

2020-04-6

机器学习内部讲义

何琨

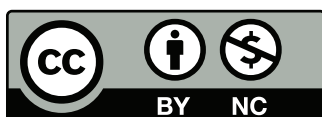
华中科技大学计算机科学与技术学院

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

Version 1.00

编译日期: 2020-04-6

对本讲义的订正或改进建议请发何琨老师邮箱:
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。感谢实验室 18、19 和部分 20 级同学们的整理。此 1.0 版本将用于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

2020.4.5

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	3
1.3.1 监督学习算法	4
1.3.2 无监督学习算法	4
1.3.3 强化学习算法	4
第二章 监督学习	5
2.1 介绍	5
2.2 设置	5
2.2.1 标签空间实例	6
2.2.2 特征向量实例	6
2.3 损失函数	7
2.3.1 例子	7
2.4 泛化	8
2.5 训练集/测试集划分	9
2.5.1 如何划分数据	9
2.6 总结	9
第三章 k 近邻	11
3.1 k 近邻算法	11
3.1.1 基本假设	11
3.1.2 分类规则	11

3.1.3	k 近邻的正式定义	11
3.1.4	k 值的选取	12
3.2	我们使用什么样的距离函数?	12
3.3	贝叶斯最优分类器	12
3.4	1-NN 收敛性证明	13
3.5	维数灾难	13
3.5.1	点和点之间的距离	13
3.5.2	点和超平面之间的距离	14
3.6	低维结构的数据	14
3.7	k-平均算法	15
第四章	感知机	17
4.1	感知机分类模型	17
4.2	感知机算法	18
4.3	感知机收敛性	19
第五章	贝叶斯与概率估计	21
5.1	联合概率分布	21
5.2	MLE	22
5.3	MAP	23
第六章	朴素贝叶斯法	25
6.1	贝叶斯分类器	25
6.2	朴素贝叶斯法	26
6.3	$P([\mathbf{x}]_{\alpha} y)$ 参数估计	28
6.3.1	实例 1: 离散特征	28
6.3.2	实例 2: 多项式特征	30
6.3.3	实例 3: 连续特征 (高斯朴素贝叶斯法)	31
6.4	朴素贝叶斯法是一种线性分类器	31
6.5	思考题	33
第七章	逻辑回归	35
7.1	逻辑回归	35
7.2	最大似然估计 (MLE)	35
7.3	最大后验估计 (MAP)	36

7.4	总结	36
第八章	梯度下降法	39
8.1	泰勒展开	39
8.2	梯度下降法：使用一阶近似	40
8.3	自适应梯度下降法 Adagrad	40
8.4	牛顿法：使用二阶近似	41
8.5	最佳实践	42
第九章	线性回归	45
9.1	假设	45
9.2	最大似然估计	46
9.3	最大后验估计	47
9.4	总结	48
第十章	SVM	49
10.1	线性分类问题	49
10.2	SVM——线性可分情形	49
10.2.1	主优化问题	50
10.2.2	对偶优化问题	50
10.2.3	支持向量	51
10.3	SVM——线性不可分情形	51
10.3.1	对偶优化问题	52
10.3.2	KKT 条件	53
10.4	核函数	53
10.4.1	多项式核函数	53
10.4.2	高斯核函数	54
10.4.3	sigmoid 核函数	54
第十一章	经验风险最小化	55
11.1	经验风险	55
11.1.1	损失函数	55
11.1.2	风险函数	56
11.1.3	经验风险	56
11.2	经验风险最小化与结构风险最小化	56

11.2.1 经验风险最小化	56
11.2.2 结构风险最小化	57
第十二章 模型调试	59
12.1 过拟合和欠拟合	59
12.2 确定最佳位置	59
12.2.1 K-Fold 交叉验证	60
12.2.2 迭代搜索	60
12.3 早停	60
第十三章 方差偏差权衡	63
13.1 概念定义	63
13.1.1 偏差	63
13.1.2 方差	63
13.2 图形表示	63
13.2.1 偏差方差可视化	63
13.2.2 思考	65
13.2.3 解答	65
13.3 数学表示	65
13.4 偏差方差困境	65
13.5 偏差、方差与过拟合、欠拟合的关系	66
13.6 偏差、方差与模型复杂度的关系	67
第十四章 核方法	69
14.1 Handcrafted Feature Expansion	69
14.2 The Kernel Trick	70
14.2.1 Gradient Descent with Squared Loss	70
14.3 Inner-Product Computation	71
14.4 General Kernels	72
第十五章 核方法 (续)	73
15.1 Well-defined kernels	73
15.2 Kernel Machines	74
15.2.1 核线性回归	74
15.2.2 Nearest Neighbors	76

15.3	Kernel SVM	76
15.3.1	Kernel SVM - the smart nearest neighbor	77
第十六章	高斯过程	79
16.1	一旦高斯，一直高斯	79
16.1.1	1. 正规化	79
16.1.2	2. 边界化	80
16.1.3	3. 和性质	80
16.1.4	4. 条件分布	80
16.2	高斯过程回归	80
16.2.1	后验预测分布	80
16.3	高斯过程——定义	81
16.4	高斯过程回归 (GPR)	81
16.4.1	加性高斯噪声	82
16.5	实践运用	82
16.5.1	边际似然和超参数学习	82
16.5.2	协方差函数- GP 模型的核心	82
16.6	总结	83
16.7	贝叶斯全局优化	83
第十七章	KD Trees	85
17.1	KD 树	85
17.1.1	创建 KD 树	86
17.1.2	近邻搜索	86
17.2	球树	87
17.2.1	创建球树	88
17.2.2	球树的应用场景	88
17.3	总结	88
第十八章	决策树	91
18.1	基本概念	92
18.1.1	信息熵	92
18.1.2	基尼指数	93
18.2	ID3	93

18.3 CART	94
第十九章 Bagging	95
19.1 Bagging	95
19.1.1 减小方差	95
19.1.2 方法: Bagging(集成学习)	96
19.1.3 Bagging 概括	97
19.1.4 Bagging 的优点	97
19.2 随机森林	98
第二十章 Boosting	101
20.1 Boosting 减小偏差	101
20.2 函数空间中的梯度下降	102
20.3 泛化的 Boosting 算法 (Anyboost)	102
20.4 案例 1: GBRT	103
20.5 案例 1: AdaBoost	104
20.5.1 寻找最好的弱学习器	104
20.5.2 寻找步长	105
20.5.3 Re-normalization	106
20.5.4 进一步分析	107
20.6 总结	108

1

引言

机器学习和人工智能对不同的人来说意味着不同的东西，但最新的方法有一个共同点：它们基于这样的想法：程序的输出应该主要是从高维度创建的，可能是巨大的数据集，只需少或无需人工干预或指导。现存在开源工具用于各种机器学习和人工智能项目。在本文中，我们将概述当今的机器学习。

1.1 什么是机器学习

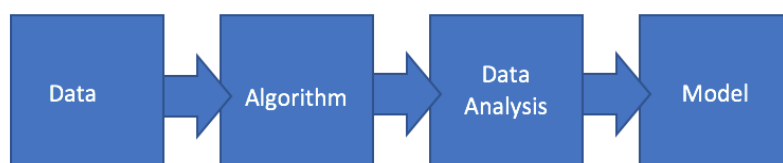
我们已经将机器学习看作是过去几年的流行语，其原因可能是应用程序的大量数据生成，过去几年计算能力的提高以及更好的算法的开发。

每个行业的行业都试图从机器学习中受益。您可能已经在使用使用它的设备。例如，像 Fitbit 这样的可穿戴式健身追踪器，或者像 GoogleHome 这样的智能家居助手。但是有更多的 ML 使用例子：

- ▶ 预测：机器学习也可用于预测系统。考虑贷款示例，为了计算故障概率，系统需要对可用数据进行分组。
- ▶ 图像识别：机器学习也可用于图像中的面部检测。在几个人的数据库中，每个人都有一个单独的类别。
- ▶ 语音识别：将口语单词翻译成文本。它用于语音搜索等。语音用户界面包括语音拨号，呼叫路由和设备控制。它还可以用于简单的数据输入和结构化文档的准备。
- ▶ 医学诊断：识别癌组织。金融业和交易：公司在欺诈调查和信用检查中使用 ML。

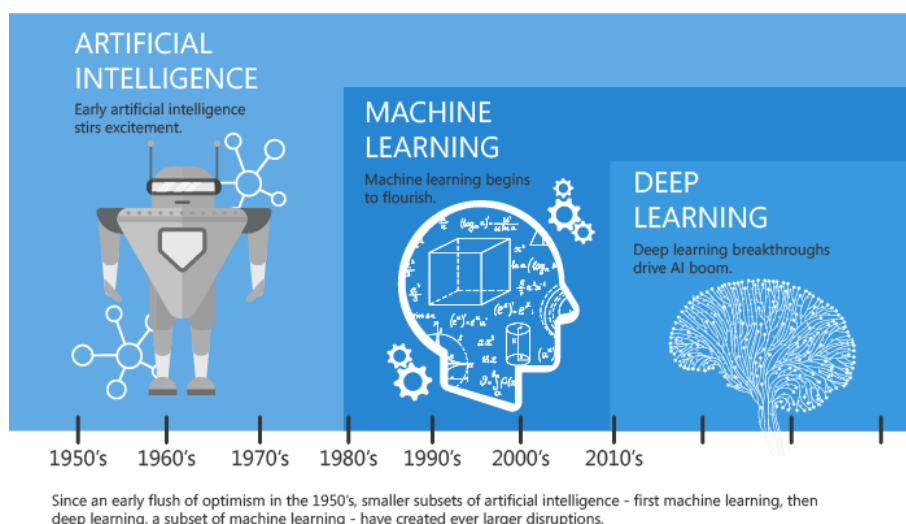
简单来说，机器学习是指系统如何从示例数据中学习解决问题而不是编写特定逻辑。根据 Arthur Samuel 的说法，机器学习算法使计算机能够从数据中学习，甚至可以自我改进，而无需明确编程。机器学习（ML）是一种算法类别，它允许软件应用程序在未经明确编程的情况下更准确地预测结果。机器学习的基本前提是构建可以接收输入数据的算法，并使用统计分析来预测输出，同时在新数据可用时更新输出。

在传统方法的情况下，我们仔细分析问题并编写代码，代码读取数据并使用其控制逻辑来确定要执行的正确部分，然后生成正确的结果。这些逻辑代码语句中的每一个都有必须定义的测试，但是我们在机器学习中定义这些测试的数据不断变化非常困难。



在机器学习的情况下，我们不会编写产生结果的逻辑。我们收集所需的数据，修改机器学习可以使用的形式，然后将此数据传递给算法，使用算法然后分析数据，最后创建一个模型，使得该模型根据数据实现解决方案。

1.2 机器学习的历史



在 20 世纪 40 年代，第一台手动操作的计算机系统 ENIAC（电子数字积分器和计算机）被发明出来。那时，“计算机”这个词被用作具有强大数值计算能力的人的名字，

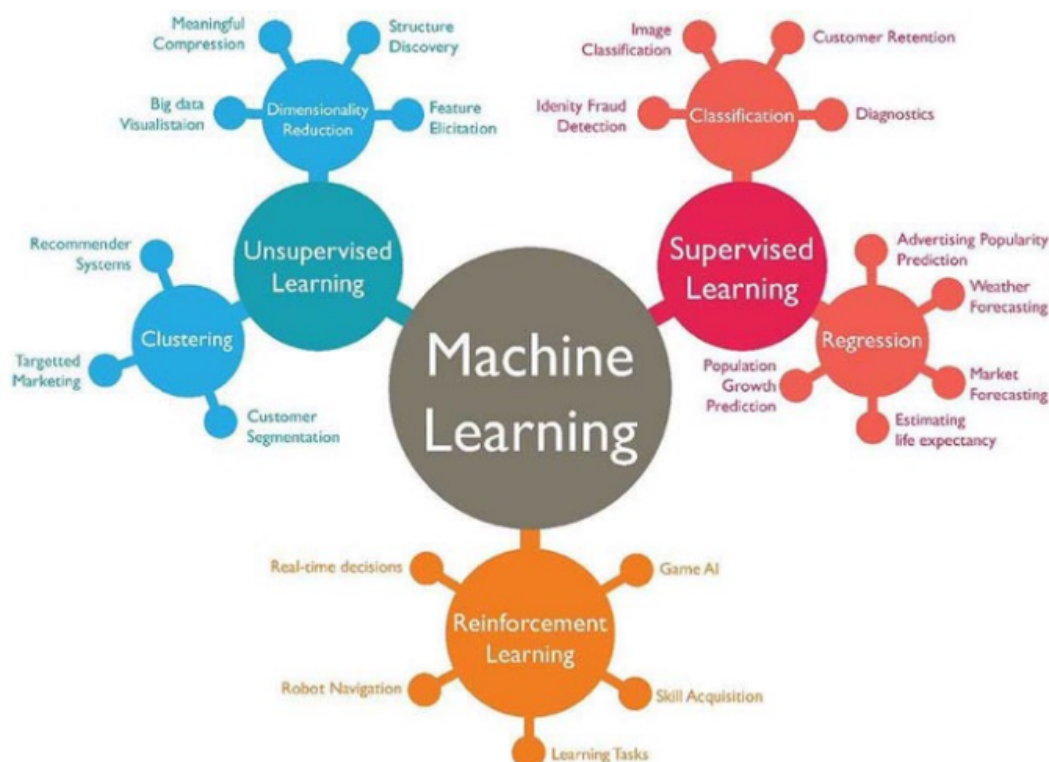
因此，ENIAC 被称为数值计算机！好吧，你可能会说它与学习无关?! 错误，从一开始就想要建立一个能够模仿人类思维和学习的机器。

随后，多亏了统计数据，机器学习在 20 世纪 90 年代变得非常有名。计算机科学与统计学的交叉在 AI 中产生了概率方法。这使得该领域进一步转向数据驱动的方法。在获得大规模数据后，科学家们开始构建能够分析和学习大量数据的智能系统。作为一个亮点，IBM 的 Deep Blue 系统击败了世界国际象棋冠军，即大师 Garry Kasparov。是的，我知道卡斯帕罗夫指责 IBM 作弊，但现在这是一段历史，深蓝在博物馆里安静地休息。

1.3 机器学习算法的类型

机器学习可分为 3 种学习算法。

- ▶ 监督学习
- ▶ 无监督学习
- ▶ 强化学习



1.3.1 监督学习算法

在监督学习中，AI 系统具有标记的数据，这意味着每个数据都标记有正确的标签。目标是很好地近似映射函数，当您有新的输入数据 (x) 时，您可以预测该数据的输出变量 (Y)。监督学习算法的类型包括：

- ▶ 分类：分类问题是当输出变量是类别时，例如“红色”或“蓝色”或“疾病”和“无疾病”。
- ▶ 回归：回归问题是输出变量是实际值，例如“美元”或“重量”。

1.3.2 无监督学习算法

在无监督学习中，AI 系统具有未标记的未分类数据，系统的算法在没有事先培训的情况下对数据起作用。输出取决于编码算法。使系统受到无监督学习是测试 AI 的一种方法。无监督学习算法的类型包括：

- ▶ 聚类：聚类问题是您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。
- ▶ 关联：关联规则学习问题是您想要发现描述大部分数据的规则，例如购买 X 的人也倾向于购买 Y。

1.3.3 强化学习算法

强化学习算法或代理通过与其环境交互来学习。代理通过正确执行获得奖励，并因错误执行而受到处罚。代理人通过最大化其奖励并最小化其惩罚而在没有人的干预的情况下学习。它是一种动态编程，使用奖励和惩罚系统训练算法。它基本上利用获得的奖励，代理改善其环境知识以选择下一个动作。

2

监督学习

2.1 介绍

监督学习的目标是根据已有的数据作出预测，例如，监督学习的一个流行应用是电子邮件垃圾邮件过滤。在这里，电子邮件需要被分类为垃圾邮件或非垃圾邮件。按照传统计算机科学的方法，人们可能会想要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良好，但它有明显的缺点。随着垃圾邮件的变化，它必须被重写，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消息。即使它成功了，它也可能不容易应用于不同的语言。机器学习使用不同的方法生成可以根据数据进行预测的程序，不是手工编程，而是从过去的数据中学习。如果我们有数据实例，我们确切地知道正确的预测是什么，这个过程就有效。例如，过去的的数据可能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，以预测每个带注释的数据实例的正确标签。

2.2 设置

我们的训练数据来源于输入对 (\mathbf{x}, y) ，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) 来源于一些分布 $P(X, Y)$, 我们想要学习函数 h , 对于新的点 $(\mathbf{x}, y) \sim P$, 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:

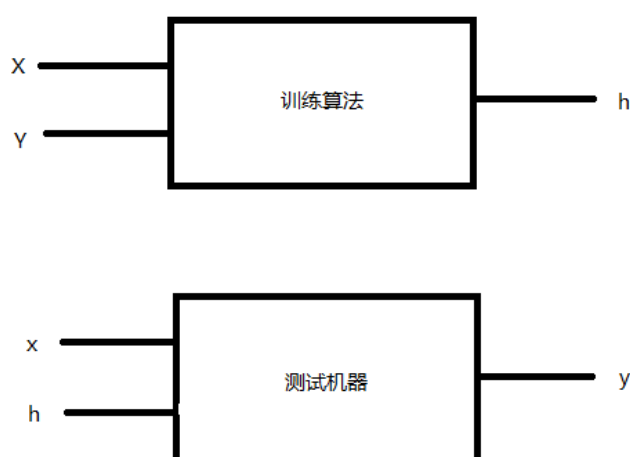


图 2-1 监督学习

Figure 2-1 supervised learning

其中函数 h 属于假想集 H , 假想集包括多种函数, 如线性分类函数、决策树、人工神经网络、SVM 等等。一个成功的机器学习实例都是基于某个假设。

下面, 对于 X 和 Y 举一些例子。

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	$c = \{0, 1\}$ or $c = \{-1, +1\}$	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	$c = \{0, 1, \dots, K\} (K \geq 2)$	如脸分类器, 一个人可以是 K 身份中的一个 (例如, “1” 表示奥巴马, “2” 表示布什)
回归	$c = R$	如预测未来的温度和人的身高

