_j)$ 表示的是变换 $\mathbf{x} \rightarrow \phi(\mathbf{x})$ 后的真实内积, 所以 K 能作为核矩阵的条件是当且仅当 K 是半正定的情况。我们知道, 矩阵 $A \in \mathbb{R}^{n \times n}$ 为半正定, 当且仅当 $\forall \mathbf{q} \in \mathbb{R}^n, \mathbf{q}^\top A \mathbf{q} \geq 0$

证明. 已知, $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$ 。设 $\Phi = [\phi(\mathbf{x}_1), \dots, \phi(\mathbf{x}_n)]^\top$, 则 $K = \Phi \Phi^\top$ 。

所以: $\mathbf{q}^\top K \mathbf{q} = (\Phi^\top \mathbf{q})^2 \geq 0$ 。

这也就是说, 如果一个矩阵 A 是半正定的, 那么他肯定可以被分解为 $A = \Phi \Phi^\top$ 的形式, 即可以作为一种核矩阵。□

9.4 定义良好的核

我们在这里研究如下三个最常见的核:

- ▶ 线性核: $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$
- ▶ RBF 核: $k(\mathbf{x}, \mathbf{z}) = \exp(-\frac{(\mathbf{x}-\mathbf{z})^2}{\sigma^2})$
- ▶ 多项式核: $k(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^\top \mathbf{z})^d$

由一个或者多个下面的这些规则结合而成的核被称为定义良好的核 (well-defined kernels):

1. $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$
2. $k(\mathbf{x}, \mathbf{z}) = c k_1(\mathbf{x}, \mathbf{z})$
3. $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$
4. $k(\mathbf{x}, \mathbf{z}) = g(k_1(\mathbf{x}, \mathbf{z}))$
5. $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) k_2(\mathbf{x}, \mathbf{z})$

$$6. k(\mathbf{x}, \mathbf{z}) = f(\mathbf{x})k_1(\mathbf{x}, \mathbf{z})f(\mathbf{z})$$

$$7. k(\mathbf{x}, \mathbf{z}) = \exp(k_1(\mathbf{x}, \mathbf{z}))$$

$$8. k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{A} \mathbf{z}$$

其中 k_1, k_2 是定义良好的核, $c \geq 0$, g 是一个正系数的多项式函数, f 可以是任意函数, $\mathbf{A} \geq 0$ 是一个半正定矩阵。一个核是定义良好的等价于它的核矩阵 K 是半正定的 (此处不证明), 即等价于下列任意一条:

1. K 的所有特征值非负
2. 存在实矩阵 P , 满足 $K = P^\top P$
3. 对于任意的实向量 \mathbf{x} , $\mathbf{x}^\top K \mathbf{x} \geq 0$.

易证, 线性核和多项式核都是定义良好的核。

定理 2 RBF 核是一个定义良好的核。

证明.

$k_1(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$ 是定义良好的 (由规则 1 可知),

$k_2(\mathbf{x}, \mathbf{z}) = \frac{2}{\sigma^2} k_1(\mathbf{x}, \mathbf{z}) = \frac{2}{\sigma^2} \mathbf{x}^\top \mathbf{z}$ 是定义良好的 (由规则 2 可知),

$k_3(\mathbf{x}, \mathbf{z}) = \exp(k_2(\mathbf{x}, \mathbf{z})) = \exp(\frac{2\mathbf{x}^\top \mathbf{z}}{\sigma^2})$ 是定义良好的 (由规则 7 可知),

$k_4(\mathbf{x}, \mathbf{z}) = \exp(-\frac{\mathbf{x}^\top \mathbf{x}}{\sigma^2}) k_3(\mathbf{x}, \mathbf{z}) \exp(-\frac{\mathbf{z}^\top \mathbf{z}}{\sigma^2}) = \exp(-\frac{\mathbf{x}^\top \mathbf{x}}{\sigma^2}) \exp(\frac{2\mathbf{x}^\top \mathbf{z}}{\sigma^2}) \exp(-\frac{\mathbf{z}^\top \mathbf{z}}{\sigma^2})$
 $= \exp(\frac{-\mathbf{x}^\top \mathbf{x} + 2\mathbf{x}^\top \mathbf{z} - \mathbf{z}^\top \mathbf{z}}{\sigma^2}) = \exp(\frac{-(\mathbf{x} - \mathbf{z})^\top (\mathbf{x} - \mathbf{z})}{\sigma^2}) = k_{RBF}(\mathbf{x}, \mathbf{z})$ 是定义良好的 (由规则 6 可知)。

□

核还可以定义在集合上。比如接下来的定理:

定理 3 下面定义在任意两个集合 $S_1, S_2 \subseteq \Omega$ 上的核是定义良好的,

$$k(S_1, S_2) = e^{|S_1 \cap S_2|}.$$

证明. 列出样本空间 Ω 的所有样本, 我们定义一个向量 $\mathbf{x}_S \in \{0, 1\}^{|\Omega|}$, 向量的每一个元素表示对应样本是否属于集合 S 。显然,

$$k(S_1, S_2) = e^{\mathbf{x}_{S_1}^\top \mathbf{x}_{S_2}},$$

而这个核是定义良好的。

□

9.5 核机器学习

我们可以通过三步来核化一个算法:

1. 证明解在样本点张成的线性空间里 (即存在 α_i 使得 $\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$);

2. 重写算法，将所有样本点 \mathbf{x}_i 输入写成内积形式的输入，比如 $\mathbf{x}_i^\top \mathbf{x}_j$ 。
3. 定义核函数，用 $k(\mathbf{x}_i, \mathbf{x}_j)$ 代替 $\mathbf{x}_i^\top \mathbf{x}_j$

9.5.1 核回归

回顾

回顾一下，线性回归的形式是通过优化下面的平方损失函数，

$$\min_{\mathbf{w}} \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i - y_i)^2,$$

来找到超平面 \mathbf{w} 。在测试样本上的预测结果为 $h(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ 。如果我们令 $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$, $\mathbf{y} = [y_1, \dots, y_n]^\top$ ，那么上述问题有闭式解：

$$\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}.$$

核化

下面我们来尝试核化这个算法，我们将解 \mathbf{w} 表示为样本点的线性组合

$$\mathbf{w} = \sum_{i=1}^n \alpha_i \mathbf{x}_i = \mathbf{X}\vec{\alpha}.$$

由上一章可知，通过观察梯度下降得知这样的 $\vec{\alpha}$ 确实存在。

类似地，对于测试函数，我们可以得到，

$$h(\mathbf{z}) = \mathbf{w}^\top \mathbf{z} = \sum_{i=1}^n \alpha_i \mathbf{x}_i^\top \mathbf{z}.$$

我们可以通过用 $k(\mathbf{x}, \mathbf{z})$ 来代替 $\mathbf{x}^\top \mathbf{z}$ 来核化这个算法，同样我们也可以得到 $\vec{\alpha}$ 的闭式解。

定理 4 核化的线性回归的解为 $\vec{\alpha} = \mathbf{K}^{-1}\mathbf{y}$

证明. 由 \mathbf{w} 的表达式可得，

$$\mathbf{X}\vec{\alpha} = \mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}$$

将其左乘 $\mathbf{X}^\top \mathbf{X} \mathbf{X}^\top$ 得，

$$(\mathbf{X}^\top \mathbf{X})(\mathbf{X}^\top \mathbf{X})\vec{\alpha} = \mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top (\mathbf{X}\mathbf{X}^\top)^{-1})\mathbf{X}\mathbf{y}$$

将 $\mathbf{K} = \mathbf{X}^\top \mathbf{X}$ 带入得，

$$\mathbf{K}^2 \vec{\alpha} = \mathbf{K}\mathbf{y}$$

将其左乘 $(\mathbf{K}^{-1})^2$ 得,

$$\vec{\alpha} = \mathbf{K}^{-1}\mathbf{y}$$

即得证。 □

我们可以将核化的线性回归扩展到岭回归, 其解变为

$$\vec{\alpha} = (\mathbf{K} + \tau^2 \mathbf{I})^{-1}\mathbf{y}.$$

实际中, $\tau^2 > 0$ 是稳定增长的, 特别是当 \mathbf{K} 不可逆时。当 $\tau^2 = 0$ 时岭回归变成了核化的线性回归。通常我们也将核化的岭回归看作是核回归。

对于测试样本, 我们已经定义 $\mathbf{w} = \mathbf{X}\vec{\alpha}$ 。对于测试点 \mathbf{z} , 其预测值为

$$h(\mathbf{z}) = \mathbf{z}^\top \mathbf{w} = \mathbf{z}^\top \underbrace{\mathbf{X}\vec{\alpha}}_{\mathbf{w}} = \underbrace{\mathbf{k}_*}_{\mathbf{z}^\top \mathbf{X}} \underbrace{(\mathbf{K} + \tau^2 \mathbf{I})^{-1}\mathbf{y}}_{\vec{\alpha}} = \mathbf{k}_* \vec{\alpha},$$

其中 \mathbf{k}_* 是测试点与训练点之间的核向量, 第 i 项对应 $[\mathbf{k}_*]_i = \phi(\mathbf{z})^\top \phi(\mathbf{x}_i)$, 即测试点与训练点的内积通过 ϕ 映射到特征空间。

9.5.2 核化最近邻算法

练习: 令 $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$ 。怎样核化最近邻算法 (欧式距离)?

9.5.3 核支持向量机

回顾

支持向量机 (SVM) 的原始形式是一个二次规划问题:

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \mathbf{w}^\top \mathbf{w} + C \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \xi_i \\ & \xi_i \geq 0, \end{aligned}$$

其对偶形式为:

$$\begin{aligned} \min_{\alpha_1, \dots, \alpha_n} \quad & \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j K_{ij} - \sum_{i=1}^n \alpha_i \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^n \alpha_i y_i = 0, \end{aligned}$$

其中 $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i)$ ，对测试样本 \mathbf{x} 我们有：

$$h(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b \right). \quad (9.1)$$

支持向量

对于支持向量前面的章节已经作出了解释，支持向量机中只有支持向量满足下面的等式约束：

$$y_i(\mathbf{w}^\top \phi(\mathbf{x}_i) + b) = 1.$$

在对偶形式中，这些支持向量对应的对偶形式，其满足 $\alpha_i > 0$ (其他输入有 $\alpha_i = 0$)。所以在测试环节我们只需要计算支持向量对应的和，训练完后 $\alpha_i = 0$ 的项可以丢弃。

求解 b

我们可以注意到在对偶形式中 b 不再是优化的一部分，但是我们需要用它来进行分类。事实上，原始解和对偶解在这个问题中是等价的 (凸优化)。在对偶形式中，支持向量对应的 $\alpha_i > 0$ ，因此我们可以通过解下列的方程来得到 b

$$\begin{aligned} y_i(\mathbf{w}^\top \phi(\mathbf{x}_i) + b) &= 1 \\ y_i \left(\sum_j y_j \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) + b \right) &= 1 \\ y_i - \sum_j y_j \alpha_j k(\mathbf{x}_j, \mathbf{x}_i) &= b. \end{aligned}$$

我们可以通过支持向量来求解 b 。实际中从数值精度方面考虑，最好将多个支持向量求得的 b 取平均。

核支持向量机-更聪明的近邻分类器

回忆一下 K 近邻算法，对于二分类问题 ($y_i \in \{+1, -1\}$)，我们可以将测试点 \mathbf{z} 的决策函数 (预测函数) 写为

$$h(\mathbf{z}) = \text{sign} \left(\sum_{i=1}^n y_i \delta^{nn}(\mathbf{x}_i, \mathbf{z}) \right),$$

其中 $\delta^{nn}(\mathbf{z}, \mathbf{x}_i) \in \{0, 1\}$ ，当且仅当 \mathbf{x}_i 是 \mathbf{z} 的 k 近邻点时有 $\delta^{nn}(\mathbf{z}, \mathbf{x}_i) = 1$ 。SVM 的决策函数

$$h(\mathbf{z}) = \text{sign} \left(\sum_{i=1}^n y_i \alpha_i k(\mathbf{x}_i, \mathbf{z}) + b \right)$$

与上面的决策函数十分相似，但是不仅仅考虑 k 近邻，它考虑了所有的训练点，核函数会将那些离得近的点赋予更高的权重 (更大的 $k(\mathbf{z}, \mathbf{x}_i)$)。从某种程度上你可以将 RBF 核看作是一种软最近邻权重分配，因为它是一种指数衰减的权重赋予方法。

10

高斯过程

我们先来回顾高斯分布的定义和性质：服从多元高斯分布的 d 维随机向量 $X = (X_1, \dots, X_d)^\top$ 可以表示为： $X \sim \mathcal{N}(\mu, \Sigma)$ ，其中 μ 是均值， Σ 是协方差矩阵，其概率密度函数如下：

$$f_X(x_1, \dots, x_d) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|} \exp\left(-\frac{1}{2}((x - \mu)^\top \Sigma^{-1} (x - \mu))\right)$$

根据中心极限定理（CLT, the central limit theorem），高斯分布在实际场景下很普遍。中心极限定理指出，随着随机变量数量的增加，具有有限方差的独立的且相同分布的随机变量将趋于正态分布。

10.1 高斯分布的性质

令高斯随机变量 $Y = \begin{bmatrix} y_A \\ y_B \end{bmatrix}$ ，平均值 $\mu = \begin{bmatrix} \mu_A \\ \mu_B \end{bmatrix}$ ，以及协方差矩阵 $\Sigma = \begin{bmatrix} \Sigma_{AA} & \Sigma_{AB} \\ \Sigma_{BA} & \Sigma_{BB} \end{bmatrix}$ 我们有以下几个性质：

1. **正规性**： $\int_Y f(y; \mu, \Sigma) dy = 1$
2. **边缘分布**：高斯分布的边缘分布也是高斯分布。边缘分布 $f(y_A) = \int_{y_B} f(y_A, y_B; \mu, \Sigma) dy_B$ 和 $f(y_B) = \int_{y_A} f(y_A, y_B; \mu, \Sigma) dy_A$ ，分别满足： $y_A \sim \mathcal{N}(\mu_A, \Sigma_{AA})$ $y_B \sim \mathcal{N}(\mu_B, \Sigma_{BB})$
3. **和分布**：如果 $y \sim \mathcal{N}(\mu, \Sigma)$ 以及 $y' \sim \mathcal{N}(\mu', \Sigma')$ 那么： $y + y' \sim \mathcal{N}(\mu + \mu', \Sigma + \Sigma')$ 。
4. **条件分布**： y_A 在 y_B 上的条件分布

$$p(y_A|y_B) = \frac{p(y_A, y_B; \mu, \Sigma)}{\int_{y_A} p(y_A, y_B; \mu, \Sigma) dy_A}$$

也是高斯分布

$$(y_A | y_B = y_B) \sim \mathcal{N}(\mu_A + \Sigma_{AB} \Sigma_{BB}^{-1} (y_B - \mu_B), \Sigma_{AA} - \Sigma_{AB} \Sigma_{BB}^{-1} \Sigma_{BA})$$

在下文中的高斯过程，我们将会用到这一性质。

10.2 高斯过程回归

10.2.1 后验预测分布

首先我们考虑一个回归问题：

$$y = f(\mathbf{x}) + \epsilon$$

$$y = \mathbf{w}^T \mathbf{x} + \epsilon \quad (\text{普通线性回归})$$

$$y = \mathbf{w}^T \phi(\mathbf{x}) + \epsilon \quad (\text{核化的岭回归})$$

到目前为止，为从数据中学习概率分布，我们已经介绍了线性回归和核回归来学习概率分布函数，这些回归方法给出了一个用 \mathbf{w} 参数化的预测模型。即为：

$$P(Y | D, X) = \int_{\mathbf{w}} P(Y, \mathbf{w} | D, X) d\mathbf{w} = \int_{\mathbf{w}} P(Y | \mathbf{w}, D, X) P(\mathbf{w} | D) d\mathbf{w}$$

然而，上述回归方法往往无法得到一个闭式解。但是，对于具有高斯似然和先验的形式下 (即岭回归假设)，由于概率分布保持高斯分布，我们可以推导出它的均值和协方差。

$$P(y_* | D, \mathbf{x}) \sim \mathcal{N}(\mu_{y_*|D}, \Sigma_{y_*|D})$$

其中，

$$\mu_{y_*|D} = K_*^T (K + \sigma^2 I)^{-1} y, \Sigma_{y_*|D} = K_{**} - K_*^T (K + \sigma^2 I)^{-1} K_*$$

因此，我们不需要类似在岭回归中最大后验估计的方式，而是对整个分布建模，即直接对 f 建模 (而不是 y)。

10.2.2 高斯过程

问题： f 是一个无限维度的函数，而多元高斯分布是针对有限维随机向量的，我们如何用有限维分布建模无限维函数？

通过高斯过程 (Gaussian Process, GP) 方法，我们可以尝试解决这个问题。高斯过程是一个随机变量的集合 (可以无穷大)，而该集合中每个有限子集的联合分布都服从多元高斯分布，即为：

$$f \sim GP(\mu, k)$$

其中 $\mu(\mathbf{x})$ 和 $k(\mathbf{x}, \mathbf{x}')$ 是相应的均值和协方差函数。通过贝叶斯方法，我们从而可以建模这个预测分布 $P(f_* | \mathbf{x}_*, D)$ ，首先使用高斯分布作为先验，将： $P(f | \mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$ ，然后把他作为条件分布并作用在训练集 D 上来建模函数分布： $f = f(X)$ （训练集）和 $f_* = f(\mathbf{x}_*)$ （测试集）。

10.2.3 高斯过程回归

首先我们假设数据集的标签 y 是由零均值的高斯分布先验得到的，而这总可以通过减去样本均值得到。即样本标签分布服从：

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+m} \end{bmatrix} \sim \mathcal{N}(0, \Sigma)$$

所有的训练和测试标签都是从 $(n + m)$ 维高斯分布中抽取的，其中 n 为训练点数， m 为测试点数。需要注意的是，真正的训练标签是 y_1, \dots, y_n ，而我们观察到的是标签为 Y_1, \dots, Y_n 的样本。

对协方差矩阵 Σ ，有以下性质：

1. $\Sigma_{ij} = E((Y_i - \mu_i)(Y_j - \mu_j))$
2. Σ 总是半正定
3. $\Sigma_{ii} = \text{Variance}(Y_i)$ ，因此 $\Sigma_{ii} \geq 0$
4. 如果 Y_i 和 Y_j 是相互独立的，例如 \mathbf{x}_i 不等于 \mathbf{x}_j 时，则 $\Sigma_{ij} = \Sigma_{ji} = 0$
5. 如果 \mathbf{x}_i 类似于 \mathbf{x}_j ，那么 $\Sigma_{ij} = \Sigma_{ji} > 0$