2.2.2 特征向量实例

我们称 \mathbf{x}_i 为特征向量, d 维中的每一维表示第 i 个样本的一个特征, 以下为几个例子。

病人数据 $\mathbf{x}_i = \{x_i^1, x_i^2, \dots, x_i^d\}$, 其中 $x_i^1=0$ or 1 , 可能代表第 i 个病人的性别, x_i^2 可能代表第 i 个病人的身高, x_i^3 可能代表第 i 个病人的年龄, 等等。在这种情况下, $d \leq 100$ 并且特征向量是密集的, 即, \mathbf{x}_i 中的非零坐标的数量相对于 d 是大的。

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, $d \sim 10000 - 10M$ 并且特征向量是稀疏的, 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。在这种情况下, $d \sim 10000 - 10M$ 并且特征向量是密集的。

2.3 损失函数

通常有两个步骤去学习一个假设函数 $\mathbf{h}()$ 。首先, 我们选择适合这个特殊的学习问题类型的机器学习算法。这定义了假设类 \mathcal{H} , 即我们可能学习的函数集。第二步是找到这个类中的最佳函数, $\mathbf{h} \in \mathcal{H}$ 。第二步实际上是学习的过程, 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 \mathbf{h} , 它在我们的训练数据中出错最少。(如果没有一个函数, 我们通常会通过一些简单性的概念来选择“最简单”的函数——但是我们将在后面的类中更详细地讨论这个问题。) 我们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优于另一个函数。这就是损失函数 (又称风险函数) 的作用。对于我们的训练集, 一个损失函数对假设函数, $\mathbf{h} \in \mathcal{H}$ 进行评估, 并告诉我们情况有多糟糕。损失越大, 情况就越糟——损失为零意味着它可以做出完美的预测。通常的做法是用训练样本的总数 n 对损失进行标准化, 这样输出就可以解释为每个样本的平均损失 (与 n 无关)。

2.3.1 例子

0-1 损失函数: 最简单的损失函数是 0-1 损失函数。它从逐个计算假设函数 \mathbf{h} 在训练集上犯了多少错误。待对于每一个例子, 如果预测错误, 则损失为 1, 否则损失为 0。归一化后的 0-1 损失函数返回分类错误的集合占训练集的比例, 也常称为训练误差。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 损失函数是不可微的、非连续的。形式上, 零一损失可以表述为:

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

这个损失函数返回了数据集 \mathcal{D} 的错误率。对于分类器分类错误的每一个例子，会造成 1 的损失，而分类正确的样本不会造成任何损失。

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 $(\mathbf{h}(x)_i - \mathbf{y}_i)^2$ 为损失。平方操作有两个效果；1. 损失是非负的，2. 损失以绝对错误预测量的平方增长。后一种性质不鼓励预测值距离实际值太远（否则后果将非常严重，会产生截然不同的假设函数）。另一方面，如果一个预测非常接近于正确，那么这个平方就会很小，为了获得零误差，人们很少关注这个例子。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} ，标签 \mathbf{y} 是根据分布 $\mathbf{P}(\mathbf{x}|\mathbf{y})$ 的概率。那么将平方损失最小化的最优预测函数是其期望值，即 $\mathbf{h}(x) = \mathbf{E}_{\mathbf{P}(\mathbf{x}|\mathbf{y})}[\mathbf{y}]$ 形式上平方损失为：

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

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。它受到 $|\mathbf{h}(x)_i - \mathbf{y}_i|$ 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} ，标签 \mathbf{y} 是根据分布 $\mathbf{P}(\mathbf{x}|\mathbf{y})$ 的概率。那么为了使绝对损失最小化，最优预测函数是预测其中值，即 $\mathbf{h}(x) = \text{MEDIAN}_{\mathbf{P}(\mathbf{x}|\mathbf{y})}[\mathbf{y}]$ 形式上，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，我们可以尝试找到使损失最小化的函数 \mathbf{h} ：

$$\mathbf{h} = \operatorname{argmin}_{\mathbf{h} \in \mathcal{H}} \mathcal{L}(\mathbf{h})$$

机器学习的很大一部分目光集中在这个问题上，如何有效地做到最小化。如果你发现在你的数据集 \mathcal{D} 上有一个使得损失函数较低的函数 $\mathbf{h}(\cdot)$ ，你怎么知道他是否可以在其他数据集上有同样的效果。

错误的例子“存储器” $\mathbf{h}(\cdot)$

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

对于这个 $\mathbf{h}(\cdot)$ ，我们在训练集 \mathcal{D} 上可以达到 0% 的错误率，但如果样本不在训练集 \mathcal{D} 上，那么情况就糟糕透了。即这个函数存在过拟合问题。

2.5 训练集/测试集划分

为了解决过拟合问题，我们将数据集 \mathcal{D} 划分为三个子集： \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，我们根据 \mathcal{D}_{TR} 来选择 $\mathbf{h}(\cdot)$ ，根据 \mathcal{D}_{TE} 来评估 $\mathbf{h}(\cdot)$ 。

思考?：我们为什么需要 \mathcal{D}_{VA} ?

\mathcal{D}_{VA} 是用来验证我们从 \mathcal{D}_{TR} 获得的 $\mathbf{h}(\cdot)$ 是否有过拟合的问题。 $\mathbf{h}(\cdot)$ 需要在 \mathcal{D}_{VA} 上被验证，如果损失函数较大，那么我们需要根据 \mathcal{D}_{TR} 重新修正我们的 $\mathbf{h}(\cdot)$ ，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} 上损失函数较小。在 \mathcal{D}_{TR} 和 \mathcal{D}_{VA} 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，如果你想训练一个电子邮件垃圾邮件过滤器，你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——这样你就可以从过去严格地预测未来。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。

根据时间，如果数据是根据时间来收集的。

通常，如果数据是时序的，我们必须按时间分割它。

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）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：

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

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

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

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：

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

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需要对 \mathcal{H} 进行假设。这个假设是取决于数据的。并对数据集/分布 \mathcal{P} 的假设进行编码。显然，没有一个完美的 \mathcal{H} 可以解决所有的问题。

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？没有假设是不可能知道答案的。ML 算法最常见的假设是要拟合的函数是局部光滑的。

3

k 近邻

3.1 k 近邻算法

3.1.1 基本假设

相似的输入可以得到相似的输出。

3.1.2 分类规则

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（也就是上面所说的 k 个邻居），把这 k 个实例出现最多的标签，设为 \mathbf{x} 的标签。

3.1.3 k 近邻的正式定义

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 $h()$, 它将返回 S_x 中最常见的标签:

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

其中 $\text{mode}(\cdot)$ 意思是选择出现最多的标签。

3.1.4 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, 选取所有数据点中频数最高的标签。

3.2 我们使用什么样的距离函数?

k 近邻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。

3.3 贝叶斯最优分类器

举例: 假设 (虽然在大多数情况下不是这样) 我们已知 $P(y|\mathbf{x})$, 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, 但是它也会犯错误。当样本没有可能性最大的标签时, 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:

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

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有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 kNN 分类器的错误率。

3.4 1-NN 收敛性证明

当 $n \rightarrow \infty$, 1-NN 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 $n \rightarrow \infty$, $\text{dist}(\mathbf{x}_{NN}, \mathbf{x}_t) \rightarrow 0$, 即 $\mathbf{x}_{NN} \rightarrow \mathbf{x}_t$. (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:

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

好消息是：当 $n \rightarrow \infty$, 1-NN 分类器仅仅比理论最好的分类器坏两倍。

坏消息是：我们面临维数灾难！

3.5 维数灾难

3.5.1 点和点之间的距离

kNN 分类器假定相似的点具有相似的标签。不幸的是，在高维空间，从概率分布中取出的点，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。我们在一个单位立方体中随即均匀的画点，然后我们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的空间。

记单位立方体为 $[0, 1]^d$. 所有的训练数据在这个立方体中均匀分布, 即 $\forall i, \mathbf{x}_i \in [0, 1]^d$. 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球的边长。显然， $\ell^d \approx \frac{k}{n}$, 即 $\ell \approx \left(\frac{k}{n}\right)^{1/d}$.

当 $d=2, \ell=0.1$

当 $d=10, \ell=0.2$

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。

一些人可能认为一种补救措施是增加训练集的数量，也就是 n ，直到测试点的邻居真正接近于测试点。然而，由于 $n = \frac{k}{\epsilon^d} = k \cdot 10^d$ 是指数级增长，当 $d > 100$ 时，我们需要的数据点数量就比整个宇宙的电子都要多了。

3.5.2 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 1-1。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，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。在我们接下来的课程中可以看到，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发明而广为人知，这种现象常常被错误归因为神经网络的复杂性。

图 3-1 维数灾难对点与点之间的距离和点与超平面之间的距离有不同的效果

3.6 低维结构的数据

然而，并不是所有数据都会丢失。数据可能分布在低维子空间或子流形上。比如，自然图像。在这种情况下，数据的真实维数可能比它所在环境空间的维数小很多。图 1-2 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……）来表述这张人脸。

图 3-2 数据集取自嵌入在三围空间的二维流形。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

3.7 k-平均算法

k-平均算法（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流行于数据挖掘领域。k-平均聚类的目的是：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他最近的均值（此即聚类中心）对应的聚类，以之作为聚类的标准。k-平均聚类与 k-近邻之间没有任何关系。

k 近邻是有标记的监督学习，k 平均则是无监督聚类算法。

4

感知机

感知机 (Perceptron) 假设数据是二分类的线性可分的数据, 数据标签是 +1 和 -1, 寻找一个超平面将数据分离开来。

4.1 感知机分类模型

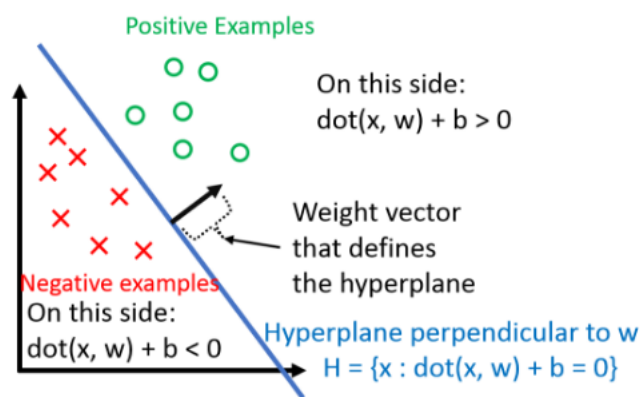


图 4-1 感知机分类示例

如图4-1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 $h(x_i) = (\vec{w} \cdot \vec{x}_i + b)$, 其中 b 是一个偏置, 如果去掉则超平面经过原点。为了方便处理, 我们可以将 b 放入到权重向量 \vec{w} 当中:

$$\vec{x}_i \text{ 改写为 } \begin{bmatrix} \vec{x}_i \\ 1 \end{bmatrix} \vec{w} \text{ 改写为 } \begin{bmatrix} \vec{w} \\ b \end{bmatrix}$$

很容易验证：

$$\begin{bmatrix} \vec{x}_i \\ 1 \end{bmatrix} \cdot \begin{bmatrix} \vec{w} \\ b \end{bmatrix} = \vec{w} \cdot \vec{x}_i + b$$

此时，感知机可以表示为：

$$h(x_i) = \text{sign}(\vec{w} \cdot \vec{x}) \quad (4.1)$$

4.2 感知机算法

感知机的模型参数是 \vec{w} （该参数定义了一个超平面），学习参数 \vec{w} 的算法如下所示。

```

Initialize  $\vec{w} = \vec{0}$                                 // Initialize  $\vec{w}$ .  $\vec{w} = \vec{0}$  misclassifies everything.
while TRUE do                                         // Keep looping
   $m = 0$                                               // Count the number of misclassifications,  $m$ 
  for  $(x_i, y_i) \in D$  do                             // Loop over each (data, label) pair in the dataset,  $D$ 
    if  $y_i(\vec{w}^T \cdot \vec{x}_i) \leq 0$  then           // If the pair  $(\vec{x}_i, y_i)$  is misclassified
       $\vec{w} \leftarrow \vec{w} + y_i \vec{x}_i$            // Update the weight vector  $\vec{w}$ 
       $m \leftarrow m + 1$                              // Counter the number of misclassification
    end if
  end for
  if  $m = 0$  then                                       // If the most recent  $\vec{w}$  gave 0 misclassifications
    break                                             // Break out of the while-loop
  end if
end while                                             // Otherwise, keep looping!

```

图4-2描述了该算法的更新过程，初始情况 $\vec{w} = \vec{0}$ ，此时超平面为垂直平面，对于分类错误的数据一次更新 \vec{w} ，如图4-2中间所示，直到最后超平面将两类数据完全分开，算法终止。

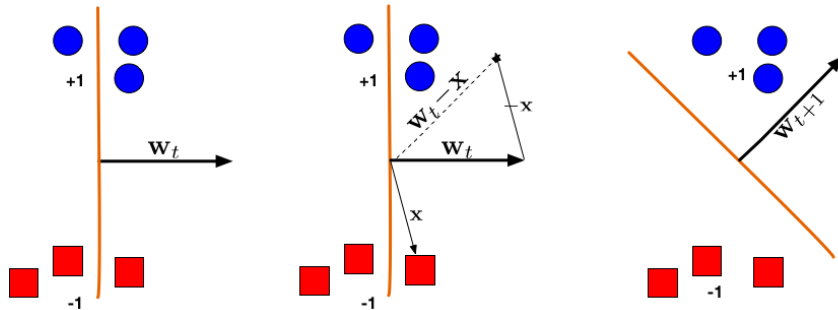


图 4-2 感知机算法运行过程

4.3 感知机收敛性

假设对于数据集 D , $\exists \vec{w}^*, y_i(\vec{w}^* \cdot \vec{x}) > 0 \forall (\vec{x}_i, y_i) \in D$ 。现在，我们对每一个数据点进行缩放，使得

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

我们记超平面为 γ ，其可定义为：

$$\gamma = \min_{(\vec{x}_i, y_i) \in D} |\vec{w}^* \cdot \vec{x}_i|$$

可视化如图4-3所示，我们有：

- 所有的数据点都在单位圆内
- γ 是超平面（蓝色线）距离最近的数据点的距离
- \vec{w}^* 在单位圆内

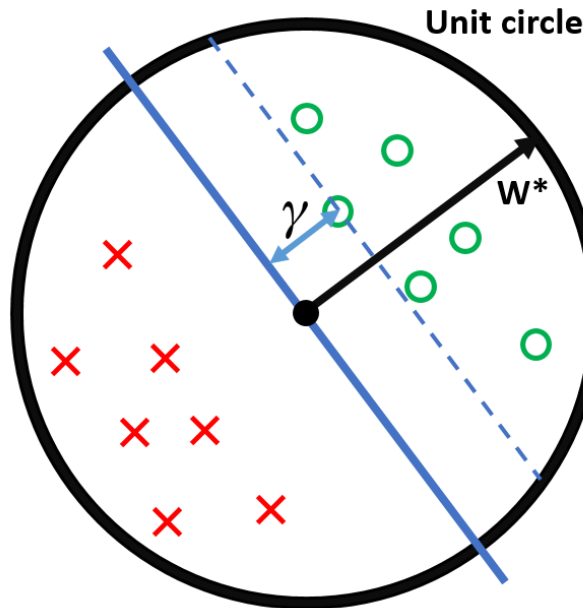


图 4-3 数据归一化

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

证明. 对于一次更新： \vec{w} 更新为 $\vec{w} + y\vec{x}$ ，我们考虑 $\vec{w} \cdot \vec{w}^*$ 和 $\vec{w} \cdot \vec{w}$ 。注意我们有 $y(\vec{x} \cdot \vec{w}) \leq 0$ (\vec{x} 被分类错误) 和 $y(\vec{x} \cdot \vec{w}^*) > 0$ 。对于一次更新，我们有

$$(\vec{w} + y\vec{x}) \cdot \vec{w}^* = \vec{w} \cdot \vec{w}^* + y(\vec{x} \cdot \vec{w}^*) \geq \vec{w} \cdot \vec{w}^* + \gamma$$

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

此外, 根据不等式 $2y(\vec{w} \cdot \vec{x}) < 0$ 和 $y^2(\vec{x} \cdot \vec{x}) \leq 1$, 我们有:

$$(\vec{w} + y\vec{x}) \cdot (\vec{w} + y\vec{x}) = \vec{w} \cdot \vec{w} + 2y(\vec{w} \cdot \vec{x}) + y^2(\vec{x} \cdot \vec{x}) \leq \vec{w} \cdot \vec{w} + 1$$

因此, 对于每一次更新, $\vec{w} \cdot \vec{w}$ 至多增加了 1。

假设我们有 M 次更新,

$$\begin{aligned} M\gamma &\leq \vec{w} \cdot \vec{w}^* \\ &= |\vec{w} \cdot \vec{w}^*| \\ &\leq \|\vec{w}\| \|\vec{w}^*\| && \text{柯西不等式} \\ &= \|\vec{w}\| && \|\vec{w}^*\| = 1 \\ &= \sqrt{\vec{w} \cdot \vec{w}} \\ &\leq \sqrt{M} \\ &\Rightarrow M\gamma \leq \sqrt{M} \\ &\Rightarrow M^2\gamma^2 \leq M \\ &\Rightarrow M \leq \frac{1}{\gamma^2} \end{aligned}$$

□

5

贝叶斯与概率估计

5.1 联合概率分布

构建概率模型的关键是定义一组随机变量，并考虑它们之间的联合概率分布，在给定联合概率分布的情况下，我们可以计算任何变量子集的条件或联合分布。那么该如何从观察到的训练数据中学习联合分布呢？

如果从一个庞大的数据库开始，例如其包含关于一百万人三个变量值得信息（表5-1），这将会很容易。给定诸如此类的大数据集，可以通过计算满足为该行指定值的数据库条目（人）的个数来容易地估计表中每行的概率。如果每行有数千个数据库条目，我们将使用此方法获得高度可靠的概率估计。