我们可以看出，这些性质和之前定义的核矩阵是非常相似的。因此，我们可以令 $\Sigma_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ 。例如，如果使用 RBF 核，协方差矩阵有下面的形式：

$$\Sigma_{ij} = \tau e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}}.$$

并且我们可以将 Σ 分解为 $\begin{pmatrix} K, K_* \\ K_*^\top, K_{**} \end{pmatrix}$ ，其中 K 是训练核矩阵， K_* 是训练-测试核矩阵， K_*^\top 是测试-训练核矩阵， K_{**} 是测试核矩阵。那么在无噪声的情况下，预测分布 f （观察到训练集时的条件分布）可以表示成：

$$f_* | (Y_1 = y_1, \dots, Y_n = y_n, \mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{x}_t) \sim \mathcal{N}(K_*^\top K^{-1} y, K_{**} - K_*^\top K^{-1} K_*),$$

其中，核矩阵 K_*, K_{**}, K 是 $\mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{x}_*$ 的函数。

10.2.4 加性高斯噪声

以上分析是建立在标签没有噪声的情况下，然而在实际场景下，标签可能是有噪声的。但如果假设噪声样本间服从相互独立，且均值为零的高斯分布，那么测试样本标签服从 $\hat{Y}_i = f_i + \epsilon_i$ ，其中 f_i 是真实标签（该真实标签是未被观察下的潜在标签），噪声表示为 $\epsilon_i \sim \mathcal{N}(0, \sigma^2)$ 。在这种情况下，新的协方差矩阵变成 $\hat{\Sigma} = \Sigma + \sigma^2 \mathbf{I}$ 。

证明. 由于 $\mathbb{E}[\epsilon_i] = \mathbb{E}[\epsilon_j] = 0$ ，并且 ϵ_i 独立于其他所有随机变量。我们可以针对 $i \neq j$ 的非对角项推导出：

$$\hat{\Sigma}_{ij} = \mathbb{E}[(f_i + \epsilon_i)(f_j + \epsilon_j)] = \mathbb{E}[f_i f_j] + \mathbb{E}[f_i] \mathbb{E}[\epsilon_j] + \mathbb{E}[f_j] \mathbb{E}[\epsilon_i] + \mathbb{E}[\epsilon_i] \mathbb{E}[\epsilon_j] = \mathbb{E}[f_i f_j] = \Sigma_{ij}$$

对于 Σ 的对角线项，即 $i = j$ 的情况下，由于 $\mathbb{E}[\epsilon_i^2] = \sigma^2$ ，其中 ϵ_i 表示方差。我们可以得到：

$$\hat{\Sigma}_{ii} = \mathbb{E}[(f_i + \epsilon_i)^2] = \mathbb{E}[f_i^2] + 2\mathbb{E}[f_i] \mathbb{E}[\epsilon_i] + \mathbb{E}[\epsilon_i^2] = \mathbb{E}[f_i f_i] + \mathbb{E}[\epsilon_i^2] = \Sigma_{ii} + \sigma^2$$

于是有，成 $\hat{\Sigma} = \Sigma + \sigma^2 \mathbf{I}$ 。证毕。 \square

将更新后的协方差矩阵代入高斯过程后验分布，得到：

$$Y_* | (Y_1 = y_1, \dots, Y_n = y_n, \mathbf{x}_1, \dots, \mathbf{x}_n) \sim \mathcal{N}(K_*^\top (K + \sigma^2 I)^{-1} y, K_{**} + \sigma^2 I - K_*^\top (K + \sigma^2 I)^{-1} K_*).$$

在实际中往往带噪声的高斯过程更稳定，因为矩阵 $(K + \sigma^2 I)$ 总是可逆矩阵。因此，我们可以通过后验均值得到预测，并且顺道还能得到预测方差，将其作为对点预测的可置信度或（不）确定性的度量。

10.3 高斯过程的参数学习

10.3.1 边缘似然和超参数学习

虽然高斯过程回归是非参数模型，但协方差函数 K 有参数。这些参数即为超参数，需要从训练数据中学习（估计）。设 θ 是 K 的所有参数组成的向量，写作 $K = K_\theta$ ，我们可以通过最大化边缘似然概率 $P(y | X, \theta)$ 估计参数。

一旦有了边缘似然及其导数，就可以使用任何封装好的更新方式，如（随机）梯度下降，共轭梯度下降等，来最小化负对数边缘似然。注意，边缘似然在其参数下不是一个凸函数，其解很可能是一个局部极小值/极大值。为了使这个过程更加鲁棒，我们可以通过使用不同的初始化重新训练，并选择最低/最高的返回值。

10.3.2 协方差函数- 高斯过程模型的核心

选择正确的协方差（即核函数），高斯过程可以提升预测。选择协方差函数是高斯过程学习阶段的模型选择过程。有三种方法可以得到一个好的协方差函数：

1. 专家知识（但是很难获得）
2. 贝叶斯模型选择（积分形式难以优化）
3. 交叉验证（耗时，但易于实现）

我们可以常用 RBF 函数核形式的协方差函数，其好处是对每个特征维具有不同的尺度。

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^T M (\mathbf{x}-\mathbf{x}')} + \sigma_n^2 \delta_{\mathbf{x}, \mathbf{x}'}$$

其中， $M = \text{diag}(\ell)^{-2}$ ， σ_f^2 是信号方差， ℓ 是一个长度尺度参数的向量， σ_n^2 是噪声方差。

10.3.3 高斯过程总结

高斯过程回归具有以下性质：

- ▶ 高斯过程是一种简洁有效的机器学习方法。
- ▶ 很容易得到获得预测的（不）置信度。
- ▶ 对训练集较小的回归问题，高斯过程简单有效。
- ▶ 由于需要求逆计算，高斯过程的运行时间复杂度是 $O(n^3)$ ，因此当 $n \gg 0$ 时求解缓慢，所以对于 n 很大的时候使用稀疏高斯过程。
- ▶ 高斯过程在分类（非高斯似然）方面应用更广。
- ▶ 我们可以在回归中建立非高斯概率模型，并对计数数据（泊松分布）进行近似推理。
- ▶ 高斯过程实现的一些代码库：GPyTorch、GPML (MATLAB)、GPys、pyGPs 和 scikit-learn (Python)

10.4 贝叶斯全局优化

训练高斯过程回归，一个常用的方法则是贝叶斯全局优化。贝叶斯优化的优化目标是算法的超参数，使其在固定的验证数据集上预测准确。假设有 d 个超参数要调优，那么数据集则表示成 d 维向量 $\mathbf{x}_i \in \mathbb{R}^d$ ，其中每个训练点表示一个特定的超参数设置，标签 $y_i \in \mathbb{R}$ 则是验证误差。不要混淆，向量 \mathbf{x}_i 对应的是超参数设置，而不是数据。例如，

在一个具有多项式核的支持向量机中，有两个超参数：正规化常数 C （也经常写作 λ ）和多项式指数 p ，那么向量 \mathbf{x}_i 则为两维向量，其中各个维度分别表示相应的超参数数值。

首先，在随机的超参数设置下训练分类器，并在验证集上评估训练的分类器。 $\mathbf{x}_1, \dots, \mathbf{x}_m$ 带有标签 y_1, \dots, y_m 。然后，通过训练一个高斯过程来预测新的超参数设置 \mathbf{x}_t 下的验证错误 y_t 。因此，可以得到一个平均映射（即超参数与预测错误率的映射） $y_t = h(\mathbf{x}_t)$ 及映射方差 $v(\mathbf{x}_t)$ 。并且也可以探索新的超参数设置。最优的超参数是置信下限最低的超参数，即

$$\arg \min_{\mathbf{x}_t} h(\mathbf{x}_t) - \kappa \sqrt{v(\mathbf{x}_t)}.$$

其中常数 $\kappa > 0$ 权衡回归的预测效果与随之带来的高不确定度。 κ 的设置取决于对于方差的厌恶（方差高），或是改进当前最佳的激励。一个很小的 κ 会则为对当前点的更多利用（exploitation），而更大的 κ 则为激进探索（exploration）新的超参数。

Algorithm 11 贝叶斯全局优化算法

Input: 误差函数 \mathcal{A} ，训练集数量 m ，测试集数量 n ，常数 κ

Output: 超参数设置 $\mathbf{x}_{i_{best}}$

for $i = 1$ to m **do**

 随机取样 \mathbf{x}_i ，例如：合理范围内的均匀采样

 计算验证误差 $y_i = \mathcal{A}(\mathbf{x}_i)$

end for

for $i = m + 1$ to n **do**

 基于 $\mathbf{x}_1, \dots, \mathbf{x}_{i-1}$ 更新核 K

$\mathbf{x}_i = \arg \min_{\mathbf{x}_t} K_t^\top (K + \sigma^2 I)^{-1} y - \kappa \sqrt{K_{tt} + \sigma^2 I - K_t^\top (K + \sigma^2 I)^{-1} K_t}$

 计算验证误差 $y_i = \mathcal{A}(\mathbf{x}_i)$

end for

找到最好的超参数设置 $i_{best} = \arg \min_i \{y_1, \dots, y_n\}$

返回最好的超参数设置 $\mathbf{x}_{i_{best}}$

11

KD Trees

回顾 k 近邻算法，假定训练集中共有 n 个数据点，每个数据 $\mathbf{x}_s \in \mathbb{R}^d$ 。测试过程中，对于每个测试数据 \mathbf{x}_t ，都需要计算测试数据与训练集中所有数据点的距离 $dist(\mathbf{x}_t, \mathbf{x}_s)$ ，然后再找到距离最近的 k 个数据点，该过程的时间复杂度为 $O(nd)$ 。所以当训练集较大时， k 近邻算法将会变得非常慢。于是，考虑引入一个新的数据结构，来加速 k 近邻算法。

11.1 KD 树

KD 树 (k-dimensional tree) 是一种数据结构，将特征空间分割为多个子空间来存储数据，形如图 11-1 所示，二维的特征空间被划分为 12 个子空间。

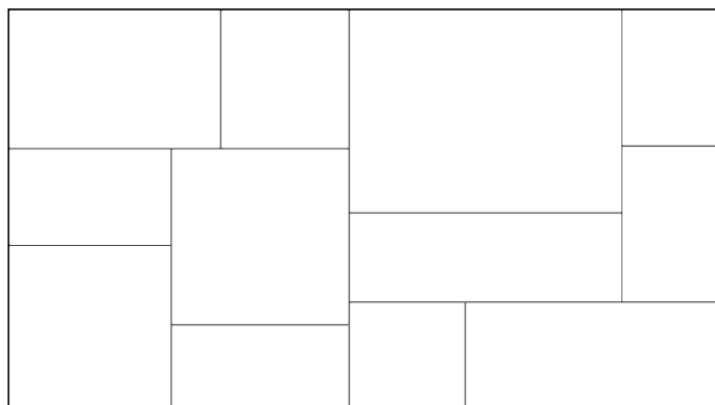


图 11-1 被 KD 树分割后的特征空间

11.1.1 创建 KD 树

KD 树的创建思想很简单，选取当前方差最大的某一维度，将均值作为分割阈值，将特征空间分为两个相同数据量的子空间，然后再对各个子空间继续递归分割.....

经过上述方法生成的 KD 树是一个平衡二叉树，形如图 11-2 所示。KD 树中由叶子节点存储数据，非叶子节点记录分割的维度以及阈值。

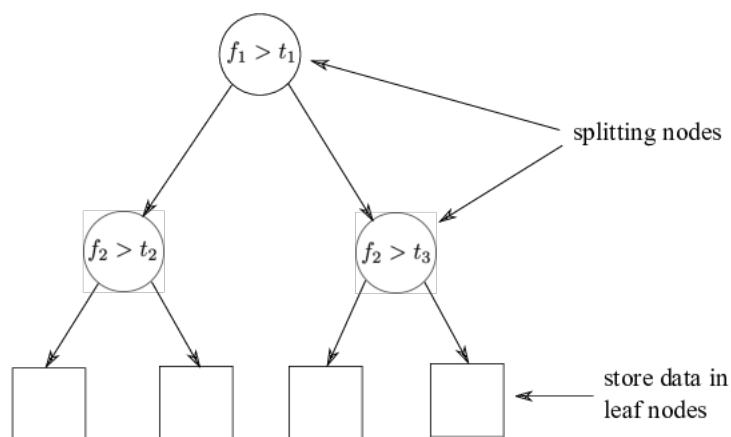


图 11-2 KD 树结构示意图

11.1.2 近邻搜索

根据几何关系，对 KD 树分割的子空间有定理 11.1.2

定理 11.1

某一子空间 \mathbb{S} 内所有数据点 \mathbf{x} ，与子空间外一数据点 \mathbf{x}_t 的距离 d_s ，均大于等于子空间边界 \mathbf{w} 与该数据点的距离 d_w 。即 $\forall \mathbf{x} \in \mathbb{S}, d_s \geq d_w$



证明. 如图 11-3 左半部分所示，子空间内任一点 \mathbf{x} 与子空间外一点 \mathbf{x}_t 的距离，可分为两部分 $d_s = d_1 + d_2$ ，子空间边界 \mathbf{w} 与 \mathbf{x}_t 的距离为 d_w 。由直角三角形的斜边大于直角边知

$$d_s = d_1 + d_2 \geq d_w + d_2 \geq d_w$$

定理得证。 □

根据定理 11.1.2，在应用 k 近邻算法时，可以忽略那些距离较远的子空间中的数据点，从而达到加速算法的目的。现以最近邻算法（即 k 近邻 $k = 1$ ）为例，描述使用带剪枝的深度优先策略，搜索 KD 树，寻找最近邻的过程：

输入测试数据为 \mathbf{x}_t

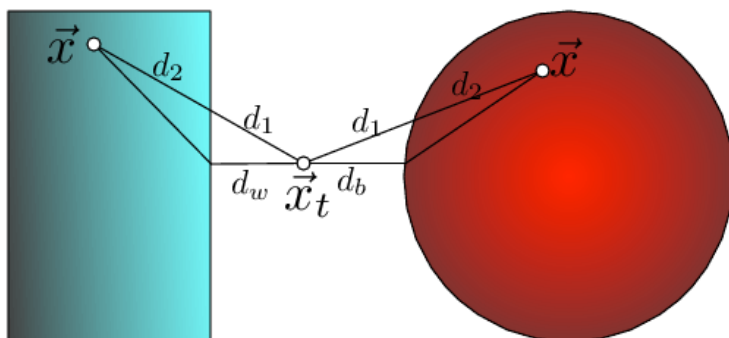


图 11-3 特征子空间的距离示意图

- 1 从 KD 树的根节点出发，找到 \mathbf{x}_t 所属叶子节点。
- 2 叶子节点对应子空间 \mathbb{S} ，计算 \mathbf{x}_t 与子空间 \mathbb{S} 中所有数据点的距离，并保存最近邻节点 \mathbf{x}_{min} 以及距离 d_{min} 。
- 3 返回到父节点，当前节点存储的分割边界为 \mathbf{w} ，若存在子树还未被访问，计算 \mathbf{w} 与 \mathbf{x}_t 的距离 d_w 。
- 4 若 $d_w < d_{min}$ （左子树还未访问，则 $<$ 替换为 \leq ），则计算未访问子树中所有叶子节点子空间中数据点与 \mathbf{x}_t ，若存在距离小于 d_{min} ，则更新 \mathbf{x}_{min} 和 d_{min} 。重复 2,3，直至返回到根节点。
- 5 输出最近邻节点 \mathbf{x}_{min} ，算法结束。

上述过程可以很容易扩展到 $k > 1$ ，只需用最大堆维护距离最近的 k 个数据点，若存在一数据点，与测试数据点的距离小于堆顶数据点，则将其送入堆中，并将堆顶数据点删除即可。

使用 KD 树进行 k 近邻搜索，最优的情况是，仅计算与一个块中数据点的距离，时间复杂度为 $O(\log n)$ ；最差的情况需要计算与所有数据点之间的距离，时间复杂度为 $O(n)$ 。

11.2 球树

球树 (Ball-trees) 与 KD 树相似，但是用高维球体代替了盒子，参看图 11-3。像以前一样，我们可以将距离分解并使用三角形不等式

$$d(x_t, x) = d_1 + d_2 \geq d_b + d_2 \geq d_b$$

如果目标点到球的距离 d_b 大于到当前最近邻的距离，我们就可以安全地舍弃掉这个球和球内的所有点。球树的球结构允许我们沿着数据点所在的底层流形分割数据，而不是

重复地分割整个特征空间。

11.2.1 创建球树

输入：集合 S , k

算法：见算法12

Algorithm 12 球树伪代码

```
1: procedure BALLTREE( $S, k$ )
2:   if  $|S| < k$  then 算法停止 end if
3:   随机选取  $\mathbf{x}_0 \in S$ 
4:   选取  $\mathbf{x}_1 = \arg \max_{\mathbf{x} \in S} d(\mathbf{x}_0, \mathbf{x})$ 
5:   选取  $\mathbf{x}_2 = \arg \max_{\mathbf{x} \in S} d(\mathbf{x}_1, \mathbf{x})$ 
6:    $\forall i = 1 \dots |S|, z_i = (\mathbf{x}_1 - \mathbf{x}_2)^\top \mathbf{x}_i \leftarrow$  将数据投影到  $(\mathbf{x}_1 - \mathbf{x}_2)$ 
7:    $m = \text{median}(z_1, \dots, z_{|S|})$ 
8:    $S_L = \{\mathbf{x} \in S : z_i < m\}$ 
9:    $S_R = \{\mathbf{x} \in S : z_i \geq m\}$ 
10:  Return 树:
    - 圆心:  $\mathbf{x} = \text{mean}(S)$ 
    - 半径:  $r = \max_{\mathbf{x} \in S} d(\mathbf{x}, \mathbf{x})$ 
    - 子树: Balltree( $S_L, k$ ) 和 Balltree( $S_R, k$ )
11: end procedure
```

注意：步骤 3、4 选取了一个大范围的方向 $(\mathbf{x}_1 - \mathbf{x}_2)$ 。

11.2.2 球树的应用场景

使用场景与 KD 树类似。

球树在低维 ($d \leq 3$) 情况下比 KD 树慢，然而在高维时远快于 KD 树。KD 树和球树都会受到维数灾难的影响。但是，如果数据存在局部结构（例如在一个低维流形上），球树在高维情况下还是可以工作。

11.3 总结

► kNN 在测试过程中很慢，因为它做了很多不必要的工作。

- ▶ **KD 树**对特征空间进行划分，这样我们就可以排除比最近 k 个邻居更远的整个分区。但是，分割是根据某一坐标轴的，不能很好地延伸到更高的维度。
- ▶ **球树**将数据点所在的流形分割，而不是整个空间。这使得它在更高的维度上表现得更好。

12

决策树

决策树是根据训练数据生成的树状结构，将特征空间划分为若干个内部输出一致的特征子空间，如图12-1a所示，特征空间被划分成了4个内部输出一致的特征子空间，决策树本质上是从训练数据中归纳出来的一组分类规则。

决策树由若干结点和有向边组成，结点分为两种：叶子结点和非叶子结点，叶子结点存储输出 c ，非叶子结点存储特征 k 以及阈值 t ，当前非叶子结点所代表的特征空间，按特征 k 的取值划分为两个子空间，结点左子树代表在特征 k 上取值 $< t$ 的部分，右子树代表在特征 k 上取值 $\geq t$ 的部分。如图12-1b所示，非叶子结点中存储特征和阈值，叶子结点存储输出标签。