然而，在其他情况下，由于需要非常大量的训练数据，可能难以学习联合分布。如果我们在表5-1中添加描述总共一百个布尔特征的变量，表格中的行数将拓展 2^{100} 个，这个数字大于 10^{30} 。不幸的是，即使我们的数据库包含了地球上的每个人，我们也没有足够的数据来获得可靠的概率估计。地球上只有大约 10^{10} 人，这意味着对于我们表中的 10^{30} 行中的大多数来说，我们将没有训练样例！这是一个重要问题，因为实际应用中的机器学习过程用来描述每个样例的特征通常超过了 100 个。例如，许多用于文本分析的机器学习算法使用数百万个特征来描述给定文档中的文本。

为了从可用的训练数据中成功地解决学习概率的问题，我们必须根据观察到的数据聪明地估计概率参数。

为了便于理解，让我们从一个简单的例子开始：抛起一枚硬币，当它落地时，它可能正面朝上，也可能反面朝上。用随机变量 X 来代表这枚硬币，正面朝上记为 $X = 1$ ，反面朝上记为 $X = 0$ 。学习问题是，估计硬币正面朝上的例子，即估计 $P(X = 1)$ 。使用

表 5-1 出自 https://www.cs.cmu.edu/~tom/mlbook/Joint_MLE_MAP.pdf

Gender	HoursWorked	Wealth	probability
female	< 40.5	poor	0.2531
female	< 40.5	rich	0.0246
female	≥ 40.5	poor	0.0246
female	≥ 40.5	rich	0.0116
male	< 40.5	poor	0.0972
male	< 40.5	rich	0.0972
male	≥ 40.5	poor	0.1341
male	≥ 40.5	rich	0.1059

θ 来表示真实的（但未知）正面朝上的概率，用 $\hat{\theta}$ 表示我们对真实的 θ 的估计。抛硬币 n 次后，记录到硬币正面朝上 α_1 次，反面朝上 α_0 次，用 D 来表示观测到的结果。

为了得到 θ ，下面将介绍两个算法，MLE 和 MAP。

5.2 MLE

最大似然估计（MLE, Maximum Likelihood Estimation）的思路是，找出一个 θ 使得抛硬币结果出现 D 的概率最大（已知模型和数据，推测参数）。

总得来说，MLE 定义如下：

$$\hat{\theta}^{\text{MLE}} = \underset{\theta}{\operatorname{argmax}} P(D|\theta) \quad (5.1)$$

$P(D|\theta)$ 为似然函数，MLE 的目标即为找到使得似然函数取值最大的参数 θ

而 $P(D|\theta)$ 可以展开为：

$$P(D|\theta) = \theta^{\alpha_1} (1 - \theta)^{\alpha_0} \quad (5.2)$$

那么，

$$\begin{aligned} \hat{\theta}^{\text{MLE}} &= \underset{\theta}{\operatorname{argmax}} P(D|\theta) \\ &= \underset{\theta}{\operatorname{argmax}} \theta^{\alpha_1} (1 - \theta)^{\alpha_0} \\ &= \underset{\theta}{\operatorname{argmax}} [\alpha_1 \ln \theta + \alpha_0 \ln(1 - \theta)] \end{aligned}$$

设函数

$$f(\theta) = \alpha_1 \ln \theta + \alpha_0 \ln(1 - \theta)$$

则

$$\frac{\partial f}{\partial \theta} = \frac{\alpha_1 - (\alpha_1 + \alpha_0)\theta}{\theta(1-\theta)}$$

当 $\frac{\partial f}{\partial \theta} = 0$ 时,

$$\theta = \frac{\alpha_1}{\alpha_1 + \alpha_0} \quad (5.3)$$

综上所述我们得出

$$\hat{\theta}^{MLE} = \frac{\alpha_1}{\alpha_1 + \alpha_0} \quad (5.4)$$

从 MLE 得出的结果中，我们可以看出其仅仅只与观察数据有关，这意味着该算法对于观察数据的依赖性很高，数据的可靠性直接影响着算法得到的结果的可靠性。然而数据的可靠性与样本数量成正比，这里就是抛硬币的次数 n ，也就是说，若想用 MLE 得出可靠的结果， n 需要尽可能大。

5.3 MAP

在真实的机器学习任务中，很多时候采集大量的样本数据是不可能的或者代价非常高，这时可以引入先验分布来解决这个问题。假设参数 θ 有一个先验概率分布，比如说，经验告诉我们，硬币一般来说是匀称的，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。

MAP 定义如下：

$$\hat{\theta}^{MAP} = \arg \max_{\theta} P(\theta|D) \quad (5.5)$$

根据贝叶斯公式，上式可展开为：

$$\hat{\theta}^{MAP} = \arg \max_{\theta} P(\theta|D) = \arg \max_{\theta} \frac{P(D|\theta)P(\theta)}{P(D)}$$

因为 $P(D)$ 并不依赖于 θ ，故而可以忽略分母

$$\hat{\theta}^{MAP} = \arg \max_{\theta} P(\theta|D) = \arg \max_{\theta} P(D|\theta)P(\theta) \quad (5.6)$$

回到抛硬币的例子，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

$$P(\theta) = \text{Beta}(\beta_0, \beta_1) = \frac{\theta^{\beta_1-1}(1-\theta)^{\beta_0-1}}{B(\beta_0, \beta_1)} \quad (5.7)$$

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 。

$$\begin{aligned}\hat{\theta} &= \arg \max_{\theta} P(D|\theta)P(\theta) \\ &= \arg \max_{\theta} \theta^{\alpha_1}(1-\theta)^{\alpha_0} \frac{\theta^{\beta_1-1}(1-\theta)^{\beta_0-1}}{B(\beta_0, \beta_1)} \\ &= \arg \max_{\theta} \frac{\theta^{\alpha_1+\beta_1-1}(1-\theta)^{\alpha_0+\beta_0-1}}{B(\beta_0, \beta_1)} \\ &= \arg \max_{\theta} \theta^{\alpha_1+\beta_1-1}(1-\theta)^{\alpha_0+\beta_0-1}\end{aligned}\tag{5.8}$$

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

观察公式，发现其形式上与式5.2相似，因此从式5.2开始到式5.4为止，将 α_1 替换为 $\alpha_1 + \beta_1 - 1$ ，将 α_0 替换为 $\alpha_0 + \beta_0 - 1$ ，即可得出结果，即

$$\hat{\theta}^{MAP} = \arg \max_{\theta} P(D|\theta)P(\theta) = \frac{(\alpha_1 + \beta_1 - 1)}{(\alpha_1 + \beta_1 - 1) + (\alpha_0 + \beta_0 - 1)}\tag{5.9}$$

6

朴素贝叶斯法

朴素贝叶斯法 (Naive Bayes) 是基于贝叶斯定理与特征条件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 。

本章叙述朴素贝叶斯方法, 包括朴素贝叶斯法的学习与分类、参数估计以及典型样例等。

6.1 贝叶斯分类器

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 $D = \{(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)\}$, 大小为 n , 且由某个联合概率 $P(X, Y)$ 独立同分布产生。因为所有的数据点都服从独立同分布, 所以可以得到

$$P(D) = P((\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)) = \prod_{\alpha=1}^n P(\mathbf{x}_\alpha, y_\alpha). \quad (6.1)$$

若有足够的数据, 就可以用类似于前面讲过的抛硬币例子来估计 $P(X, Y)$ 。我们将联合分布 $P(X, Y)$ 想象成一个多面的骰子, 每个可能的数据点 (\mathbf{x}, y) 都代表骰子的一个面, 可以通过频率计算来估计一个特定面出现的概率

$$\hat{P}(\mathbf{x}, y) = \frac{\sum_{i=1}^n I(\mathbf{x}_i = \mathbf{x} \wedge y_i = y)}{n}, \quad (6.2)$$

其中, 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 $P(Y|X)$ 而不是 $P(X, Y)$ 。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

$$\hat{P}(y|\mathbf{x}) = \frac{\hat{P}(y, \mathbf{x})}{\hat{P}(\mathbf{x})} = \frac{\sum_{i=1}^n I(\mathbf{x}_i = \mathbf{x} \wedge y_i = y)}{\sum_{i=1}^n I(\mathbf{x}_i = \mathbf{x})}. \quad (6.3)$$

如图6-1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：

$$\hat{P}(y|\mathbf{x}) = \frac{|C|}{|B|}. \quad (6.4)$$

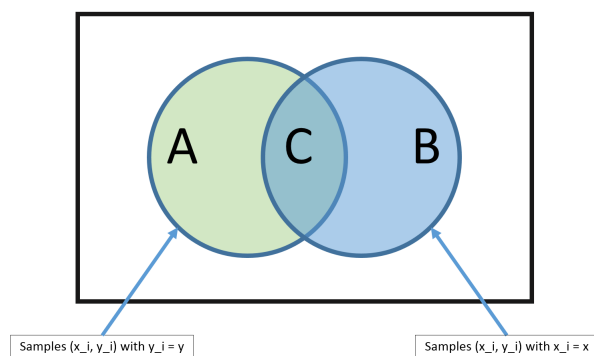


图 6-1 MLE 的韦恩图表示。

Figure 6-1 The Venn diagram illustrates the MLE method.

但用极大似然估计会出现一个问题，MLE 方法一般适用于有足够多的服从独立同分布的数据集的情况下。在高维特征空间或者特征 \mathbf{x} 是连续的情况下，会使得 $|B| \rightarrow 0$ 且 $|C| \rightarrow 0$ ，导致无法进行估计。

6.2 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问题。朴素贝叶斯法需要做一个恰当的假设，下文将会介绍。

首先，我们将上一节中式 (6.3) 做一个变换

$$P(y|\mathbf{x}) = \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})}. \quad (6.5)$$

估计 $P(y)$ 是容易的。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

$$P(y = c) = \frac{\sum_{i=1}^n I(y_i = c)}{n} = \hat{\pi}_c. \quad (6.6)$$

而估计 $P(\mathbf{x}|y)$ 是不太容易的，需要对其做一定的假设，可称其为朴素贝叶斯假设 (Naive Bayes Assumption)，具体而言是

$$P(\mathbf{x}|y) = \prod_{\alpha=1}^d P(x_{\alpha}|y), \quad (6.7)$$

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

朴素贝叶斯假设是一个较强的假设，等于是说用于分类的特征在类确定的条件下都是条件独立的。这一假设使朴素贝叶斯法变得简单，但有时会牺牲一定的分类准确率。

贝叶斯分类器经常会被用来进行垃圾邮件过滤，此时训练数据的特征就是邮件内容本身，而类别标签只有两类，即是垃圾邮件或不是垃圾邮件。朴素贝叶斯假设表达的含义就是邮件内容中的每个单词在已经知道邮件是否为垃圾邮件的情况下都是条件独立的。很显然这个假设实际上是不成立的，因为在写邮件的时候，我们不是从词库中独立地随机挑选单词来构成句子，而是根据上下文的关联性来组织单词。尽管朴素贝叶斯假设是违背现实的，但是在用朴素贝叶斯法进行实践的时候，却还有着不错的效果。

如图6-2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。在这些训练数据中，单独去估计特征的某个维度可能不能直接将两种分类类别区别开来，而朴素贝叶斯法可以容易地将两者区分。

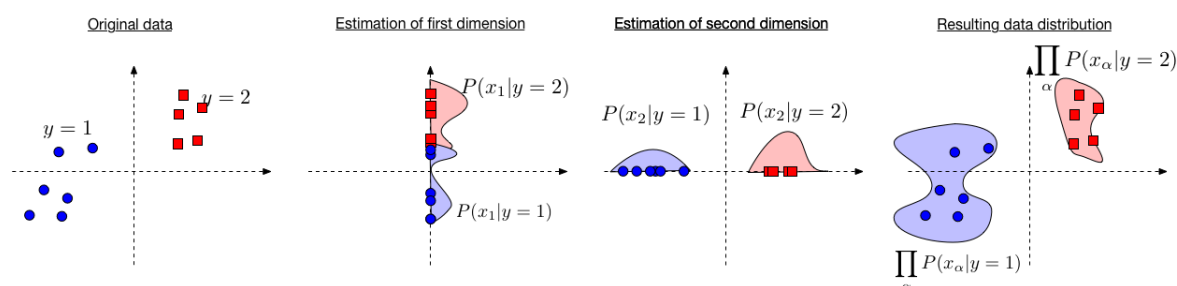


图 6-2 朴素贝叶斯法思想的图示。我们可以在每个输入特征维度上独立地估计 $P(x_{\alpha}|y)$ 的值（如中间两幅图所示），通过朴素贝叶斯假设 $P(\mathbf{x}|y) = \prod_{\alpha} P(x_{\alpha}|y)$ 可以进一步估计整体数据的分布（如最右边图所示）。

Figure 6-2 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

$$\begin{aligned}
 \mathbf{w} &= \operatorname{argmax}_{\mathbf{w}} P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \prod_{i=1}^n P(y_i, \mathbf{x}_i | \mathbf{w}) && (\text{由朴素贝叶斯假设可得}) \\
 &= \operatorname{argmax}_y \frac{P(\mathbf{x}|y)P(y)}{P(\mathbf{x})} \\
 &= \operatorname{argmax}_y P(\mathbf{x}|y)P(y) && (P(\mathbf{x}) \text{ 与 } y \text{ 无关}) \\
 &= \operatorname{argmax}_y \prod_{\alpha=1}^d P(x_{\alpha}|y)P(y) && (\text{由朴素贝叶斯假设可得}) \\
 &= \operatorname{argmax}_y \sum_{\alpha=1}^d \log(P(x_{\alpha}|y)) + \log(P(y)) && (\text{因为 } \log \text{ 是一个单调函数}).
 \end{aligned}$$

估计 $\log(P(x_{\alpha}|y))$ 的值是相对容易的，因为我们只需要考虑一个维度上的问题，并且 $P(y)$ 的估计不受该假设影响。接下来我们将介绍如何对朴素贝叶斯法的参数进行估计。

6.3 $P([\mathbf{x}]_{\alpha}|y)$ 参数估计

现在我们知道可以用上一节介绍的朴素贝叶斯假设来很方便地得到 $P(y|\mathbf{x})$ ，下面将介绍三个典型的实例来对参数进行估计并运用朴素贝叶斯法。

6.3.1 实例 1: 离散特征

该实例中的数据特征表现为离散的

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

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

$$P(x_{\alpha} = j | y = c) = [\theta_{jc}]_{\alpha} \quad (6.8)$$

$$\sum_{j=1}^{K_\alpha} [\theta_{jc}]_\alpha = 1, \quad (6.9)$$

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

$$[\hat{\theta}_{jc}]_\alpha = \frac{\sum_{i=1}^n I(y_i = c) I(x_{i\alpha} = j) + l}{\sum_{i=1}^n I(y_i = c) + l K_\alpha}, \quad (6.10)$$

其中 $x_{i\alpha} = [\mathbf{x}_i]_\alpha$, l 是平滑参数。如果取 $l = 0$ 那么式 (6.10) 即为 MLE, 如果 $l > 0$ 那么式 (6.10) 变成 MAP (Maximum a posteriori estimation), 如果取 $l = +1$ 那么式 (6.10) 为拉普拉斯平滑 (Laplace Smoothing)。

如果忽略平滑参数 l , 这就等价于

$$\frac{\text{数据中分类标签为 } c \text{ 且特征维度 } \alpha \text{ 取值为 } j \text{ 的个数}}{\text{数据中分类标签为 } c \text{ 的个数}}. \quad (6.11)$$

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, 如图 6-3。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。

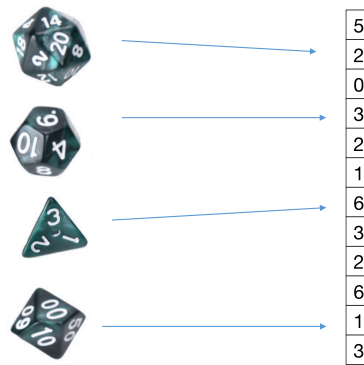


图 6-3 离散特征下的朴素贝叶斯法。对每一个分类类别, d 维的特征中的每一维都有一个对应的骰子, 我们假设训练数据点的特征值是由这些骰子一一对应投掷得到的。

Figure 6-3 Illustration of categorical NB.