使用决策树对实例 $x = [x^{(1)}, \dots, x^{(n)}]$ 进行预测，就是从根结点 (k_0, t_0) 开始，若 $x^{(k_0)} < t$ ，则进入左子树；若 $x^{(k_0)} \geq t$ ，则进入右子树，然后循环上述操作直到进入叶子结点 c ，则决策树对于该实例的预测结果就是 c 。



图 12-1 示意图

12.1 基本概念

假设数据集为 $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$, 属性集 $A = \{a_1, a_2, \dots, a_c\}$, $y_i \in A$, 共 n 个样本, 属于 c 个类别。

输入: 训练集 S , 属性集 A

输出: 以 $node$ 为根结点的一颗决策树

算法: TreeGenerate(S, A) 见算法13

Algorithm 13 决策树学习基本算法

```
生成结点 node
if S 中样本全属于同一类别 C then
    将 node 标记为 C 类叶结点
    return
end if
if A =  $\emptyset$  or S 中样本在 A 上取值相同 then
    将 node 标记为叶结点, 其类别标记为 S 中样本数最多的类
    return
end if
从 A 中选择最优划分属性  $a_*$ 
for  $a_*$  的每一个值  $a_*^v$  do
    为 node 生成一个分支, 令  $S_v$  表示 S 在  $a_*$  上取值为  $a_*^v$  的样本子集
    if  $S_v$  为空 then
        将分支结点标记为叶结点, 其类别标记为 S 中样本数最多的类
    else
        以 TreeGenerate( $S_v, A \setminus \{a_*\}$ ) 为分支结点
    end if
end for
end for
```

决策树学习的关键是如何选择最优划分属性, 一般而言, 随着划分过程的不断进行, 我们希望决策树的分支结点所包含的样本尽可能的属于同一个类别, 即结点的“纯度”(purity) 越来越高。

12.1.1 信息熵

信息熵 (information entropy) 是度量样本集合纯度最常用的一种指标, 假定样本集合 X 中第 i 类样本所占的比例为 $p_i (i = 1, 2, 3, \dots, |c|)$, 则 X 的信息熵定义为

$$H(X) = - \sum_{i=1}^{|c|} p_i \log p_i \quad (12.1)$$

其中:

$$p_i = \frac{|S_i|}{|S|} \quad (12.2)$$

$H(X)$ 的值越小, 则 X 的纯度越高。由此可以定义树或者子树的熵:

$$H(S) = p^L H(S^L) + p^R H(S^R) \quad (12.3)$$

其中, p^L 和 p^R 分别为 S 的左右子树的输入占 S 的比例:

$$p^L = \frac{|S^L|}{|S|}, p^R = \frac{|S^R|}{|S|} \quad (12.4)$$

12.1.2 信息增益

假定离散属性 a 有 V 个可能的取值 a^1, a^2, \dots, a^V , 若使用 a 来对样本集 X 进行划分, 则会产生 V 个分支结点, 其中第 V 个分支结点包含了 X 中所有在属性 a 上取值为 a^V 的样本, 记作 X^V 。我们可以根据上述式子计算出 X^V 的信息熵, 再考虑到不同的分支结点所包含的样本数不同, 给每个分支结点赋予权重 $|X^V|/|X|$, 即样本数越多的分支结点的影响越大, 于是可计算出用属性 a 对样本集 X 进行划分所获得的“信息增益”, 其定义如下:

$$H(X, a) = H(X) - \sum_{v=1}^V \frac{|X^v|}{|X|} H(X^v) \quad (12.5)$$

一般而言, 信息增益越大, 则意味着使用属性 a 来进行划分所获得的“纯度提升”越大。因此, 我们可以用信息增益来进行决策树的划分属性选择, 例如在每次划分时选择属性 $a_* = \arg \max_{a \in A} H(X, a)$ 。著名的 ID3 决策树学习算法就是以信息增益为准则来划分属性。

12.1.3 增益率

事实上，信息增益准则对可取数值数目较多的属性有所偏好（比如，取标号作为一个候选划分属性，那么每个分支结点仅包含一个样本，分支结点纯度达到最大，由此其信息增益将远远大于其他属性），为减少这种偏好可能带来的不利影响，我们可以使用“增益率” (gain ratio) 来选择最优划分属性，采用与式 (12.5) 相同的符号表示，增益率定义为：

$$g_R(X, a) = \frac{H(X, a)}{H_a(X)} \quad (12.6)$$

其中：

$$H_a(X) = - \sum_{v=1}^V \frac{|X^v|}{|X|} \log_2 \frac{|X^v|}{|X|} \quad (12.7)$$

称为属性 a 的“固有价值” (intrinsic value)。属性 a 的可能取值数目越多 (即 V 越大)，则其固有价值通常越大。

C4.5 算法采用增益率来选择最优划分属性，需要注意的是，增益率准则对可取值数目较少的属性有所偏好，因此，C4.5 算法并不是直接选择增益率最大的候选划分属性，而是使用了一个启发式：先从候选划分属性中找出信息增益高于平均水平的属性，再从中选择增益率最高的。

12.1.4 基尼指数

属于第 k 类的样本子集为 $S_k = \{(x, y) \in S | y = k\}$ ，且有 $S = S_1 \cup \dots \cup S_c, \forall i, j \in Z^c$ 。那么根据极大似然，样本点属于第 k 类的概率为 $p_k = \frac{|S_k|}{|S|}$ 。其基尼指数为

$$G(S) = \sum_{k=1}^c p_k(1 - p_k) = 1 - \sum_{k=1}^c p_k^2 \quad (12.8)$$

如果数据集被分为两部分 S_1, S_2 ，那么整体的基尼指数为

$$G(S) = \frac{|S_1|}{|S|} G(S_1) + \frac{|S_2|}{|S|} G(S_2) \quad (12.9)$$

所以决策树的基尼指数为

$$G^T(S) = \frac{|S_L|}{|S|} G^T(S_L) + \frac{|S_R|}{|S|} G^T(S_R) \quad (12.10)$$

其中 G^T 表示树的基尼指数, S_L, S_R 表示左子树、右子树对应的数据集，有 $S = S_L \cup S_R$ 和 $S_L \cap S_R = \emptyset$ 。

基尼指数 $G(S)$ 表示数据集 S 的不确定性，基尼指数 $G^T(S)$ 表示按该决策树划分后数据集 S 的不确定性，基尼指数越大，表示数据集的不确定性越大，这一点与熵的性质相似，所以都可以用来近似的代表分类误差率。

12.2 ID3

Quinlan 在 1986 年提出的 ID3 是经典的决策树模型，ID3 的核心思想是，在决策树各个结点上用信息增益大小作为选择特征的标准，递归地构建决策树。

$$ID(S) : \begin{cases} \text{if } \exists \bar{y} \text{ s.t. } \forall (x, y) \in S, y = \bar{y} \Rightarrow \text{return leaf with } \bar{y} \\ \text{if } \exists \bar{x} \text{ s.t. } \forall (x, y) \in S, x = \bar{x} \Rightarrow \text{return leaf with } y \\ \text{common}(y : (x, y) \in S) \text{ or mean} \end{cases} \quad (12.11)$$

式子 (12.11) 表示 ID3 决策树生成算法终止的两种条件，第一种条件是当前数据子集中所有样本均是同一标签 y ，那么不再进一步分割数据子集，并在当前位置生成一个输出标签为 y 的叶子结点；第二种条件是当前数据子集中所有样本的输入值完全一样，对应的标签为 y ，那么同样不再进一步分割数据子集，并在当前位置生成一个输出标签为 y 的叶子结点。

每次划分选择信息增益最大的特征 f ，并确定相应的阈值 t ，将当前的数据集递归地划分为两部分：

$$\begin{cases} S^L = \{(x, y) \in S : x_f \leq t\} \\ S^R = \{(x, y) \in S : x_f \geq t\} \end{cases} \quad (12.12)$$

信息增益作为划分的特征选择依据，存在偏向于选取取值较多的特征，因为取值越多不确定性越大，自然信息熵偏高。针对这一问题，Quinlan 在 1993 年提出了改进的 C4.5，C4.5 决策树生成算法的流程基本与 ID3 相似，只不过构建决策树非叶子结点时，将信息增益比作为选择特征的标准。

12.3 CART

Beriman 等人在 1984 提出的分类与回归树 (classification and regression tree, CART) 模型，是应用非常广泛的决策树模型，既可以用于分类问题也可以用于回归问题。

分类问题和回归问题，递归构建决策树非叶子结点时，确定最优特征以及最优划分阈值的准则有所不同，对于分类问题，使用基尼指数作为的准则；对于回归问题，则使用平方误差 ($\sum_{(x_i, y_i) \in S} (y_i - f(x_i))^2$) 最小化准则。

CART 决策树生成算法大致流程是，首先，对于训练样本集 S ，在所有可能的特征和可能的划分阈值中，选取基尼指数最小的特征和划分阈值，作为最优特征和最优划分阈值；其次，为当前结点生成两个子结点，按照选取的最优特征和最优划分阈值，将训练样本集分给两个子结点；然后，在子结点中重复上述步骤，直至结点中的样本数量小于一定数目，或结点样本集的基尼指数小于一定值，则生成叶子结点并返回；最后，算法运行结束就得到了根据训练样本生成的决策树。

13

Bagging

13.1 Bagging

Bagging 是集成学习 Bootstrap Aggregating ((Breiman 96)) 的缩写, 它也称为装袋算法。简单来说, 就是通过使用 bootstrap 抽样得到若干不同的训练集, 以这些训练集分别建立模型, 即得到一系列的基分类器, 这些分类器由于来自不同的训练样本, 他们对同一测试集的预测效果不一样. 因此, Bagging 算法随后对基分类器的一系列预测结果进行投票 (分类问题) 和平均 (回归问题), 从而得到每一个测试集样本的最终预测结果, 这一集成后的结果往往是准确而稳定的.

13.1.1 bagging 减小方差

记偏差/方差的分解为:

$$\underbrace{\mathbb{E}[(h_D(x) - y)^2]}_{\text{Error}} = \underbrace{\mathbb{E}[(h_D(x) - \bar{h}(x))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}[(\bar{h}(x) - \bar{y}(x))^2]}_{\text{Bias}} + \underbrace{\mathbb{E}[(\bar{y}(x) - y(x))^2]}_{\text{Noise}}$$

我们的目标是减少方差项: $\mathbb{E}[(h_D(x) - \bar{h}(x))^2]$.

为此, 我们期望 $h_D \rightarrow \bar{h}$.

弱大数定律

弱大数定律表明, 对于服从独立同分布的随机变量 x_i , 其均值为 \bar{x} , 我们有,

$$\frac{1}{m} \sum_{i=1}^m x_i \rightarrow \bar{x} \text{ as } m \rightarrow \infty$$

将其应用于分类器: 假设我们有来自 P^n 的 m 个训练集 D_1, D_2, \dots, D_m 。在每一个训练集上单独训练一个分类器并取其平均值:

$$\hat{h} = \frac{1}{m} \sum_{i=1}^m h_{D_i} \rightarrow \bar{h} \quad \text{as } m \rightarrow \infty$$

我们将多个分类器的平均值称为分类器的集成

好消息: 如果有 $\hat{h} \rightarrow \bar{h}$, 方差分量的误差也必定随之消失, 如: $\mathbb{E}[(\hat{h}(x) - \bar{h}(x))^2] \rightarrow 0$

问题: 我们并没有 m 个数据集 D_1, \dots, D_m , 我们仅有一个数据集 D .

13.1.2 方法: Bagging(Bootstrap Aggregating)

通过在数据集 D 中有放回的均匀随机抽样, 来模拟分布 P 的采样。

i.e. 令 $Q(X, Y|D)$ 表示这样一个概率分布, 其均匀随机的从数据集 D 中, 抽取训练样本 (\mathbf{x}_i, y_i) 。更正式的说, $Q((\mathbf{x}_i, y_i)|D) = \frac{1}{n} \quad \forall (\mathbf{x}_i, y_i) \in D$ with $n = |D|$.

我们通过采样得到集合 $D_i \sim Q^n$, i.e. $|D_i| = n$, 并且 D_i 是从 $Q|D$ 中有放回的抽样得到的

Q: $\mathbb{E}[|D \cap D_i|]$ 是什么?

Bagged 分类器: $\hat{h}_D = \frac{1}{m} \sum_{i=1}^m h_{D_i}$

注意到: $\hat{h}_D = \frac{1}{m} \sum_{i=1}^m h_{D_i} \rightarrow \bar{h}$ (这里不能使用 W.L.L.N, 因为 W.L.L.N 仅仅满足于独立同分布的样本). 但在实际中, bagging 依旧非常有效的降低了方差

Analysis

虽然我们不能证明新的样本是独立同分布的, 但显然它们是来自原始分布 P 的。假设 P 是离散的, 并且在数据集上 $\Omega = x_1, \dots, x_N$, 有 $P(X = x_i) = p_i$ (N 非常的大) (为了简

化，从现在开始，我们不考虑标签)

$$\begin{aligned}
 Q(X = x_i) &= \sum_{k=1}^n \underbrace{\binom{n}{k} p_i^k (1 - p_i)^{n-k}}_{\text{Probability that are } k \text{ copies of } x_i \text{ in } D} \underbrace{\frac{k}{n}}_{\text{Probability pick one of these copies}} \\
 &= \frac{1}{n} \sum_{k=1}^n \underbrace{\binom{n}{k} p_i^k (1 - p_i)^{n-k} k}_{\substack{\text{Expected value of} \\ \text{Binomial Distribution} \\ \text{with parameter } p_i \\ \mathbb{E}[\mathbb{B}(p_i, n)] = np_i}} \\
 &= \frac{1}{n} np_i
 \end{aligned}$$

$= p_i \leftarrow \text{TATAAA!!}$ Each data set D'_i is drawn from P , but not independently.

这里有一个简单直接的论证，为什么 $Q(X = x_i) = P(X = x_i)$ 。到目前为止，我们假设， D 来自 P^n ，然后 Q 又从 D 中采样得到。其实你并没有必要按这个顺序去操作它。你也可以反序的从 Q 查看采样：考虑你首先使用 Q 来预留 D 中的一个点，i.e.，序号从 $1, \dots, n$ ，其中 i 表示在 D 中采样了的第 i 个数据点。到目前为止，你仅仅有下标 i ，你仍需要用数据点 (x_i, y_i) 来填充它。为此，你从 P 中采样点 (x_i, y_i) 。显然，它与你选择那个下标并没有关系，所以我们有 $Q(X = x) = P(X = x)$ 。

13.1.3 Bagging 概括

1. 有放回的从数据集 D 中采样 m 个数据集 D_1, \dots, D_m
2. 对每一个训练集 D_j ，训练一个分类器 $h_j()$
3. 最终分类器为 $h(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^m h_j(\mathbf{x})$ 。

实际上， m 越大，集成效果就越好，但在某个时刻，你会获得递减的回报。请注意，设置 m 为不必要的高值，只会减慢分类器的速度，但不会增加分类器的错误。

13.1.4 Bagging 的优点

- 容易实现。
- 降低了方差，对高方差分类器有很强的效益。
- 由于预测的结果是许多分类器的平均值，因此可以获得平均得分和方差。后者可以解释为预测的不确定性。尤其是在回归任务中，这种不确定性在其他方面很难获得。例如，假设预测房价为 \$300,000。如果一个买家想决定出价多少，那么知道这个预测是否有标准差 $+\$10,000$ 或 $+\$50,000$ 是非常有价值的。

- Bagging 提供了测试误差的无偏估计，我们称之为 out-of-bag error。其思想是，数据集 D_k 的所有点中，有一个点每次训练都没有被选择。如果我们将所有这些数据集的分类器 h_k 平均化，我们就得到了一个分类器（具有稍小的 m ），它没有 (\mathbf{x}_i, y_i) 上训练过，因此它等价于一个测试样本。如果我们计算所有这些分类器的误差，我们就得到了真实测试误差的估计。好处是我们可以不减少训练集的情况下做到这一点。我们只是按预期运行装袋，就可以免费获得这种所谓的 out-of-bag error。

更正式的说，对于每个训练点 $(\mathbf{x}_i, y_i) \in D$ ，令 $S_i = \{k | (\mathbf{x}_i, y_i) \notin D_k\}$ - 换句话说 S_i 是所有不包含 (\mathbf{x}_i, y_i) 的训练集 D_k 的集合。记分类器在这些数据集的平均结果为

$$\tilde{h}_i(\mathbf{x}) = \frac{1}{|S_i|} \sum_{k \in S_i} h_k(\mathbf{x}).$$

Bagging 误差变成了所有分类器产生的平均误差/损失

$$\epsilon_{\text{OOB}} = \frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in D} l(\tilde{h}_i(\mathbf{x}_i), y_i).$$

事实上，这是对测试误差的估计，因为对于每个训练点，我们使用了在训练期间从未见过该训练点的分类器子集。如果 m 足够大，我们取出一些分类器的事实没有显着影响，估计非常可靠。

13.2 随机森林

随机森林是最著名和最有用的装袋算法之一！随机森林本质上是一个 Bagging 决策树，它略微修改了分裂的标准。算法工作步骤如下：

1. 有放回的从数据集 D 中采样 m 个数据集， D_1, \dots, D_m 。
2. 在每一个训练集 D_j 上，训练一个决策树 $h_j()$ (max-depth= ∞)，其做了一个小改动：先随机的，无放回的选择 $k \leq d$ 个特征，然后再根据 $k \leq d$ 个特征，考虑如何进行分割（这会进一步增加树的方差）。
3. 最终的分类器为 $h(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^m h_j(\mathbf{x})$ 。

随机森林是最好的，最流行，最简单的开源分类器之一

它有两个主要的原因：

- 随机森林只有两个超参数， m 和 k 。它对二者都非常的敏感， k 的一个好选择是 $k = \sqrt{d}$ （其中 d 表示特征的总数）。你可以在计算力的范围类，将 m 设置的尽量大

- 决策树不需要大量预处理。例如，特征可以具有不同的比例，幅度或斜率。这在具有异构数据的情况下非常有利，例如医疗设置，其中特征可以是诸如血压，年龄，性别，.....，每个都以完全不同的单位记录。

随机森林有用的变种：

- 将每个训练集拆分为两个分区 $D_l = D_l^A \cup D_l^B$ ，其中 $D_l^A \cap D_l^B = \emptyset$ 。在 D_l^A 上构建树并估计 D_l^B 上的叶标签。如果一个叶子在 D_l^B 中只有一个点，你必须停止分裂。这样做的好处是每棵树和 RF 分类器都变得一致。
- 不要将每棵树长到最深处，而是根据遗漏样本进行修剪。这可以进一步改善你的偏差/方差权衡。

14

Boosting

14.1 Boosting 减小偏差

给定一个由若干个弱学习器 (weak learner) 组成的假设类 \mathbb{H} , 我们如何构建一个由弱分类器集合 (H) 构成的低偏置 (low bias) 的强学习器 (strong learner)?