所以，我们可以得到预测方法

$$\operatorname{argmax}_y P(y = c | \mathbf{x}) \propto \operatorname{argmax}_y \hat{\pi}_c \prod_{\alpha=1}^d [\hat{\theta}_{jc}]_{\alpha}. \quad (6.12)$$

6.3.2 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 $x_{\alpha} = 10$ 那么有可能这封邮件是垃圾邮件，因为“spam”这个词在邮件中出现了 10 次；如果一封邮件它的 $x'_{\alpha} = 20$ ，那么相比前者这一封更可能是垃圾邮件，但是实例 1 中的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

$$x_{\alpha} \in \{0, 1, 2, \dots, m\} \text{ 且 } m = \sum_{\alpha=1}^d x_{\alpha}. \quad (6.13)$$

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 $P(\mathbf{x} | y)$

$$P(\mathbf{x} | m, y = c) = \frac{m!}{x_1! \cdot x_2! \cdot \dots \cdot x_d!} \prod_{\alpha=1}^d (\theta_{\alpha c})^{x_{\alpha}}, \quad (6.14)$$

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 $P(\mathbf{x} | y = \text{spam})$ ，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

$$\hat{\theta}_{\alpha c} = \frac{\sum_{i=1}^n I(y_i = c) x_{i\alpha} + l}{\sum_{i=1}^n I(y_i = c) m_i + l \cdot d}, \quad (6.15)$$

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

$$\frac{\text{所有垃圾邮件中含有单词}\alpha\text{的个数}}{\text{所有垃圾邮件的单词总数}}. \quad (6.16)$$

所以，我们可以得到预测方法

$$\operatorname{argmax}_c P(y = c | \mathbf{x}) \propto \operatorname{argmax}_c \hat{\pi}_c \prod_{\alpha=1}^d \hat{\theta}_{\alpha c}^{x_\alpha} \quad (6.17)$$

6.3.3 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

$$x_\alpha \in \mathbb{R} \quad (\text{每一维的特征都取实数值}). \quad (6.18)$$

用高斯分布来表征 $P(x_\alpha | y)$

$$P(x_\alpha | y = c) = \mathcal{N}(\mu_{\alpha c}, \sigma_{\alpha c}^2) = \frac{1}{\sqrt{2\pi}\sigma_{\alpha c}} e^{-\frac{1}{2}\left(\frac{x_\alpha - \mu_{\alpha c}}{\sigma_{\alpha c}}\right)^2}. \quad (6.19)$$

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

$$\mu_{\alpha c} \leftarrow \frac{1}{n_c} \sum_{i=1}^n I(y_i = c) x_{i\alpha} \quad (\text{其中 } n_c = \sum_{i=1}^n I(y_i = c)) \quad (6.20)$$

$$\sigma_{\alpha c}^2 \leftarrow \frac{1}{n_c} \sum_{i=1}^n I(y_i = c) (x_{i\alpha} - \mu_{\alpha c})^2. \quad (6.21)$$

6.4 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

$$h(\mathbf{x}) = \operatorname{argmax}_y P(y) \prod_{\alpha=1}^d P(x_\alpha | y) = \operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b). \quad (6.22)$$

也就是说朴素贝叶斯法是一种线性分类器，图6-4形象地描述了这一性质，具体表达为

$$\mathbf{w}^\top \mathbf{x} + b > 0 \iff h(\mathbf{x}) = +1. \quad (6.23)$$

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

$$[\mathbf{w}]_\alpha = \log(\theta_{\alpha+}) - \log(\theta_{\alpha-}) \quad (6.24)$$

$$b = \log(\pi_+) - \log(\pi_-). \quad (6.25)$$

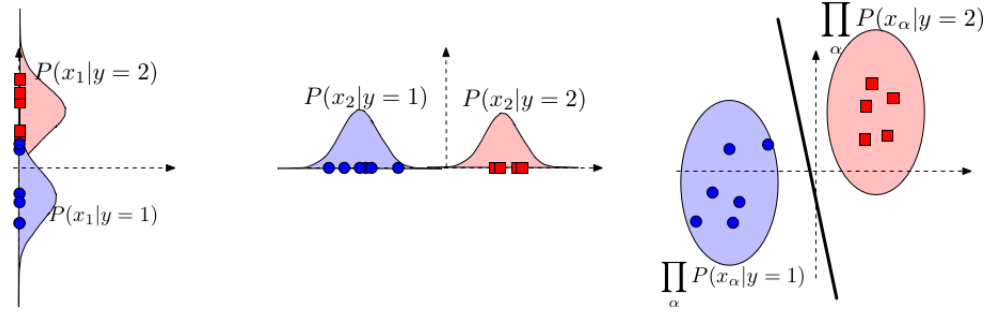


图 6-4 在很多情况下，朴素贝叶斯法将生成一个线性决策边界。此处假定 $P(x_\alpha | y)$ 满足高斯分布且对每种类别 c 标准差 $\Sigma_{\alpha,c}$ 都是理想的（尽管可能不同的 α 维度上值不相同）。最右侧图中黑色的直线表示决策边界，即 $P(y = 1|\mathbf{x}) = P(y = 2|\mathbf{x})$ 。

Figure 6-4 Naive Bayes leads to a linear decision boundary in many common cases.

如果用式 (6.24) 和式 (6.25) 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

$$P(y | \mathbf{x}) = \frac{1}{1 + e^{-y(\mathbf{w}^\top \mathbf{x} + b)}} \quad (6.26)$$

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

6.5 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6-1的训练数据学习一个朴素贝叶斯分类器并预测 $\mathbf{x} = (2, S)^T$ 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 $A_1 = \{1, 2, 2\}$ 、 $A_2 = \{S, M, L\}$ ， 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 6-1 训练数据

Table 6-1 Training data

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$[\mathbf{x}]_1$	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
$[\mathbf{x}]_2$	<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>
y	-1	-1	1	1	-1	-1	-1	1	1	1	1	1	1	1	-1

7

逻辑回归

7.1 逻辑回归

本章讲述了与高斯朴素贝叶斯对应的判别模型逻辑回归。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。在之前的章节中，我们介绍了高斯朴素贝叶斯 $P(y|x_i) = \frac{1}{1 + e^{-y(\mathbf{w}^T x_i + b)}}$ ，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 $\prod_i P(y_i|x_i; \mathbf{w}, b)$ 。

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。

7.2 最大似然估计 (MLE)

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：

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

取对数，得到：

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

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 $l(\mathbf{w}) = \sum_{i=1}^n \log(1 + e^{-y_i \mathbf{w}^T x_i})$ 上应用梯度下降寻找最优的参数。

7.3 最大后验估计 (MAP)

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} 。

7.4 总结

逻辑回归是与朴素贝叶斯对应的判别模型。在朴素贝叶斯中，对于每一个标签 y ，我们都为其建模 $P(x|y)$ ，之后求解最能区分这两种分布的决策边界。在逻辑回归中，我们不对数据分布建立模型 $P(x|y)$ ，而是直接对 $P(y|x)$ 建模。逻辑回归具有与朴素贝叶斯同形的概率函数形式 $P(y|x_i) = \frac{1}{1+e^{-y(\mathbf{w}^T x_i + b)}}$ ，却没有对 $P(x|y)$ 做任何假设，这使得逻辑回

归更加灵活，但这种灵活性使得模型更易过拟合，因此需要更多的训练数据。在数据很少的情况下，如果模型假设合适，朴素贝叶斯往往优于逻辑回归。然而，随着数据集增大，逻辑回归往往优于朴素贝叶斯，朴素贝叶斯的缺点是对 $P(x|y)$ 的假设可能并不完全正确。如果假设正确，即数据是从我们在朴素贝叶斯中假设的分布中提取出来的，那么逻辑回归和朴素贝叶斯收敛的结果将会完全相同（但朴素贝叶斯收敛地更快）。

8

梯度下降法

我们希望最小化一个凸的, 连续的, 可微的代价函数 $l(w)$. 在本节中, 我们讨论两种最受欢迎的“爬山算法”: 梯度下降法和牛顿法。

算法:

初始化 \mathbf{w}_0

重复以下步骤直到收敛:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \mathbf{s}$$

if $\|\mathbf{w}^{t+1} - \mathbf{w}^t\|_2 < \epsilon$, 判定收敛!

8.1 泰勒展开

如果你对一个函数 l 了解不多, 怎样将它最小化呢? 窍门就是把它当成一个比它真实难度小得多的函数。这可以用泰勒近似来实现。假设范数 $\|\mathbf{s}\|_2$ 很小 (也就是说 $\mathbf{w} + \mathbf{s}$ 非常接近 \mathbf{w}), 我们可以用函数 $l(\mathbf{w} + \mathbf{s})$ 的一阶和二阶导数来近似:

$$l(\mathbf{w} + \mathbf{s}) \approx l(\mathbf{w}) + g(\mathbf{w})^T \mathbf{s}$$

$$l(\mathbf{w} + \mathbf{s}) \approx l(\mathbf{w}) + g(\mathbf{w})^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T H(\mathbf{w}) \mathbf{s}$$

这里, $g(\mathbf{w}) = \nabla l(\mathbf{w})$ 代表梯度, $H(\mathbf{w}) = \nabla^2 l(\mathbf{w})$ 代表 l 的 Hessian 矩阵。如果 $\|\mathbf{s}\|_2$ 足够小, 两种近似都是有效的。如果 l 二阶可导的话, 第二种方法更加精确, 不过需要消耗更多的计算资源。

8.2 梯度下降法：使用一阶近似

在梯度下降法中我们仅仅使用梯度（一阶）。换言之，我们假设函数 l 在 \mathbf{w} 附近是线性的并且近似于 $\ell(\mathbf{w}) + g(\mathbf{w})^\top \mathbf{s}$ 。我们的目标是找到一个向量 \mathbf{s} 使这个函数最小化。在最陡下降中，对于小的 $\alpha > 0$ ，我们设定：

$$\mathbf{s} = -\alpha g(\mathbf{w})$$

当 $\ell(\mathbf{w} + \mathbf{s}) < \ell(\mathbf{w})$ 时，可以直接证明出：

$$\underbrace{\ell(\mathbf{w} + (-\alpha g(\mathbf{w})))}_{\text{after one update}} \approx \underbrace{\ell(\mathbf{w}) - \alpha \overbrace{g(\mathbf{w})^\top g(\mathbf{w})}^{>0}}_{>0} < \underbrace{\ell(\mathbf{w})}_{\text{before}}$$

设定学习速率 $\alpha > 0$ 是一项黑暗的艺术。只有当它足够小时，梯度下降才会收敛（如图 1-1 左所示）。如果它太大，算法很容易发散失控（如图 1-1 右所示）。一个保险起见（但有时会慢）的选择是设定 $\alpha = \frac{t_0}{t}$ ，这可以保证它最终变得足够小使得算法收敛。

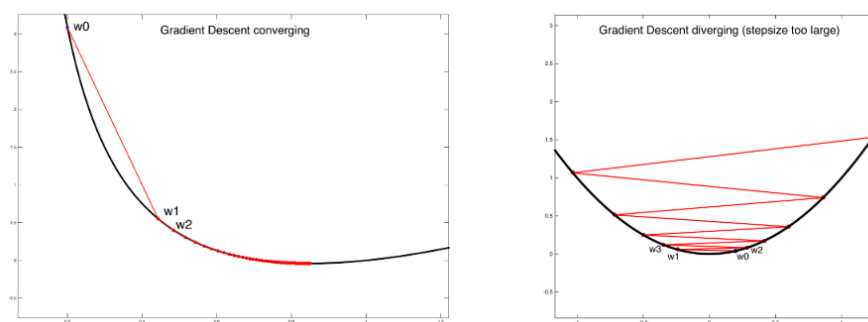


图 8-1

8.3 自适应梯度下降法 Adagrad

一种方法是对每一个特征自适应设定步长。Adagrad 算法保留梯度大小平方的滑动平均，并且对于大梯度的特征设置一个小的学习速率，对于小梯度的特征设置一个大的学习速率。如果不同的特征在数量级上相差很大或者在频率上相差很大，为它们设置不同的学习速率非常重要。比如说，词数统计和共同词、生僻词的差别就很大。

Adagrad 算法：

初始化 \mathbf{w}_0 和 \mathbf{z} : $\forall d: w_d^0 = 0$ 且 $z_d = 0$

重复以下步骤直到收敛:

$$\mathbf{g} = \frac{\partial f(\mathbf{w})}{\partial \mathbf{w}} \text{ // 计算梯度}$$

$$\forall d: z_d \leftarrow z_d + g_d^2$$

$$\forall d: w_d^{t+1} \leftarrow w_d^t - \alpha \frac{g_d}{\sqrt{z_d + \epsilon}}$$

if $\|\mathbf{w}^{t+1} - \mathbf{w}^t\|_2 < \delta$, 收敛! // 对于一些小的 $\delta > 0$

8.4 牛顿法：使用二阶近似

牛顿法假定代价函数 l 二阶可微并且使用 Hessian 矩阵 (二阶泰勒近似) 来近似。Hessian 矩阵包含所有二阶偏导，定义方式如图 1-2 所示：

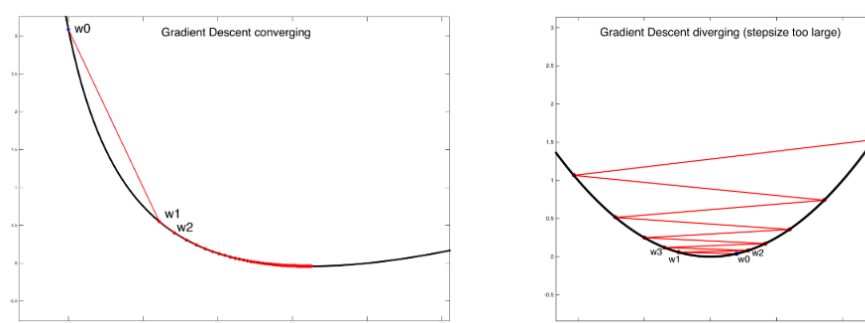


图 8-2

由于函数 l 的正定性，用 Hessian 矩阵是对称平方矩阵，也是半正定矩阵。

注：我们称一个矩阵 \mathbf{M} 为半正定矩阵，如果它仅仅有非负的特征值。换言之，对于任意向量 \mathbf{x} ，都满足 $\mathbf{x}^T \mathbf{M} \mathbf{x} \geq 0$ 。

它用以下方法近似：

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

等式右边是一个凸的抛物线，所以我们可以通过求解以下优化方程来找到它的最小值：

$$\operatorname{argmin}_s \ell(\mathbf{w} + \mathbf{s}) \approx \ell(\mathbf{w}) + \mathbf{g}(\mathbf{w})^T \mathbf{s} + \frac{1}{2} \mathbf{s}^T \mathbf{H}(\mathbf{w}) \mathbf{s}$$

为了找到目标函数的最小值，我们对变量 \mathbf{s} 求它的一阶导数，令它等于 0，来求解 \mathbf{s} ：

$$\mathbf{g}(\mathbf{w}) + \mathbf{H}(\mathbf{w}) \mathbf{s} = 0 \tag{8.1}$$

$$\Rightarrow \mathbf{s} = -[\mathbf{H}(\mathbf{w})]^{-1} \mathbf{g}(\mathbf{w}). \tag{8.2}$$

如果近似足够精确且得出的步数足够小， \mathbf{s} 的选择可以收敛非常快。否则它会发散。如果函数“平坦”或者在某些维度上近似“平坦”，发散经常会发生。在这种情况下，二阶导数接近于 0，而二阶导数的倒数会变得非常之大，这导致巨量的步数。不同于梯度下降法，这里没有步长可以保证步数足够小而且局限在“附近”。由于泰勒近似仅仅在“附近”精确，大量的步数可能会导致当前的估计移动到泰勒近似不再精确的区域。

8.5 最佳实践

1. 矩阵 $H(\mathbf{w})$ 的维数是 $H(\mathbf{w})$ ，计算起来很耗时。一个好的近似方法是，仅仅计算它的对角线元素，并让它与很小步长的更新数据相乘。本质上讲，你接下来把牛顿法和梯度下降法做一个混合，到那时你根据 Hessian 矩阵的逆矩阵来为每个维度设置步长。

2. 为了避免牛顿法的发散，一种好的方法是用梯度下降法 (甚至是随机梯度下降法) 开始，再用牛顿法完成优化。通常来说，使用牛顿法的二阶近似，在最优点附近更加适合。

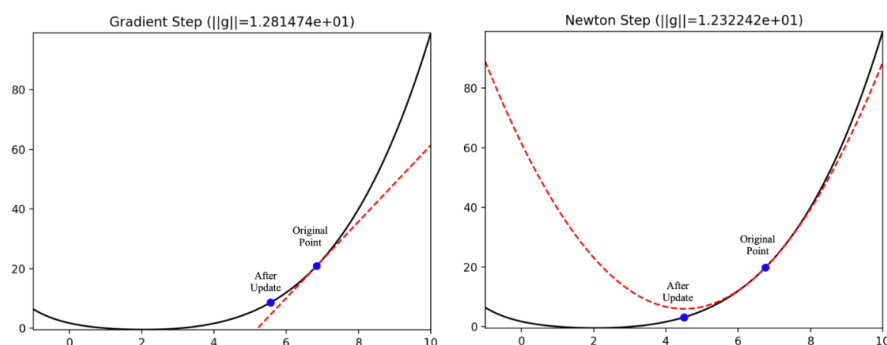


图 8-3 梯度下降法 (左图) 和牛顿法 (右图)。黑线是代价函数，红色虚线是近似函数。梯度下降法沿着函数的线性近似向下移动点。牛顿法则把点移动到近似抛物线的最小值。

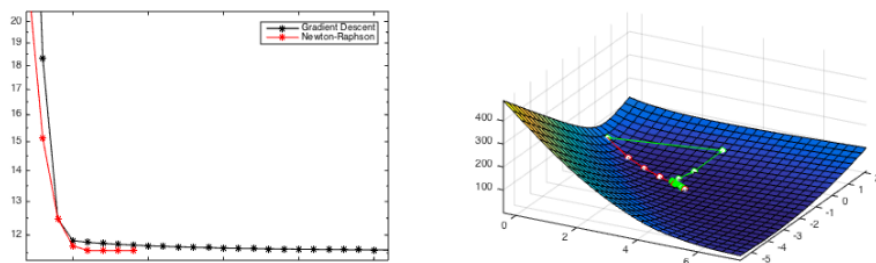


图 8-4 使牛顿法在 8 次迭代后收敛的一个起始点

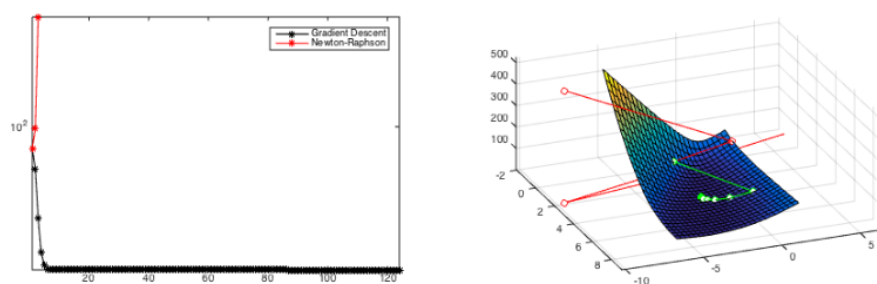


图 8-5 使牛顿法发散的一个起始点

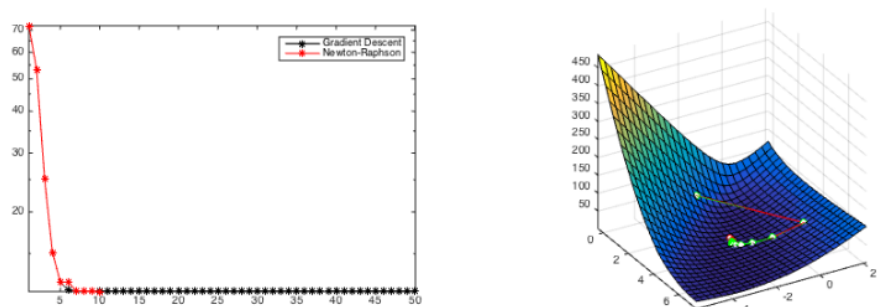


图 8-6 和图 1-4 相同的起始点，但是在使用梯度下降法 6 次后才使用牛顿法。可以发现几步后就收敛

图 1-4，图 1-5，图 1-6 是牛顿法和梯度下降法的对比。梯度下降法从所有起始点经过 100 次以上的迭代后往往是收敛的。如果梯度下降法收敛的话 (如图 1-4 所示)，牛顿法可以更快 (在 8 次迭代后收敛)，但是牛顿法可能发散 (如图 1-5 所示)。图 1-6 所示是两种方法的混用，在使用梯度下降法执行 6 步后切换到牛顿法。它仅仅在 10 次更新后就收敛了。

9

线性回归

9.1 假设

数据假设: $y_i \in \mathbb{R}$

模型假设: $y_i = \mathbf{w}^\top \mathbf{x}_i + \epsilon_i$ where $\epsilon_i \sim N(0, \sigma^2)$

$$\Rightarrow y_i | \mathbf{x}_i \sim N(\mathbf{w}^\top \mathbf{x}_i, \sigma^2) \Rightarrow P(y_i | \mathbf{x}_i, \mathbf{w}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(\mathbf{x}_i^\top \mathbf{w} - y_i)^2}{2\sigma^2}}$$

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, 可以从其它维度添加偏置)。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} 。

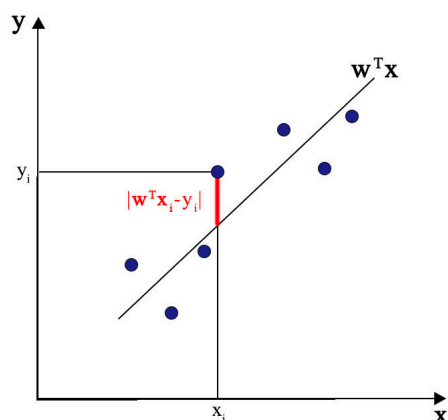


图 9-1 线性回归

Figure 9-1 Linear Regression

9.2 最大似然估计

$$\mathbf{w} = \underset{\mathbf{w}}{\operatorname{argmax}} P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w})$$

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^n P(y_i, \mathbf{x}_i | \mathbf{w})$$

由独立性可得

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i | \mathbf{w})$$

链式法则

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i)$$

 \mathbf{x}_i 与 \mathbf{w} 无关

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w})$$

 $P(\mathbf{x}_i)$ 是常数

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^n \log[P(y_i | \mathbf{x}_i, \mathbf{w})]$$

log 是单调函数

$$= \underset{\mathbf{w}}{\operatorname{argmax}} \sum_{i=1}^n \left[\log \left(\frac{1}{\sqrt{2\pi\sigma^2}} \right) + \log \left(e^{-\frac{(\mathbf{x}_i^\top \mathbf{w} - y_i)^2}{2\sigma^2}} \right) \right]$$

插入概率分布

$$= \underset{\mathbf{w}}{\operatorname{argmax}} -\frac{1}{\sqrt{2\sigma^2}} \sum_{i=1}^n (\mathbf{x}_i^\top \mathbf{w} - y_i)^2$$

第一项是常数, 且 $\log(e^z) = z$

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

最小化; 为便于解释用 $\frac{1}{n}$ 取均方误差

即优化损失函数 $l(\mathbf{w}) = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^\top \mathbf{w} - y_i)^2$ 。这个损失函数又被称作平方损失或最小二乘 (OLS)。OLS 可以通过梯度下降或牛顿法进行优化，也可以给出闭式解。

闭式解： $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{y}^\top$ where $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ and $\mathbf{y} = [y_1, \dots, y_n]$.

9.3 最大后验估计

额外模型假设： $P(\mathbf{w}) = \frac{1}{\sqrt{2\pi\tau^2}} e^{-\frac{\mathbf{w}^\top \mathbf{w}}{2\tau^2}}$

$$\begin{aligned}
 \mathbf{w} &= \operatorname{argmax}_{\mathbf{w}} P(\mathbf{w} | y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n) \\
 &= \operatorname{argmax}_{\mathbf{w}} \frac{P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w}) P(\mathbf{w})}{P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n)} \\
 &= \operatorname{argmax}_{\mathbf{w}} P(y_1, \mathbf{x}_1, \dots, y_n, \mathbf{x}_n | \mathbf{w}) P(\mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i, \mathbf{x}_i | \mathbf{w}) \right] P(\mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i | \mathbf{w}) \right] P(\mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) P(\mathbf{x}_i) \right] P(\mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \left[\prod_{i=1}^n P(y_i | \mathbf{x}_i, \mathbf{w}) \right] P(\mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \sum_{i=1}^n \log P(y_i | \mathbf{x}_i, \mathbf{w}) + \log P(\mathbf{w}) \\
 &= \operatorname{argmax}_{\mathbf{w}} \frac{1}{2\sigma^2} \sum_{i=1}^n (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 + \frac{1}{2\tau^2} \mathbf{w}^\top \mathbf{w} \\
 &= \operatorname{argmax}_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 + \lambda \|\mathbf{w}\|_2^2 \quad \lambda = \frac{\sigma^2}{n\tau^2}
 \end{aligned}$$

这种方法叫做岭回归 (Ridge Regression)。闭式解为： $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1} \mathbf{X}\mathbf{y}^\top$ where $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$ and $\mathbf{y} = [y_1, \dots, y_n]$

9.4 总结

最小二乘：

- ▶ $\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^\top \mathbf{w} - y_i)^2$
- ▶ 平方损失
- ▶ 无正则化
- ▶ 闭式解： $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top)^{-1} \mathbf{X}\mathbf{y}^\top$

岭回归：

- ▶ $\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i^\top \mathbf{w} - y_i)^2 + \lambda \|\mathbf{w}\|_2^2$
- ▶ 平方损失
- ▶ l2-正则化
- ▶ 闭式解： $\mathbf{w} = (\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I})^{-1} \mathbf{X}\mathbf{y}^\top$

10

SVM

Support Vector Machines (SVM)

10.1 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：

$$R_D(h) = \Pr_{x \sim D} [h(x) \neq f(x)]. \quad (10.1)$$

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：

$$H = \{\mathbf{x} \mapsto \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^N, b \in \mathbb{R}\}. \quad (10.2)$$

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”。

10.2 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。

10.2.1 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 $x_0 \in \mathbb{R}^N$ 到超平面 $\mathbf{w} \cdot \mathbf{x} + b = 0$ 的距离为：

$$\frac{|\mathbf{w} \cdot \mathbf{x}_0 + b|}{\|\mathbf{w}\|}. \quad (10.3)$$

对于规范超平面，定义其边界大小 ρ 为：

$$\rho = \min_{(\mathbf{x}, y) \in S} \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \quad (10.4)$$

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：

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{subject to:} \quad & y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1, \forall i \in [1, m] \end{aligned} \quad (10.5)$$

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。

10.2.2 对偶优化问题

对优化问题（10.5）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

$$\begin{aligned} g(\alpha) &= \inf_{\mathbf{w}, b} L(\mathbf{w}, b, \alpha) \\ &= \sum_{i=1}^m \alpha_i + \inf_{\mathbf{w}} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - \left(\sum_{i=1}^m \alpha_i y_i \mathbf{x} \right) \cdot \mathbf{w} \right\} + \inf_b \left\{ - \sum_{i=1}^m \alpha_i y_i b \right\} \\ &= \begin{cases} \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) & \text{如果 } \sum_{i=1}^m \alpha_i y_i = 0 \\ -\infty & \text{其他情况} \end{cases} \end{aligned} \quad (10.6)$$

因而，对偶问题可以表述为：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{subject to :} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \\ & \alpha_i \geq 0, \quad i = 1, \dots, m \end{aligned} \quad (10.7)$$

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）。

10.2.3 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：

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \quad (10.8)$$

$$\nabla_b L = - \sum_{i=1}^m \alpha_i y_i = 0 \implies \sum_{i=1}^m \alpha_i y_i = 0 \quad (10.9)$$

$$\forall i, \alpha_i [y_i ((\mathbf{w} \cdot \mathbf{x}_i + b) - 1)] = 0 \implies \alpha_i = 0 \vee y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1. \quad (10.10)$$

从互补松弛条件（10.10）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（10.8）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**。

10.3 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

$$y_i [\mathbf{w} \cdot \mathbf{x}_i + b] < 1. \quad (10.11)$$

为此，我们引入松弛变量 $\xi_i \geq 0$ ，使得

$$y_i [\mathbf{w} \cdot \mathbf{x}_i + b] \geq 1 - \xi_i. \quad (10.12)$$

这样一来，主优化问题改写成：

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^m \xi_i^p \\ \text{subject to :} \quad & y_i(\mathbf{w} \cdot \mathbf{x} + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, i \in [1, m] \end{aligned} \quad (10.13)$$

下面只考虑 $p = 1$ 时的情形。

10.3.1 对偶优化问题

对优化问题 (10.13) 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

$$\begin{aligned} g(\alpha, \beta) &= \inf_{\mathbf{w}, b, \xi} L(\mathbf{w}, b, \xi, \alpha, \beta) \\ &= \sum_{i=1}^m \alpha_i \\ &\quad + \inf_{\mathbf{w}} \left\{ \frac{1}{2} \|\mathbf{w}\|^2 - \left(\sum_{i=1}^m \alpha_i y_i \mathbf{x} \right) \cdot \mathbf{w} \right\} \\ &\quad + \inf_b \left\{ - \sum_{i=1}^m \alpha_i y_i b \right\} \\ &\quad + \inf_{\xi} \left\{ \sum_{i=1}^m (C - \alpha_i - \beta_i) \xi_i \right\} \\ &= \begin{cases} \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j), & \text{如果 } \sum_{i=1}^m \alpha_i y_i = 0 \text{ 且 } \alpha_i + \beta_i = C, \\ -\infty, & \text{其他情况。} \end{cases} \end{aligned} \quad (10.14)$$

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：

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \\ \text{subject to :} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \end{aligned} \quad (10.15)$$

10.3.2 KKT 条件

线性不可分问题的最优解满足如下 KKT 条件：

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i = 0 \implies \mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i \quad (10.16)$$

$$\nabla_b L = - \sum_{i=1}^m \alpha_i y_i = 0 \implies \sum_{i=1}^m \alpha_i y_i = 0 \quad (10.17)$$

$$\nabla_{\xi_i} L = C - \alpha_i - \beta_i = 0 \implies \alpha_i + \beta_i = C \quad (10.18)$$

$$\forall i, \alpha_i [y_i ((\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i)] = 0 \implies \alpha_i = 0 \vee y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1 - \xi_i. \quad (10.19)$$

$$\forall i, \beta_i \xi_i = 0 \implies \beta_i = 0 \vee \xi_i = 0. \quad (10.20)$$

结合 (10.18) 和 (10.19), (10.20) 可改写成：

$$\alpha_i = C \vee y_i (\mathbf{w} \cdot \mathbf{x}_i + b) = 1. \quad (10.21)$$

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 。

10.4 核函数

从对偶问题 (10.7) 和 (10.15) 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。

10.4.1 多项式核函数

对常数 $c > 0$ ，定义 d 维多项式核函数为：

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + c)^d \quad (10.22)$$

10.4.2 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:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x}' - \mathbf{x}\|^2}{2\sigma^2}\right) \quad (10.23)$$

10.4.3 sigmoid 核函数

对常数 $a, b \geq 0$, 定义 sigmoid 核函数 为:

$$\forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^N, K(\mathbf{x}, \mathbf{x}') = \tanh(a(\mathbf{x} \cdot \mathbf{x}') + b) \quad (10.24)$$

因此, 基于核函数的 SVM 表述为如下问题:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^m \alpha_i - \sum_{i,j=1}^m \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ \text{subject to :} \quad & \sum_{i=1}^m \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \end{aligned} \quad (10.25)$$

11

经验风险最小化

机器学习的目的是通过训练样本，寻找一个最优的函数，使得函数对输入的估计与实际的输出之间的期望风险最小化。期望风险最小化依赖于样本的输入与其输出之间的函数映射关系。在机器学习系统中，一般指代先验概率和类条件概率，然而这两者在实际应用中，都无法准确获得。因此，在实际中，常常利用样本的算术平均值来替代理想的期望，利用已知的经验数据（训练样本）来计算得到的误差（经验风险）。经验风险最小化（Empirical risk minimization, ERM）是指对参数求经验风险来逼近理想的期望风险的最小值。

本章叙述经验风险最小化，包括经验风险的定义、经验风险最小化与结构风险最小化。

11.1 经验风险

11.1.1 损失函数

给定样本 x 及其标签 y ，假设存在模型对 x 的标签进行预测，将其记为 $\hat{y} = f(x)$ ，那么损失函数就是衡量预测值与真实值差距的函数，记为 $L(y, \hat{y})$ 。例如平方损失函数记为

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

11.1.2 风险函数

设随机变量 (X, Y) 满足联合分布 $P(X, Y)$, 风险函数就是损失函数的期望

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

由于现实中 (X, Y) 的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。

11.1.3 经验风险

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 $N \rightarrow \infty$, $R_{\text{emp}}(f) \rightarrow R_{\text{exp}}(f)$

11.2 经验风险最小化与结构风险最小化

11.2.1 经验风险最小化

已知经验风险为 $R_{\text{emp}}(f)$, 经验风险最小化定义如下

$$\min_{f \in F} \frac{1}{N} \sum_{i=1}^N L(y_i, f(x_i))$$

当样本量足够多的时候, 经验风险最小化可以保证很好的效果, 但是当样本小的时候, 很容易过拟合。

当损失函数是对数损失函数的时候, 经验风险最小化就等价于极大似然估计。给定样本集合 $T = \{(x_1, y_1), (x_2, y_2), \dots, (x_N, y_N)\}$, 已经有关于随机变量 (X, Y) 的概率分布形式 $P(X, Y; \theta)$, 但是参数 θ 需要估计, 那么使用极大似然估计有

$$\max_{\theta} \prod_{i=1}^N P(y_i | x_i; \theta)$$

取对数, 可得

$$\max_{\theta} \log\left(\prod_{i=1}^N P(y_i | x_i; \theta)\right) = \max_{\theta} \sum_{i=1}^N \log P(y_i | x_i; \theta) = \min_{\theta} \sum_{i=1}^N -\log P(y_i | x_i; \theta)$$

定义损失函数 $L(y, P(y|x)) = -\log P(y|x)$ ，那么可以得到

$$\min_{\theta} \sum_{i=1}^N -\log P(y_i|x_i; \theta) = \min_{\theta} \sum_{i=1}^N L(y_i, P(y_i|x_i; \theta))$$

等价于

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^N L(y_i, P(y_i|x_i; \theta))$$

也就是经验风险最小化的形式。

11.2.2 结构风险最小化

当样本数据量小的时候，很容易发生过拟合。为了防止过拟合，需要在经验风险的基础上加上表示模型结构复杂度的惩罚项（正则化项），如下

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

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。可以看到，当模型的经验风险越来越小的时候，模型的复杂度则会越来越大。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。

12

模型调试

对于经验风险最小化 (ERM) 模型

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^n \underbrace{l_{(s)}(h_{\mathbf{w}}(\mathbf{x}_i), y_i)}_{\text{Loss}} + \underbrace{\lambda r(\mathbf{w})}_{\text{Regularizer}} \quad (12.1)$$

该如何设定 λ 来权衡偏差和方差呢？

12.1 过拟合和欠拟合

在数据集上学习分类器时，可能出现两种问题情况——过拟合和欠拟合，每种情况都与推断训练集中的数据适用于未知数据的程度有关。

欠拟合：学习到的分类器对于训练集的数据都无法很好地解释，由于没有充分考虑到训练集中的信息，在这种情况下下的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过于具体，无法用于准确推断未知数据。尽管训练误差会不断降低，但是由于分类器开始依赖仅出现于训练集中而不广泛存在的关系进行决策，测试误差将再次开始增大。

12.2 确定最佳位置

我们将数据集分成训练集和验证集。针对不同的 λ (一般选取 $10^{-5} 10^{-4} 10^{-3} 10^{-2} 10^{-1} 10^0 10^1 10^2 \dots$) 我们在训练集上训练模型，并且在验证集上进行验证。

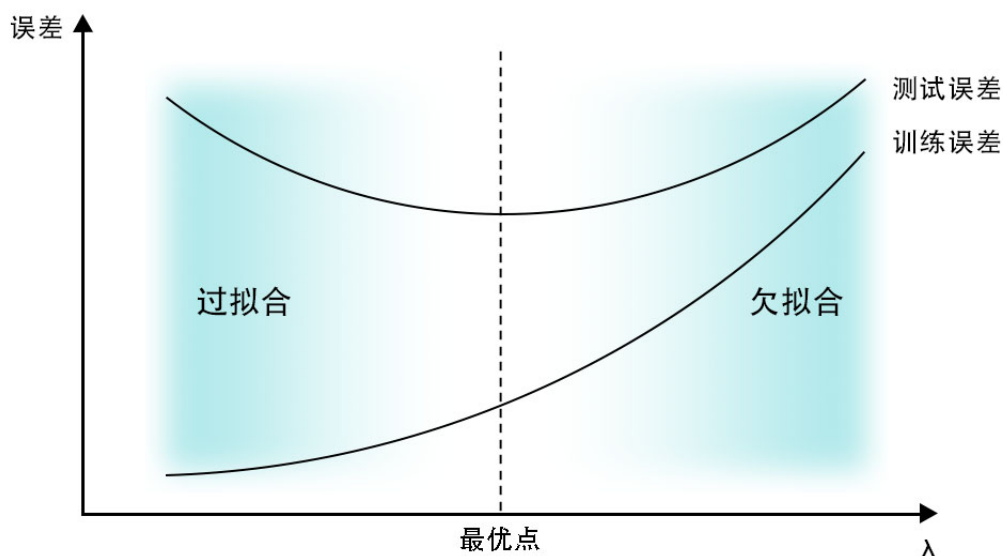


图 12-1 过拟合和欠拟合

12.2.1 K-Fold 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 (即每个集合上会运行一次), 之后再平均每次运行的验证误差, 这样可以很好的忽视数据存在的标准差。在极端情况下, 可以令 $k = n$, 即每次只用一个数据点 (也就是留一法交叉验证)。留一法对于小的数据集有很好的表现。