首先进行 T 轮迭代创建一个集成分类器 $H_T(\vec{x}) = \sum_{t=1}^T \alpha_t h_t(\vec{x})$, 在第 t 次迭代, 我们在集成分类器添加一个弱分类器 $\alpha_t h_t(\vec{x})$ 。使用一个凸的并且可导的函数 ℓ 作为损失函数, 那么集成分类器的损失函数可以写成:

$$\ell(H) = \frac{1}{n} \sum_{i=1}^n \ell(H(\mathbf{x}_i), y_i). \quad (14.1)$$

假设我们已经完成了前 t 次迭代, 并且此时已经有了由前 t 个学习器构成的集成分类器 $H_t(\vec{x})$ 。在 $t+1$ 次迭代时我们想要集成一个新的弱学习器 h_{t+1} , 此时我们需要找一个可以最小化损失函数 ℓ 弱学习器, 即:

$$h_{t+1} = \operatorname{argmin}_{h \in \mathbb{H}} \ell(H_t + \alpha h). \quad (14.2)$$

找到 h_{t+1} 后, 我们就可以将其添加到我们的集成分类器 $H_{t+1} := H_t + \alpha h_{t+1}$ 。

那么我们如何去寻找这个 $h \in \mathbb{H}$?


这时我们就需要使用函数空间 (function space) 的梯度下降算法, 在函数空间中, 内积可以定义为 $\langle h, g \rangle = \int h(x)g(x)dx$, 这里由于我们使用离散的训练集, 我们将其重定义为 $\langle h, g \rangle = \sum_{i=1}^n h(\mathbf{x}_i)g(\mathbf{x}_i)$ 。

14.2 函数空间中的梯度下降

对于给定的集成分类器 H ，我们想要找到一个合适的步长 α 和弱学习器 h 来最小化损失函数 $\ell(H + \alpha h)$ 。我们该怎么做？

在 $\ell(H + \alpha h)$ 上使用泰勒估计!!! 我们将上式在 H 处展开：

$$\ell(H + \alpha h) \approx \ell(H) + \alpha \langle \nabla \ell(H), h \rangle. \quad (14.3)$$

 注 14.1. 公式 14.3: $\ell(H_{T+1}) \approx \ell(H_T) + \alpha \langle \nabla \ell(H_T), h \rangle$.

泰勒估计 14.3 只有在 $\ell(H)$ 附近一个小区域内成立，只要我们保证 α 很小。因此我们将 α 固定为一个很小的常量 (比如 $\alpha \approx 0.1$)，当步长 α 固定后，我们就可以利用公式 14.3 来寻找最优的弱学习器 h ：

$$\operatorname{argmin}_{h \in H} \ell(H + \alpha h) \approx \operatorname{argmin}_{h \in H} \langle \nabla \ell(H), h \rangle = \operatorname{argmin}_{h \in H} \sum_{i=1}^n \frac{\partial \ell}{\partial [H(\mathbf{x}_i)]} h(\mathbf{x}_i) \quad (14.4)$$


参考公式 14.1，我们将损失函数写为 $\ell(H) = \sum_{i=1}^n \ell(H(\mathbf{x}_i)) = \ell(H(x_1), \dots, H(x_n))$ (每一个预测都是损失函数的输入)，则 $\frac{\partial \ell}{\partial H}(\mathbf{x}_i) = \frac{\partial \ell}{\partial [H(\mathbf{x}_i)]}$ 。

到此为止，如果我们能够有一个算法 \mathbb{A} 能够通过下式求解出 h_{t+1} ：

$$h_{t+1} = \operatorname{argmin}_{h \in H} \sum_{i=1}^n \underbrace{\frac{\partial \ell}{\partial [H(\mathbf{x}_i)]}}_{r_i} h(x) \quad (14.5)$$

我们就可以进行 boosting。

所以我们需要一个函数 $\mathbb{A}(\{(\mathbf{x}_1, r_1), \dots, (\mathbf{x}_n, r_n)\}) = \operatorname{argmin}_{h \in H} \sum_{i=1}^n r_i h(\mathbf{x}_i)$ 。只需要满足 $\sum_{i=1}^n r_i h(\mathbf{x}_i) < 0$ ，loss 就会不断下降。

 注 14.2. $\sum_{i=1}^n r_i h(\mathbf{x}_i) < 0$ 意味着 loss 会下降，推导如下：

$$\sum_{i=1}^n r_i h(\mathbf{x}_i) < 0 \Rightarrow \ell(H_T) < \ell(H_T) + \alpha \langle \nabla \ell(H_T), h \rangle \Rightarrow \ell(H_T) < \ell(H_{T+1})$$

14.3 泛化的 Boosting 算法 (Anyboost)

AnyBoost 的算法流程：

```

Input:  $\ell, \alpha, \{(\mathbf{x}_i, y_i)\}, \mathbb{A}$ 
 $H_0 = 0$ 
for  $t=0:T-1$  do
     $\forall I : r_i = \frac{\partial \ell((H_t(\mathbf{x}_1), y_1), \dots, (H_t(\mathbf{x}_n), y_n))}{\partial H(\mathbf{x}_i)}$ 
     $h_{t+1} = \mathbb{A}(\{(\mathbf{x}_1, r_1), \dots, (\mathbf{x}_n, r_n)\}) = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n r_i h(\mathbf{x}_i)$ 
    if  $\sum_{i=1}^n r_i h_{t+1}(\mathbf{x}_i) < 0$  then
         $H_{t+1} = H_t + \alpha_{t+1} h_{t+1}$ 
    else
        return  $(H_t)$ 
    end
end
return  $H_T$ 

```

14.4 案例 1: 梯度增强回归树 (GBRT)

已知条件:

- ▶ 分类任务 ($y_i \in \{+1, -1\}$) 或者回归任务 ($y_i \in \mathcal{R}^k$)
- ▶ 执行回归任务的弱学习器 ($h \in \mathbb{H}, h(\mathbf{x}) \in \mathcal{R}, \forall \mathbf{x}$), 如深度固定 (例如 depth=4) 的回归树。
- ▶ 步长 α 被设置为微小的常量 (超参)
- ▶ 损失函数: 任何一个可导的凸的损失函数 $\mathcal{L}(H) = \sum_{i=1}^n \ell(H(\mathbf{x}_i))$

为了能够使用回归树来构造梯度 boosting, 我们必须找到一个可以最小化 $h = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n r_i h(\mathbf{x}_i)$ 的回归树 $h()$, 其中 $r_i = \frac{\partial \ell}{\partial H(\mathbf{x}_i)}$ 。

我们做出两个假设:

1. 首先, 我们假设 $\sum_{i=1}^n h^2(\mathbf{x}_i) = \text{常量}$ 。将 $\sum_{i=1}^n h^2(\mathbf{x}_i)$ 设置为常量之后, 向量 h 就会分布到一个圆周上, 此时我们就可以只关注它的方向。
2. 假设 CART 树满足条件: $\forall h \in \mathbb{H} \Rightarrow \exists -h \in \mathbb{H}$ (这个假设通常是真的)。这样我们可以定义负梯度 $t_i = -r_i$ 。



注 14.3. 通过对弱学习器 h 预测结果进行规范化 (normalize), 我们可以确保 $\sum_{i=1}^n h^2(\mathbf{x}_i) = \text{常量}$

根据以上假设, 我们有以下转化:

$$\begin{aligned}
 &\Rightarrow \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n r_i h(\mathbf{x}_i) \\
 &\Rightarrow = \operatorname{argmin}_{h \in \mathbb{H}} -2 \sum_{i=1}^n t_i h(\mathbf{x}_i) \\
 &\Rightarrow = \operatorname{argmin}_{h \in \mathbb{H}} \underbrace{\sum_{i=1}^n t_i^2}_{\text{constant}} - 2 \sum_{i=1}^n t_i h(\mathbf{x}_i) + \underbrace{\sum_{i=1}^n h^2(\mathbf{x}_i)}_{\text{constant}} \\
 &\Rightarrow = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n (h(\mathbf{x}_i) - t_i)^2
 \end{aligned}$$

因此, 我们只需要将 t_i 作为 x_i 的标签。所以在每个迭代过程, 我们就可以将 t_1, \dots, t_n 作为数据的真实标签来学习。

如果我们的损失函数是平方损失 (squared loss), 如 $\ell(H) = \frac{1}{2} \sum_{i=1}^n (H(\mathbf{x}_i) - y_i)^2$, 那么我们就可以得到:

$$t_i = -\frac{\partial \ell}{\partial H(\mathbf{x}_i)} = y_i - H(\mathbf{x}_i),$$

此时 t_i 就是残差, \mathbf{r} 表示一个由 \mathbf{y} 指向 \mathbf{H} 的向量。

GBRT 的算法流程:

```

Input:  $\ell, \alpha, \{(\mathbf{x}_i, y_i)\}, \mathbb{A}$ 
 $H = 0$ 
for  $t=1:T$  do
     $\forall i: t_i = y_i - H(\mathbf{x}_i)$ 
     $h = \operatorname{argmin}_{h \in \mathbb{H}} (h(\mathbf{x}_i) - t_i)^2$ 
     $H \leftarrow H + \alpha h$ 
end
return  $H$ 

```

14.5 案例 2: 自适应提升 (AdaBoost)

已知条件:

- ▶ 设置: 分类任务 ($y_i \in \{+1, -1\}$)
- ▶ 弱学习器: $h \in \mathbb{H}, h(\mathbf{x}_i) \in \{-1, +1\}, \forall x$
- ▶ 步长: 我们通过**线性搜索** (line-search) 来寻找最合适的步长 α
- ▶ 损失函数: 指数形式的损失函数 $\ell(H) = \sum_{i=1}^n e^{-y_i H(\mathbf{x}_i)}$