12.2.2 迭代搜索

进行两步搜索：一，寻找 λ 的最佳数量级。二，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 $\lambda = 5, 10, 15, 20, 25, \dots, 95$ 。

12.3 早停

只进行 M (≥ 0) 步的梯度优化, 即使优化过程还没有收敛。

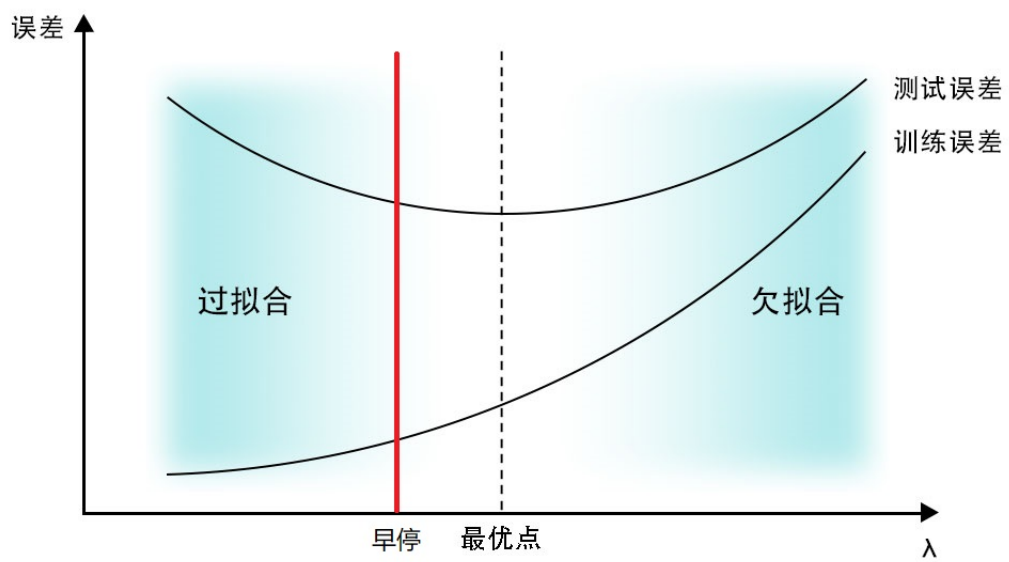


图 12-2 早停

13

方差偏差权衡

了解不同的误差来源如何导致偏差和方差，有助于我们改进数据拟合过程，从而得到更准确的模型。我们用三种方式定义偏差和方差：概念上的、图形上的和数学上的。

13.1 概念定义

13.1.1 偏差

偏差度量了学习算法的期望预测与真实结果的偏离程度，即刻画了学习算法本身的拟合能力。

13.1.2 方差

方差度量了同样大小的训练集的变动所导致的学习性能的变化，即刻画了数据扰动所造成的影响。

13.2 图形表示

13.2.1 偏差方差可视化

由下图13-1所示。我们对方差与偏差做了图形可视化。假设红色的靶心区域是学习算法完美的正确预测值，蓝色点为训练数据集所训练出的模型对样本的预测值，当我们从靶心逐渐往外移动时，预测效果逐渐变差。想象一下，我们可以每次重复模型构建过

程，以获得图中上的许多单独的命中。从上面的图片中很容易可以看到，左边一列的蓝色点比较集中，右边一列的蓝色点比较分散，它们描述的是方差的两种情况。比较集中的属于方差比较小，比较分散的属于方差比较大的情况。我们再从蓝色点与红色靶心区域的位置关系来看，靠近红色靶心的属于偏差较小的情况，远离靶心的属于偏差较大的情况。

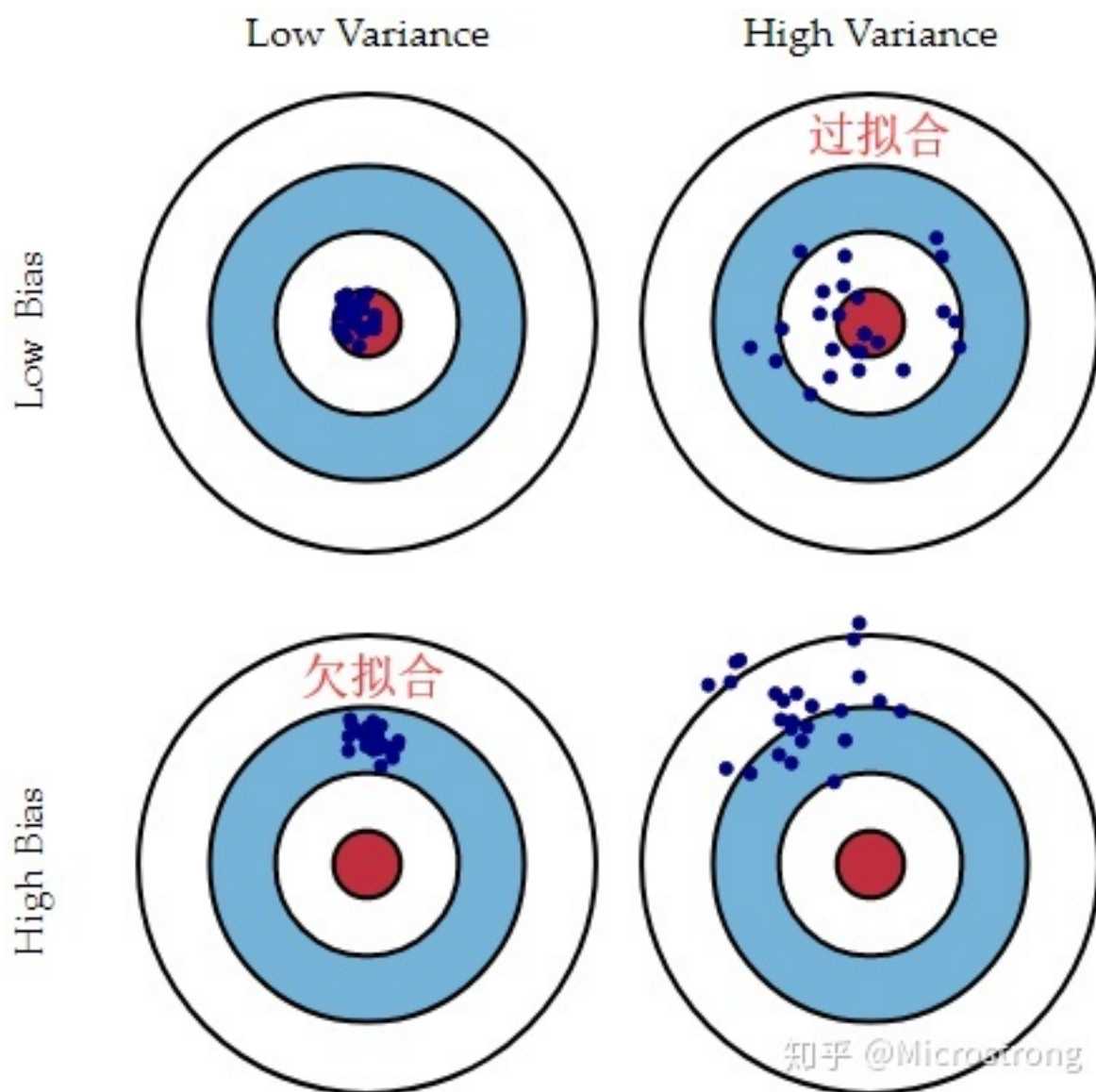


图 13-1 偏差与方差

13.2.2 思考

从上图13-1中可以看出，模型不稳定时会出现偏差小、方差大的情况，那么偏差和方差作为两种度量方式有什么区别呢？

13.2.3 解答

Bias 的对象是单个模型，是期望输出与真实标记的差别。它描述了模型对本训练集的拟合程度。Variance 的对象是多个模型，是相同分布的不同数据集训练出模型的输出值之间的差异。它刻画的是数据扰动对模型的影响。

13.3 数学表示

如果我们把需要预测的变量表示成 Y ，而自变量为 X ，假设 Y 与 X 的映射关系可以表达为 $Y = f(X) + \epsilon$ ，其中 ϵ 表示均值为 0 的高斯分布的标准差 ϵ 。假设对于真实预测 $f(x)$ 我们模型训练出的预测为 $\hat{f}(x)$ ，在这种情况下，我们可以得到模型的均方误差为

$$Err(x) = E[(Y - \hat{f}(x))^2]$$

上述误差公式可以分解成偏差与方差：

$$Err(x) = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma_\epsilon^2$$

显然可以看出，上述公式也可以对应表示成

$$Err(x) = Bias^2 + Variance + Irreducible Error$$

其中第三项为噪声，这在任何机器学习模型都不可去除。然而对于方差与偏差，假如我们可以有完美的模型，并且有无限多的数据去调整模型，我们有可能将偏差与方差变为 0。但实际上，我们无法获得完美的模型，同样在现实生活中我们也不可能无限的数数据，因此在训练过程中，对于方差与偏差的权衡就显得尤为重要。

13.4 偏差方差困境

一般来说，偏差与方差是有冲突的，这称为偏差-方差困境 (bias-variance dilemma)。下图给出了一个示意图。给定学习任务，假定我们能控制学习算法的训练程度，则在训练不足时，学习器的拟合能力不够强，训练数据的扰动不足以使学习器产生显著变化，

此时偏差主导了泛化错误率；随着训练程度的加深，学习器的拟合能力逐渐增强，训练数据发生的扰动渐渐能被学习器学到，方差逐渐主导了泛化错误率；在训练程度充足后，学习器的拟合能力已经非常强，训练数据发生的轻微扰动都会导致学习器发生显著变化，若训练数据自身的、非全局的特性被学习器学到了，则将发生过拟合。

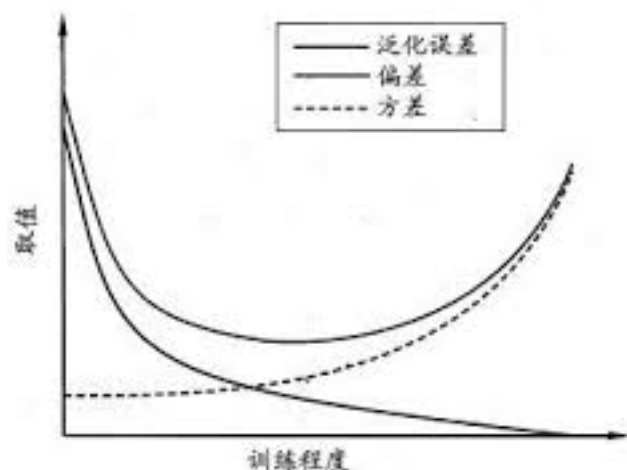


图 13-2 偏差方差权衡

13.5 偏差、方差与过拟合、欠拟合的关系

一般来说，简单的模型会有一个较大的偏差和较小的方差，复杂的模型偏差较小方差较大。

欠拟合：模型不能适配训练样本，有一个很大的偏差。

在模型表达能力不足时，在此情况下，我们向模型提供多少数据不重要，因为模型根本无法表示数据的基本关系，模型不能适配训练样本，有一个很大的偏差，因此我们需要更复杂的模型。

过拟合：模型很好的适配训练样本，但在测试集上表现很糟，有一个很大的方差

方差就是指模型过于拟合训练数据，以至于没办法把模型的结果泛化。而泛化正是机器学习要解决的问题，如果一个模型只能对一组特定的数据有效，换了数据就无效，我们就说这个模型过拟合。这就是模型很好的适配训练样本，但在测试集上表现很糟，有一个很大的方差。

13.6 偏差、方差与模型复杂度的关系

了解了偏差和方差，我们总结一下偏差和方差的来源：我们训练的机器学习模型，必不可少地对数据有依赖性。但是，如果不清楚数据服从一个什么样的分布，或是没办法拿到所有数据（肯定拿不到所有数据），那么我们训练出来的模型和真实模型之间就存在不一致性。这种不一致主要来源两个方面：偏差和方差。

既然偏差和方差无法避免的，那么我们有什么办法尽量减少它对模型的影响呢？

一个好的办法就是正确选择模型的复杂度。复杂度高的模型通常对训练数据有很好的拟合能力，但是对测试数据就不一定了。而复杂度太低的模型又不能很好的拟合训练数据，更不能很好的拟合测试数据。因此，模型复杂度和模型偏差和方差具有如下图13-3所示关系。

从上图13-3可以看出，我们训练模型的时候要慎重的选择模型的复杂度，使得模型的泛化能力与表达能力都较好。

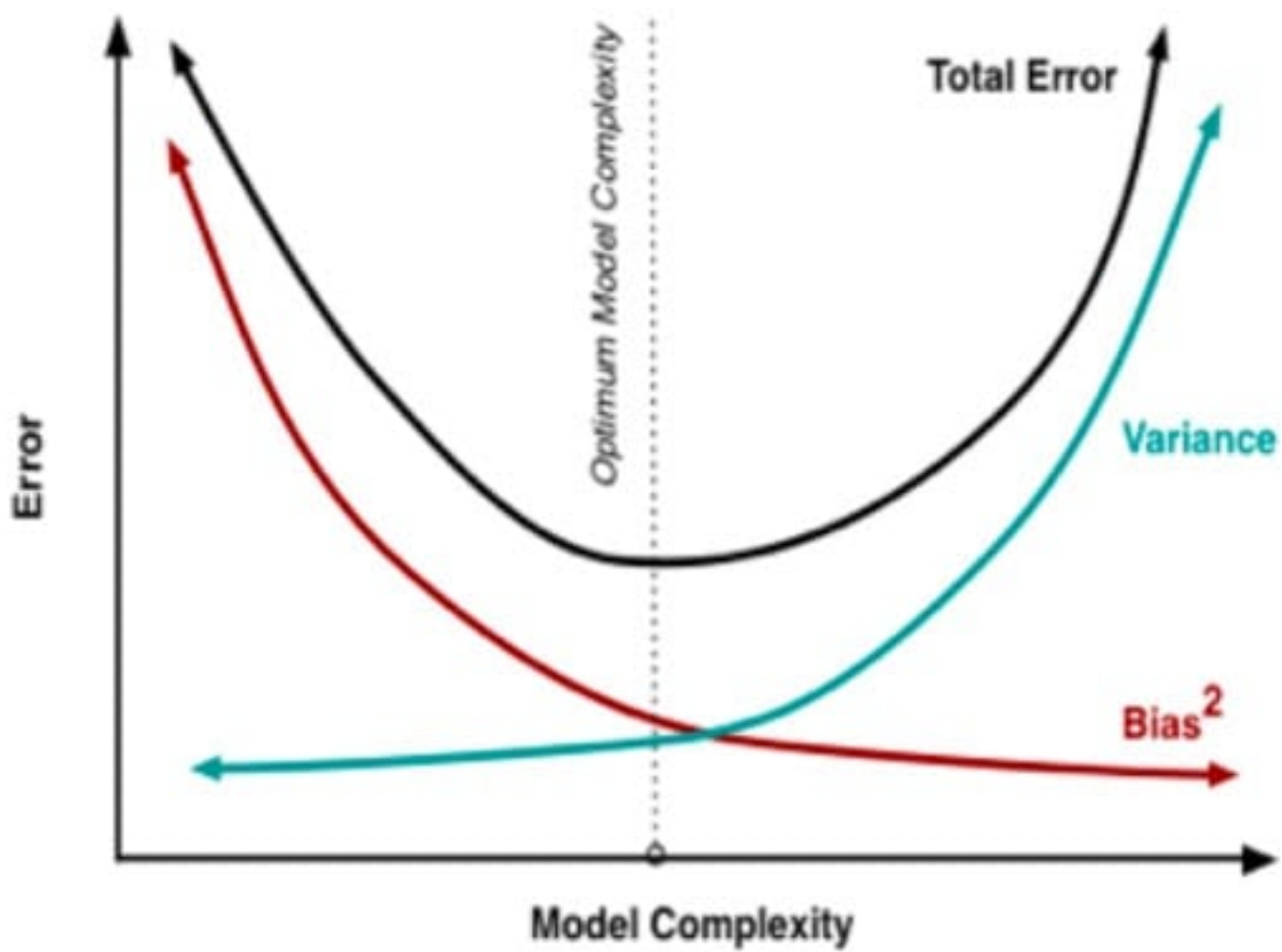


图 13-3 偏差方差与复杂度的关系

14

核方法

线性分类器很好，但是如果不存在线性决策边界呢？事实证明，有一种很好的方法可以将非线性合并到大多数线性分类器中。

14.1 Handcrafted Feature Expansion

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。有数据向量 $x \in \mathbb{R}^d$ ，我们应用变换 $x \rightarrow \phi(x)$, $\phi(\mathbf{x}) \in \mathbb{R}^D$ 。通常因为我们添加维度捕捉非线性特性之间的相关性所以 $D \gg d$ 。

优点：它是简单的，问题仍然保持凸。(即你仍然可以使用普通的梯度下降代码)

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

提问： $\phi(\mathbf{x})$ 的维度是多少？这个新表示 $\phi(\mathbf{x})$ ，非常富有表现力，使得复杂的非线性边界可以存在。但维度极高，这使得我们的算法变得难以忍受。

14.2 The Kernel Trick

14.2.1 Gradient Descent with Squared Loss

现在，注意当我们用很多损失函数做梯度下降时，梯度是输入样本的线性组合。以平方损失为例：

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

步长为 $s \geq 0$ 的梯度下降规则随时间更新 \mathbf{w}

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

现在我们要证明我们可以把 \mathbf{x} 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，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，我们选择 $\mathbf{w}_0 = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$ 。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 $\alpha_1 = \dots = \alpha_n = 0$ 。我们现在证明在整个梯度下降过程中的系数 $\alpha_1, \dots, \alpha_n$ 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：

$$\alpha_i^t = \alpha_i^{t-1} - s \gamma_i^{t-1}, \text{ and we have } \alpha_i^t = -s \sum_{r=0}^{t-1} \gamma_i^r$$

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规则。我们跟踪的 n 系数 $\alpha_1, \dots, \alpha_n$ 。既然 \mathbf{w} 可以写成训练集的线性组合，我们也可以用训练输入之间的内积来表示 \mathbf{w} 与任意输入 \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 $\ell(\mathbf{w}) = \sum_{i=1}^n (\mathbf{w}^\top \mathbf{x}_i - y_i)^2$ 重写为输入的内积。

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

在测试时，我们也只需要这些系数对一个测试输入 \mathbf{x}_t 进行预测，就可以用测试点和训练点之间的内积来表示整个分类器：

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

你回忆起到主题了吗？为了学习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对数据向量之间的内积。

14.3 Inner-Product Computation

回到刚开始的例子 $\phi(\mathbf{x}) = \begin{pmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1 x_2 \\ \vdots \\ x_{d-1} x_d \\ \vdots \\ x_1 x_2 \cdots x_d \end{pmatrix}$ ，内积 $\phi(\mathbf{x})^\top \phi(\mathbf{z})$ 可以被形式化为：

$$\phi(\mathbf{x})^\top \phi(\mathbf{z}) = 1 \cdot 1 + x_1 z_1 + x_2 z_2 + \cdots + x_1 x_2 z_1 z_2 + \cdots + x_1 \cdots x_d z_1 \cdots z_d = \prod_{k=1}^d (1 + x_k z_k)$$

2^d 项的和变成了 d 项的乘积。我们可以从上面的公式中计算出内积的时间是 $O(d)$ 而不是 $O(2^d)!$ 事实上，我们可以预先计算它们并将它们存储在一个核矩阵中。

$$\underbrace{K(\mathbf{x}_i, \mathbf{x}_j)}_{\text{this is called the Kernel Matrix}} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

如果我们存储矩阵 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低维度量级计算。

14.4 General Kernels

以下是一些常见的内核函数：

Linear: $K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$ (线性核函数等价于使用一个好的旧的线性分类器——但是如果数据的维数 d 很高，使用核函数矩阵会更快。)

Polynomial: $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x}^\top \mathbf{z})^d$

Radial Basis Function (RBF) (aka Gaussian Kernel): $K(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|^2}{\sigma^2}}$ RBF 内核是最流行的内核！它是一个通用逼近器！！它对应的特征向量是无限维的。

Exponential Kernel: $K(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|}{\sigma^2}}$

Laplacian Kernel: $K(\mathbf{x}, \mathbf{z}) = e^{-\frac{\|\mathbf{x}-\mathbf{z}\|}{\sigma}}$

sigmoid Kernel: $K(\mathbf{x}, \mathbf{z}) = \tanh(\mathbf{a}^\top \mathbf{x} + c)$

思考： $K(\cdot, \cdot)$ 可以是任意矩阵吗？不，这个矩阵 $K(x_i, x_j)$ 对应于真实的内积变换后 $\mathbf{x} \rightarrow \phi(\mathbf{x})$ 。这是当且仅当 K 是正半定的情况。

定义：矩阵 $A \in \mathbb{R}^{n \times n}$ 为半正定当且仅当 $\forall \mathbf{q} \in \mathbb{R}^n, \mathbf{q}^\top A \mathbf{q} \geq 0$

为什么这样？记住 $K(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z})$. A matrix of form $A = \begin{pmatrix} \mathbf{x}_1^\top \mathbf{x}_1, \dots, \mathbf{x}_1^\top \mathbf{x}_n \\ \vdots \\ \mathbf{x}_n^\top \mathbf{x}_1, \dots, \mathbf{x}_n^\top \mathbf{x}_n \end{pmatrix} =$

$$\begin{pmatrix} \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} (\mathbf{x}_1, \dots, \mathbf{x}_n) \text{ 必须为半正定，因为: } \mathbf{q}^\top A \mathbf{q} = \left(\begin{pmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_n \end{pmatrix} \mathbf{q} \right)^\top \left(\begin{pmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_n \end{pmatrix} \mathbf{q} \right) \geq 0$$

对于 $\forall \mathbf{q} \in \mathbb{R}^n$ 您甚至可以在集合、字符串、图形和分子上定义内核。

15

核方法（续）

15.1 Well-defined kernels

下面是一些常见的核：

- ▶ 线性核： $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$
- ▶ RBF 核： $k(\mathbf{x}, \mathbf{z}) = e^{-\frac{(\mathbf{x}-\mathbf{z})^2}{\sigma^2}}$
- ▶ 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核结合而成的核被称为 *well-defined kernels*：

1. $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$
2. $k(\mathbf{x}, \mathbf{z}) = ck_1(\mathbf{x}, \mathbf{z})$
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. $k(\mathbf{x}, \mathbf{z}) = e^{k_1(\mathbf{x}, \mathbf{z})}$
8. $k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{A} \mathbf{z}$

其中 k_1, k_2 是 well-defined kernels, $c \geq 0$, g 是一个正系数的多项式函数, f 可以是任意函数, $\mathbf{A} \succeq 0$ 是一个半正定矩阵。一个核是 well-defined 等价于它的核矩阵 K 是半正定的（此处不证明），或者说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. 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 well-defined kernels。

定理 2 RBF 核是一个 well-defined 核

证明.

$k_1(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$ 是 well-defined(由上面第一个 well-defined 核可知),

$k_2(\mathbf{x}, \mathbf{z}) = \frac{2}{\sigma^2} k_1(\mathbf{x}, \mathbf{z}) = \frac{2}{\sigma^2} \mathbf{x}^\top \mathbf{z}$ 是 well-defined(由上面第二个 well-defined 核可知),

$k_3(\mathbf{x}, \mathbf{z}) = e^{k_2(\mathbf{x}, \mathbf{z})} = e^{\frac{2\mathbf{x}^\top \mathbf{z}}{\sigma^2}}$ 是 well-defined(由上面第七个 well-defined 核可知),

$k_4(\mathbf{x}, \mathbf{z}) = e^{-\frac{\mathbf{x}^\top \mathbf{x}}{\sigma^2}} k_3(\mathbf{x}, \mathbf{z}) e^{-\frac{\mathbf{z}^\top \mathbf{z}}{\sigma^2}} = e^{-\frac{\mathbf{x}^\top \mathbf{x}}{\sigma^2}} e^{\frac{2\mathbf{x}^\top \mathbf{z}}{\sigma^2}} e^{-\frac{\mathbf{z}^\top \mathbf{z}}{\sigma^2}} = e^{\frac{-\mathbf{x}^\top \mathbf{x} + 2\mathbf{x}^\top \mathbf{z} - \mathbf{z}^\top \mathbf{z}}{\sigma^2}} = e^{\frac{-(\mathbf{x}-\mathbf{z})^\top (\mathbf{x}-\mathbf{z})}{\sigma^2}} = k_{RBF}(\mathbf{x}, \mathbf{z})$ 是 well-defined(由上面第六个 well-defined 核可知)。

□

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 well-defined,

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

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 well-defined。

□

15.2 Kernel Machines

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

15.2.1 核线性回归

Recap

线性回归的形式是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 $\mathbf{X} = [\mathbf{x}_1, \dots, \mathbf{x}_n]$, $\mathbf{y} = [y_1, \dots, y_n]^\top$, 那么上述问题的有闭式解:

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

Kernelization

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 α 的闭式解。

定理 4 核回归的解为 $\vec{\alpha} = \mathbf{K}^{-1}\mathbf{y}$

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 $\mathbf{X}^\top \mathbf{X}$ 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。通常我们也将核岭回归看作是核回归。

Testing

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

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

或者说，其他某些算法也有这样的闭式解：

$$h(\mathbf{z}) = \mathbf{k}_* (\mathbf{K} + \tau^2 \mathbf{I})^{-1} \mathbf{y},$$

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。

15.2.2 Nearest Neighbors

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)?

15.3 Kernel SVM

SVM 的原始形式是一个二次规划问题：

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

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 $\mathbf{w} = \sum_{i=1}^n \alpha_i y_i \phi(\mathbf{x}_i)$,

$$h(\mathbf{x}) = \text{sign} \left(\sum_{i=1}^n \alpha_i y_i k(\mathbf{x}_i, \mathbf{x}) + b \right). \quad (15.1)$$

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：

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

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 $\alpha_i = 0$)。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。

Recovering 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

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

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)。

15.3.1 Kernel SVM - the smart nearest neighbor

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。

16

高斯过程

我们先来回顾高斯分布的定义和性质：一个高斯随机变量 $X \sim \mathcal{N}(\mu, \Sigma)$ ，其中 μ 是均值， Σ 是协方差矩阵，其概率密度函数如下： $P(x; \mu, \Sigma) = \frac{1}{(2\pi)^{\frac{d}{2}} |\Sigma|} e^{-\frac{1}{2}((x-\mu)^\top \Sigma^{-1}(x-\mu))}$ 其中 $|\Sigma|$ 是 Σ 的行列式高斯分布在实际数据中经常出现，主要因为中心极限定理（CLT, the central limit theorem）。中心极限定理表明，样本的平均值 $m > 0$ 的情况下，样本近似正态分布，独立于原始状态分布，假设它具有有限的均值和方差。

中心极限定理的说明：在上图中，随机变量 Y 是由柱状图所示的分布绘制的。1,2 和 9,10 的值是有可能的，但是中间的 3-8 是取不到的。这个分布看起来根本就不是高斯分布，然而他们的样本均值为 $\bar{Y}_j = \frac{1}{N} \sum_{i=1}^N y_i$ ，其中我们抽取 m 个样本： $\bar{Y}_1, \dots, \bar{Y}_m$ ，随着 N 增加，其变得更加“高斯”

16.1 一旦高斯，一直高斯

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：

16.1.1 1. 正规化

$$\int_y p(y; \mu, \Sigma) dy = 1 \quad (\text{of course!})$$

16.1.2 2. 边界化

边际分布 $p(y_A) = \int_{y_B} p(y_A, y_B; \mu, \Sigma) dy_B$ 以及 $p(y_B) = \int_{y_A} p(y_A, y_B; \mu, \Sigma) dy_A$, 两者都是高斯分布: $y_A \sim \mathcal{N}(\mu_A, \Sigma_{AA})$ $y_B \sim \mathcal{N}(\mu_B, \Sigma_{BB})$

16.1.3 3. 和性质

如果 $y \sim \mathcal{N}(\mu, \Sigma)$ 以及 $y' \sim \mathcal{N}(\mu', \Sigma')$ 那么: $y + y' \sim \mathcal{N}(\mu + \mu', \Sigma + \Sigma')$.

16.1.4 4. 条件分布

y_A 在 y_B 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 $y_A|y_B = y_B \sim \mathcal{N}(\mu_A + \Sigma_{AB}\Sigma_{BB}^{-1}(y_B - \mu_B), \Sigma_{AA} - \Sigma_{AB}\Sigma_{BB}^{-1}\Sigma_{BA})$. 这一性质对高斯过程假设的推导是有用的。

16.2 高斯过程回归

16.2.1 后验预测分布

考虑一个回归问题:

$$y = f(\mathbf{x}) + \epsilon \quad (16.1)$$

$$y = \mathbf{w}^T \mathbf{x} + \epsilon \quad (\text{OLS and ridge regression}) \quad (16.2)$$

$$y = \mathbf{w}^T \phi(\mathbf{x}) + \epsilon \quad (\text{kernel ridge regression}). \quad (16.3)$$

$$(16.4)$$

回想一下我们从数据中估计概率的目标。到目前为止, 我们已经开发了 OLS 和 (内核) 岭回归作为回归问题的解决方案。这些解给出了一个特定参数 \mathbf{w} 的预测模型。一般来说, 后验预测分布为: $P(Y | D, X) = \int_{\mathbf{w}} P(Y, \mathbf{w} | D, X) d\mathbf{w} = \int_{\mathbf{w}} P(Y | \mathbf{w}, D, X) P(\mathbf{w} | D) d\mathbf{w}$ 不幸的是, 在封闭的形式下, 上述问题往往难以解决。然而, 对于具有高斯似然和先验的特殊情况 (即岭回归假设), 这个表达式是高斯的, 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, $\mu_{y_*|D} = K_*^T (K + \sigma^2 I)^{-1} y$ 以及 $\Sigma_{y_*|D} = K_{**} - K_*^T (K + \sigma^2 I)^{-1} K_*$ 因此, 我们不像在岭回归中那样做 MAP, 而是对整个分布建模, 让我们忘记 \mathbf{w} 和内核技巧, 直接对 f 建模 (而不是 y)!

16.3 高斯过程——定义

问题描述: f 是一个无限维度的函数! 但是, 多元高斯分布是针对有限维随机向量的。

定义: GP 是随机变量 (RV) 的集合 (可能无穷大), 因此 RV 的每个有限子集的联合分布都是多元高斯分布: $f \sim GP(\mu, k)$ 其中 $\mu(\mathbf{x})k(\mathbf{x}, \mathbf{x}')$ 是均值的协方差函数! 现在现在, 为了模型预测分布 $P(f_* | \mathbf{x}_*, D)$ 我们可以用贝叶斯方法之前通过 GP: $P(f | \mathbf{x}) \sim \mathcal{N}(\mu, \Sigma)$ 和条件在训练数据 D 模型 $f = f(\mathbf{X})$ 的联合分布 (向量训练观察) 和 $f_* = f(\mathbf{x}_*)$ 在测试输入 (预测)。

16.4 高斯过程回归 (GPR)

我们假设, 在我们观察训练标签之前, 这些标签是由零均值先验高斯分布得出的:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \\ y_t \end{bmatrix} \sim \mathcal{N}(0, \Sigma) \text{ 通过减去样本均值, 零均值总是可能的。所有的训练和测试标签都是从}$$

$(n+m)$ 维高斯分布中抽取的, 其中 n 为训练点数, m 为测试点数。注意, 真正的训练标签是 y_1, \dots, y_n , 我们观察到的是 Y_1, \dots, Y_n 的样本。这个分布是否给出了有意义的分布取决于我们如何选择协方差矩阵 Σ 。我们认为 Σ 有以下属性: 1. $\Sigma_{ij} = E((Y_i - \mu_i)(Y_j - \mu_j))$ 2. Σ 总是半正定的 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 完全与 \mathbf{x}_j 不同, 那么 $\Sigma_{ij} = \Sigma_{ji} = 0$ 5. 如果 \mathbf{x}_i 相似于 \mathbf{x}_j , 那么 $\Sigma_{ij} = \Sigma_{ji} > 0$

从支持向量机的核矩阵可以看出, 这是非常相似的。因此, 我们可以简单地让 $\Sigma_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ 。例如, 如果我们使用 RBF 核 (即 “平方指数核”), 则:

$$\Sigma_{ij} = \tau e^{\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma^2}}. \quad (16.5)$$

如果我们用多项式核, 那么 $\Sigma_{ij} = \tau(1 + \mathbf{x}_i^\top \mathbf{x}_j)^d$ 因此我们可以将 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。(无噪声) 条件分布的隐函数 f 的值可以被写为:

$$f_* | (Y_1 = y_1, \dots, Y_n = y_n, \mathbf{x}_1, \dots, \mathbf{x}_n, \mathbf{x}_t) \sim \mathcal{N}(K_*^\top K^{-1} y, K_{**} - K_*^\top K^{-1} K_*), \quad (16.6)$$

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

16.4.1 加性高斯噪声

在很多应用中，观察到的标签可能是有噪声的。如果我们假设这个为独立零均值的高斯噪声，那么我们会发现 $\hat{Y}_i = f_i + \epsilon_i$ ，其中 f_i 是真实目标（未观察到的为潜在目标），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 $\hat{\Sigma} = \Sigma + \sigma^2 \mathbf{I}$ 我们可以针对 $i \neq j$ 的非对角项推导出：

$$\hat{\Sigma}_{ij} = \mathbb{E}[(f_i + \epsilon_i)(f_j + \epsilon_j)] = \mathbb{E}[f_i f_j] + \mathbb{E}[f_i] \mathbb{E}[\epsilon_j] + \mathbb{E}[f_j] \mathbb{E}[\epsilon_i] + \mathbb{E}[\epsilon_i] \mathbb{E}[\epsilon_j] = \mathbb{E}[f_i f_j] = \Sigma_{ij}, \quad (16.7)$$

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。对于 Σ 的对角线项，比如 $i = j$ 的情况下，我们可以得到：

$$\hat{\Sigma}_{ii} = \mathbb{E}[(f_i + \epsilon_i)^2] = \mathbb{E}[f_i^2] + 2\mathbb{E}[f_i] \mathbb{E}[\epsilon_i] + \mathbb{E}[\epsilon_i^2] = \mathbb{E}[f_i f_i] + \mathbb{E}[\epsilon_i^2] = \Sigma_{ii} + \sigma^2, \quad (16.8)$$

由于 $\mathbb{E}[\epsilon_i^2] = \sigma^2$ ，其中 ϵ_i 表示方差。将更新后的协方差矩阵代入高斯过程后验分布，得到：

$$Y_* | (Y_1 = y_1, \dots, Y_n = y_n, \mathbf{x}_1, \dots, \mathbf{x}_n) \sim \mathcal{N}(K_*^\top (K + \sigma^2 I)^{-1} y, K_{**} + \sigma^2 I - K_*^\top (K + \sigma^2 I)^{-1} K_*). \quad (16.9)$$

在实际中往往上述方程更稳定，因为矩阵 $(K + \sigma^2 I)$ 总是可逆的，只要 σ^2 足够大。因此，对于预测，我们可以使用后验均值，此外，我们得到预测方差，作为对点预测的可信度或(不)确定性的衡量。