14.5.1 寻找最好的弱学习器

首先我们计算梯度 $r_i = \frac{\partial \ell}{\partial H(\mathbf{x}_i)} = -y_i e^{-y_i H(\mathbf{x}_i)}$ 。为了表示方便, 我们也定义每个数据的权重 $w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)}$, 其中 $Z = \sum_{i=1}^n e^{-y_i H(\mathbf{x}_i)}$, 注意, 此时的规范化因子 Z 和我们的损失函数相同, 因此我们就可以把 w_i 视为数据 (\mathbf{x}_i, y_i) 对总体损失值的贡献, 贡献越大权重越大。

为了寻找下一个最优的学习器, 我们需要解决以下优化问题: (假设我们的任务为

二分类)

$$h(\mathbf{x}_i) = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n r_i h(\mathbf{x}_i) \quad \left(\text{替换: } r_i = e^{-H(\mathbf{x}_i)y_i} \right) \quad (14.6)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} - \sum_{i=1}^n y_i e^{-H(\mathbf{x}_i)y_i} h(\mathbf{x}_i) \quad \left(\text{替换: } w_i = \frac{1}{Z} e^{-H(\mathbf{x}_i)y_i} \right) \quad (14.7)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} - \sum_{i=1}^n w_i y_i h(\mathbf{x}_i) \quad \left(h(\mathbf{x}_i)y_i = 1 \iff h(\mathbf{x}_i) = y_i \right) \quad (14.8)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i - \sum_{i:h(\mathbf{x}_i) = y_i} w_i \quad \left(\sum_{i:h(\mathbf{x}_i) = y_i} w_i = 1 - \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i \right) \quad (14.9)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i \quad \left(\text{分类错误的加权和} \right) \quad (14.10)$$

我们将分类错误样本的损失值的加权和表示为 $\epsilon = \sum_{i:h(\mathbf{x}_i)y_i=-1} w_i$ ，可以发现，AdaBoost 只需要将训练数据和训练集上的分布 (如所有样本规范化后的权重 w_i) 作为输入就可以返回一个可以降低分类错误加权和的分类器 $h \in H$ ，而且为了使 $h(\mathbf{x}) = \sum_i r_i h(\mathbf{x}_i)$ 为负数，我们只需要保证 $\epsilon < 0.5$ 。

14.5.2 寻找步长 α

在 GBRT 中，我们需要设置步长 α 为一个小的常量，但是对于 AdaBoost，我们可以自适应找到最优的步长 (可以尽可能最大程度地最小化损失函数 ℓ)。

给定 ℓ, H, h ，我们需要解决如下优化问题：

$$\alpha = \operatorname{argmin}_{\alpha} \ell(H + \alpha h) \quad (14.11)$$

$$= \operatorname{argmin}_{\alpha} \sum_{i=1}^n e^{-y_i [H(\mathbf{x}_i) + \alpha h(\mathbf{x}_i)]} \quad (14.12)$$

对 α 求导并令其为 0，我们有如下推导过程：

$$\sum_{i=1}^n y_i h(\mathbf{x}_i) e^{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))} = 0 \quad (y_i h(\mathbf{x}_i) \in \{+1, -1\}) \quad (14.13)$$

$$- \sum_{i:h(\mathbf{x}_i)y_i=1} e^{\underbrace{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))}_1} + \sum_{i:h(\mathbf{x}_i)y_i=-1} e^{\underbrace{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))}_{-1}} = 0 \quad \left(w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)} \right) \quad (14.14)$$

$$- \sum_{i:h(\mathbf{x}_i)y_i=1} w_i e^{-\alpha} + \sum_{i:h(\mathbf{x}_i)y_i=-1} w_i e^{\alpha} = 0 \quad \left(\epsilon = \sum_{i:h(\mathbf{x}_i)y_i=-1} w_i \right) \quad (14.15)$$

$$-(1 - \epsilon) e^{-\alpha} + \epsilon e^{\alpha} = 0 \quad (14.16)$$

$$e^{2\alpha} = \frac{1 - \epsilon}{\epsilon} \quad (14.17)$$

$$\alpha = \frac{1}{2} \ln \frac{1 - \epsilon}{\epsilon} \quad (14.18)$$

我们发现最优的步长大小是可以求解 (存在解析解) 的，因此 AdaBoost 收敛速度很快！

14.5.3 Re-normalization

当我们加入一个新的学习器之后，如 $H_{t+1} = H_t + \alpha h$ ，需要重新计算所有的权重并且重新规范化。权重 \hat{w}_i 更新很直观表示如下：

$$\hat{w}_i \leftarrow \hat{w}_i * e^{-\alpha h(\mathbf{x}_i) y_i}$$

并且规范化因子 Z 变为：


$$Z \leftarrow Z * 2\sqrt{\epsilon(1 - \epsilon)}.$$

综合前面两个，我们可以得到总的更新公式：

$$w_i \leftarrow w_i \frac{e^{-\alpha h(\mathbf{x}_i) y_i}}{2\sqrt{\epsilon(1 - \epsilon)}}.$$



注 14.4. 只要 H 满足条件 $\forall h \in H, -h \in H$ ，就不会有 $\epsilon > 0.5$ 。原因：如果 h 有 $error \epsilon$ ，必然会存在 $-h$ 有 $error 1 - \epsilon$ ，因此我们只需要简单的交换 h 和 $-h$ 就可以获得一个更小的 $error$ 。

 **注 14.5.** 当 $\epsilon = 0.5$ 时，内部的循环会终止，大多数情况下，它会趋于 0.5，此时我们的最后一个加入的分类器 h 的效果和投硬币的效果是一样的，不会对集成分类器有任何的帮助。当 $\epsilon = 0.5$ 时，步长 $\alpha = 0$ 。

AdaBoost 的算法流程：

```

Input:  $\ell, \alpha, \{(\mathbf{x}_i, y_i)\}, \mathbb{A}$ 
 $H_0 = 0$ 
 $\forall i: w_i = \frac{1}{n}$ 
for  $t=0:T-1$  do
     $h = \operatorname{argmin}_h \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i$     [ $h = \mathbb{A}((w_1, \mathbf{x}_1, y_1), \dots, (w_n, \mathbf{x}_n, y_n))$ ]
     $\epsilon = \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i$ 
    if  $\epsilon < \frac{1}{2}$  then
         $\alpha = \frac{1}{2} \ln(\frac{1-\epsilon}{\epsilon})$ 
         $H_{t+1} = H_t + \alpha h$ 
         $\forall i: w_i \leftarrow \frac{w_i e^{-\alpha h(\mathbf{x}_i) y_i}}{2\sqrt{\epsilon(1-\epsilon)}}$ 
    else
        return ( $H_t$ )
    end
end
return ( $H_T$ )

```

14.5.4 进一步分析

让我们对参数的更新过程进一步分析：

权重更新

$$\hat{w}_i \leftarrow \hat{w}_i * e^{-\alpha h(\mathbf{x}_i) y_i}$$

分类正确的时候， $h(\mathbf{x}_i) y_i = +1$ ，此时权重的系数 $e^{-\alpha} < 1$ ，因此权重会变小，而当分类错误的时候恰恰相反，权重会变大。

规范化因子更新

$$Z \leftarrow Z * 2\sqrt{\epsilon(1-\epsilon)}.$$

T 次迭代后，我们可以用 Z 来求得损失函数的界限：

$$\ell(H) = Z = n \prod_{t=1}^T 2\sqrt{\epsilon_t(1-\epsilon_t)},$$

其中， n 表示 $T=0$ 时， $Z_0 = n$ ，此时所有的权重都为 $\frac{1}{n}$ 。如果我们定义 $c = \max_t \epsilon_t$ ，则有下式成立：

$$\ell(H) \leq n \left[2\sqrt{c(1-c)} \right]^T.$$

函数 $c(1-c)$ 在 $c = \frac{1}{2}$ 时取得最大值，但是我们知道 $\epsilon_t < \frac{1}{2}$ ，因此最大值取不到，只能有 $c(1-c) < \frac{1}{4}$ ，我们可以将其写成： $c(1-c) = \frac{1}{4} - \gamma^2$ ，其中 γ 表示某一个值。这样我们就有：

$$\ell(H) \leq n(1 - 4\gamma^2)^{\frac{T}{2}}.$$

也就是说，训练时的 loss 会指数级下降！

我们甚至可以计算出多少轮迭代之后训练误差为 0，因为我们知道，训练的 loss 可以作为训练误差 (定义为： $\sum_{i=1}^n \delta_{H(\mathbf{x}_i) \neq y_i}$) 的上界 (在所有情况下有： $\delta_{H(\mathbf{x}_i) \neq y_i} < e^{-y_i H(\mathbf{x}_i)}$)，因此我们可以计算多少步之后 loss 值会小于 1，也就表示不会有一个样本被分类错误。

$$n(1 - 4\gamma^2)^{\frac{T}{2}} < 1 \Rightarrow T > \frac{2 \log(n)}{\log(\frac{1}{1-4\gamma^2})}.$$

这也表示在 $O(\log(n))$ 次迭代后，训练误差会变成 0！

14.6 总结

Boosting 是一个将弱分类器转化成强分类器的一个很好的方法，它定义了一个算法族，如梯度 Boosting, AdaBoost, LogitBoost 和很多其他的，其中梯度增强回归树 (GBRT) 是排序问题最流行的方法之一！

参考文献

- [1] Tom M. Mitchell. *Machine learning, International Edition*. McGraw-Hill Series in Computer Science. McGraw-Hill, 1997.
- [2] Arthur L. Samuel. “Some Studies in Machine Learning Using the Game of Checkers”. 刊于: *IBM Journal of Research and Development* 3.3 (1959), pp. 210–229.
- [3] Gerald Tesauro. “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play”. 刊于: *Neural Comput.* 6.2 (1994), pp. 215–219.