16.5 实践运用

16.5.1 边际似然和超参数学习

虽然 GPR 是非参数模型，但协方差函数 K 有参数。这些参数就是所谓的超参数，它们需要从训练数据中学习 (= 估计)。 θ 是 K 的所有参数的向量， $K = K_\theta$ ，我们可以通过最大化边际似然概率 $P(y | X, \theta)$ 一旦你有了边际似然及其导数，你就可以使用任何封装好的解决方案，如 (随机) 梯度下降，或共轭梯度下降 (注意: 最小化负对数边际似然)。注意，边缘似然在其参数中不是一个凸函数，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健壮，您可以使用不同的初始化重新运行优化算法，并选择最低/最高的返回值。

16.5.2 协方差函数- GP 模型的核心

通过选择正确的协方差/核函数，GPs 获得了很大的预测能力。选择协方差函数是 GP 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. 专业知识

(拥有起来很棒，但是很难获得) 2. 贝叶斯模型选择 (更可能是分析上难以处理的积分!!)
 3. 交叉验证 (耗时——但易于实现) 一个经常使用并且的协方差函数是 RBF，它对每个特征维具有不同的长度尺度。 $k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 e^{-\frac{1}{2}(\mathbf{x}-\mathbf{x}')^T M(\mathbf{x}-\mathbf{x}')} + \sigma_n^2 \delta_{\mathbf{x}, \mathbf{x}'}$, 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。

16.6 总结

高斯过程回归具有以下性质: 1.GPs 是一种简洁有效的 ML 方法 2. 我们没有代价的获得了预测的 (不) 确定性。3.GPs 对于训练数据集较小的回归问题非常有效。4. 运行时间 $O(n^3) \leftarrow$ 矩阵求逆 ($n \gg 0$ 时求解缓慢) \Rightarrow 对于大数字 n 使用稀疏 GPs。5.GPs 在分类 (非高斯似然) 方面涉及更多。6. 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。7. GP 实现: GPyTorch、GPML (MATLAB)、GPyS、pyGPs 和 scikit-learn (Python)

16.7 贝叶斯全局优化

GP 回归的一个很好的应用是贝叶斯全局优化。这里的目标是优化机器学习算法的超参数, 使其在固定的验证数据集上运行良好。假设有 d 个超参数要调优, 那么数据集由 d 维向量 $\mathbf{x}_i \in \mathcal{R}^d$ 组成, 其中每个训练点表示一个特定的超参数设置, 标签 $y_i \in \mathcal{R}$ 表示验证误差。不要混淆, 这次向量 \mathbf{x}_i 对应的是超参数设置, 而不是数据。例如, 在一个具有多项式内核的支持向量机中, 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 。首先, 在一些随机的超参数设置下训练分类器, 并在验证集上评估分类器。 $\mathbf{x}_1, \dots, \mathbf{x}_m$ 带有标签 y_1, \dots, y_m 。现在, 可以训练一个高斯过程来预测任何新的超参数设置 \mathbf{x}_t 下的验证错误 y_t 。实际上, 你得到一个平均预测 $y_t = h(\mathbf{x}_t)$ 及其方差 $v(\mathbf{x}_t)$ 。如果你有更多的时间, 你现在可以探索一个新的点。最保证的点是置信下限最低的点, 即

$$\operatorname{argmin}_{\mathbf{x}_t} h(\mathbf{x}_t) - \kappa \sqrt{v(\mathbf{x}_t)}. \quad (16.10)$$

常数 $\kappa > 0$ 决定了你想要探索可能是好点的程度, 因为你很不确定 (方差高), 或者说决定了说你有多想要利用你的知识来改进当前最佳点, 直到目前为止最好的设置为止。一个很小的 κ 会导致更多的利用, 而一个更大的 κ 会更加激进的探索新的超参数设置。算法: $\mathcal{A}, m, n, \kappa$ For $i = 1$ to m \mathbf{x}_i 随机取样例如: 合理范围内的均匀采样 $y_i = \mathcal{A}(\mathbf{x}_i)$ 计算验证误差 End for For $i = m + 1$ to n 基于 $\mathbf{x}_1, \dots, \mathbf{x}_{i-1}$ 更新核 K $\mathbf{x}_i = \operatorname{argmin}_{\mathbf{x}_t} K_t^\top (K + \sigma^2 I)^{-1} y - \kappa \sqrt{K_{tt} + \sigma^2 I - K_t^\top (K + \sigma^2 I)^{-1} K_t} y_i = \mathcal{A}(\mathbf{x}_i)$ 计算验证误差 End for $i_{best} = \operatorname{argmin}_i \{y_1, \dots, y_n\}$ 找到探索过的最好的超参数设置返回 $\mathbf{x}_{i_{best}}$ 返回探索过的最好的超参数设置

图中显示了在四个训练点 (黑色十字) 上训练的高斯过程, 并在 $[-5,5]$ 区间内在密集网格上进行评估。红线表示每个测试点的预测值。灰色阴影区域表示预测的不确定性 (离均值两个标准差)。测试点越接近训练点 (即具有更多相似的特性), 得到的预测就越确定。BO 比网格搜索快得多。

17

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 $x_s \in \mathbb{R}^d$ 。测试过程中, 对于每个测试数据 x_t , 都需要计算测试数据与训练集中所有数据点的距离 $dist(x_t, x_s), \forall s \in N_1^n$, 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。

17.1 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 17-1 所示, 二维的特征空间被划分为 12 个子空间。

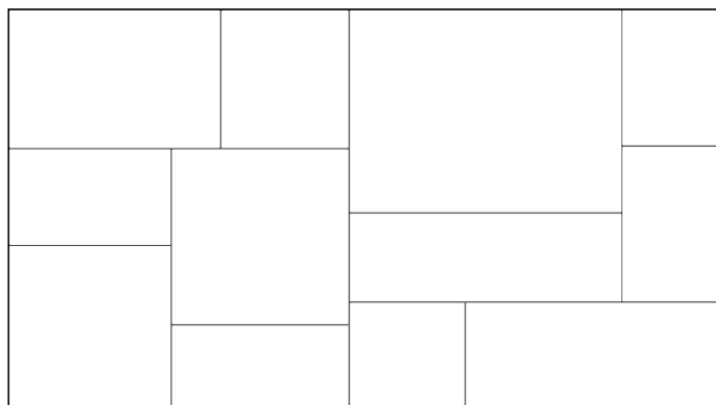


图 17-1 被 KD 树分割后的特征空间

17.1.1 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 17-2 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。

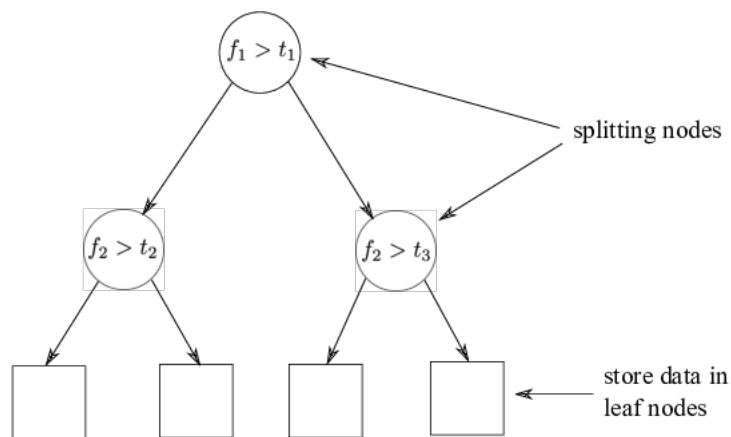


图 17-2 KD 树结构示意图

17.1.2 近邻搜索

根据几何关系，对 kd 树分割的子空间有定理 17.1.2

定理 17.1

某一子空间 S 内所有数据点 x ，与子空间外一数据点 x_t 的距离 d_s ，均小于等于子空间边界 w 与该数据点的距离 d_w 。即 $\forall x \in S, d_s \leq d_w$



证明. 如图 17-3 左半部分所示，子空间内任一点 x 与子空间外一点 x_t 的距离，可分为两部分 $d_s = d_1 + d_2$ ，子空间边界与 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 17.1.2，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 x_t

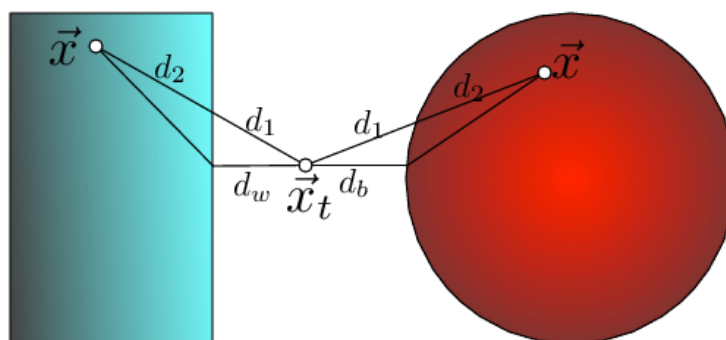


图 17-3 特征子空间的距离示意图

- 1 从 KD 树的根节点出发，找到 x_t 所属叶子节点。
- 2 叶子节点对应子空间 S ，计算 x_t 与子空间 S 中所有数据点的距离，并保存最近邻节点 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 w ，若存在子树还未被访问，计算 w 与 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 x_t ，若存在距离小于 d_{min} ，则更新 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 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 $O(n)$ 。

17.2 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 17-3。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。

17.2.1 创建球树

输入：集合 S , $n = |S|$, k

算法：见算法1

Algorithm 1 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  $x_0 \in S$ 
4:   选取  $x_1 = \arg \max_{x \in S} d(x_0, x)$ 
5:   选取  $x_2 = \arg \max_{x \in S} d(x_1, x)$ 
6:    $\forall i = 1 \dots |S|, z_i = (x_1 - x_2)^T x_i \leftarrow$  将数据投影到  $(x_1 - x_2)$ 
7:    $m = \text{median}(z_1, \dots, z_{|S|})$ 
8:    $S_L = \{x \in S : z_i < m\}$ 
9:    $S_R = \{x \in S : z_i \geq m\}$ 
10:  Return 树:
    - 圆心:  $c = \text{mean}(S)$ 
    - 半径:  $r = \max_{x \in S} d(x, c)$ 
    - 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 $(x_1 - x_2)$ 。

17.2.2 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，然而在高维时远快与 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。

17.3 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。

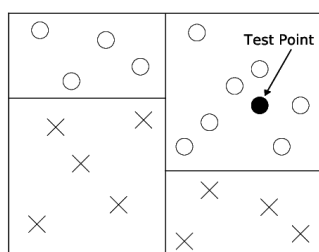
18

决策树

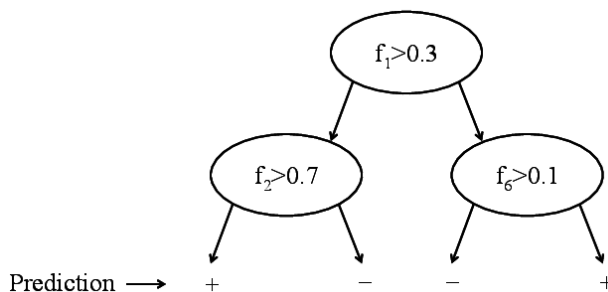
决策树是根据训练数据生成的树状结构，将特征空间划分为若干个内部输出一致的特征子空间，如图18-1a所示，特征空间被划分成了4个内部输出一致的特征子空间，决策树本质上是从训练数据中归纳出来的一组分类规则。

决策树由若干节点和有向边组成，节点分为两种：叶子节点和非叶子节点，叶子节点存储输出 c ，非叶子节点存储特征 k 以及阈值 t ，当前非叶子节点所代表的特征空间，按特征 k 的取值划分为两个子空间，结点左右子树代表在特征 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18-1b所示，非叶子节点中存储特征和阈值，叶子节点存储输出标签。

使用决策树对实例 $x = [x^{(1)}, \dots, x^{(n)}]$ 进行预测，就是从根节点 (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节点 c ，则决策树对于该实例的预测结果就是 c 。



(a) 特征空间



(b) 决策树

图 18-1 示意图

18.1 基本概念

假设数据集为 $S = \{(x_1, y_1), \dots, (x_n, y_n)\}$, $y_i \in \{1, 2, \dots, c\}$, 共 n 个样本, 属于 c 个类别。

18.1.1 信息熵

设有随机变量 X , 概率分布为 $P(X = x_i) = p_i, i = 1, 2, \dots, n$, 那么随机变量 X 的熵为

$$H(X) = - \sum_{i=1}^n p_i \log p_i \quad (18.1)$$

记随机变量 X 的概率分布为 p , 从熵的定义式子 (18.1) 可知, 熵只与 X 的概率分布有关, 那么 X 的熵也可以记作 $H(p)$ 。

设有随机变量 (X, Y) , 其联合概率分布为 $P(X = x_i, Y = y_j) = p_{ij}, i = 1, 2, \dots, n; j = 1, 2, \dots, m$, 在随机变量 X 给定的情况下, 随机变量 Y 的条件熵为

$$H(Y|X) = \sum_{i=1}^n P(X = x_i) H(Y|X = x_i) \quad (18.2)$$

一般将 $H(Y)$ 与 $H(Y|X)$ 的差称为互信息 (也有称作信息增益)

$$g(Y, X) = H(Y) - H(Y|X) \quad (18.3)$$

$g(Y, X)$ 表示已知 X 相比于未知 X , 随机变量 Y 不确定性的下降程度, 也就是 X 相对 Y 的信息量。在决策树中, 训练集 S 按特征 f 进行分割的信息增益记为 $g(S, f)$ 。

特征 f 对于数据集 S 的信息增益比 $g_R(S, f)$, 为其信息增益熵 $g(S, f)$ 与数据集 S 关于特征 f 值的熵 $H_f(S)$ 的比, 如式子 (18.4) 所示

$$g_R(S, f) = \frac{g(S, f)}{H_f(S)} \quad (18.4)$$

其中 $H_f(S) = - \sum_{i=1}^{n_f} \frac{|S_i|}{|S|} \log \frac{|S_i|}{|S|}$, n_f 为特征 f 所有可能取值的数目

决策树的信息熵为

$$H(S) = p^L H(S^L) + p^R H(S^R) \quad (18.5)$$

其中 $p^L = \frac{|S^L|}{|S|}, p^R = \frac{|S^R|}{|S|}$

18.1.2 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

$$G(S) = \sum_{k=1}^c p_k(1 - p_k) = 1 - \sum_{k=1}^c p_k^2 \quad (18.6)$$

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

$$G(S) = \frac{|S_1|}{|S|} G(S_1) + \frac{|S_2|}{|S|} G(S_2) \quad (18.7)$$

所以决策树的基尼指数为

$$G^T(S) = \frac{|S_L|}{|S|} G^T(S_L) + \frac{|S_R|}{|S|} G^T(S_R) \quad (18.8)$$

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。

18.2 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, 在决策树各个节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。

$$ID(S) : \begin{cases} \text{if } \exists \bar{y} \text{ s.t. } \forall (x, y) \in S, y = \bar{y} \Rightarrow \text{return leaf with } \bar{y} \\ \text{if } \exists \bar{x} \text{ s.t. } \forall (x, y) \in S, x = \bar{x} \Rightarrow \text{return leaf with } \\ \text{common}(y : (x, y) \in S) \text{ or mean} \end{cases} \quad (18.9)$$

式子 (18.9) 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, 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 的叶子节点; 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, 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 的叶子节点。

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：

$$\begin{cases} S^L = \{(x, y) \in S : x_f \leq t\} \\ S^R = \{(x, y) \in S : x_f \geq t\} \end{cases} \quad (18.10)$$

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节点时，将信息增益比作为选择特征的标准。

18.3 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节点时，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，为当前节点生成两个子节点，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节点；然后，在子节点中重复上述步骤，直至节点中的样本数量小于一定数目，或节点样本集的基尼指数小于一定值，则生成叶子节点并返回；最后，算法运行结束就得到了根据训练样本生成的决策树。

19

Bagging

19.1 Bagging

Bagging 是集成学习 Bootstrap Aggregating ((Breiman 96)) 的缩写, 它也称为套袋法, 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.

19.1.1 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 $h_D \rightarrow \bar{h}$.

弱大数定理

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, i.e. $\mathbb{E}[(\hat{h}(x) - \bar{h}(x))^2] \rightarrow 0$

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 .

19.1.2 方法: Bagging(集成学习)

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) from D 。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)

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

$= p_i \leftarrow \underline{TATAAA}!!$ 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 i^{th} 个数据点。到目前为止，你仅仅有下标 i ，你仍需要用一个数据点 (x_i, y_i) 来填充它。为此，你从 P 中采样点 (x_i, y_i) from P 。显然，它与你选择那个下标并没有关系，所以我们有 $Q(X = x) = P(X = x)$ 。

19.1.3 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. 最终的分类器为 $h(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^m h_j(\mathbf{x})$ 。

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 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。

19.1.4 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 中的所有点，每次都仅有一个训练点没有被选择。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。

19.2 随机森林

随机森林是最著名和最有用的袋装算法之一！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 $D_l^A \cap D_l^B = \text{emptyset}$ 。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。这具有以下优点：每个树以及 RF 分类器变为 [\[https://en.wikipedia.org/wiki/Consistent\]](https://en.wikipedia.org/wiki/Consistent) 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。这可以进一步改善您的偏差/差异权衡。

20

Boosting

20.1 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:

$$\ell(H) = \frac{1}{n} \sum_{i=1}^n \ell(H(\mathbf{x}_i), y_i). \quad (20.1)$$

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:

$$h_{t+1} = \operatorname{argmin}_{h \in \mathbb{H}} \ell(H_t + \alpha h). \quad (20.2)$$

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 $\langle h, g \rangle = \sum_{i=1}^n h(\mathbf{x}_i)g(\mathbf{x}_i)$ 。

20.2 函数空间中的梯度下降

给定集成分类器 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：

$$\ell(H + \alpha h) \approx \ell(H) + \alpha \langle \nabla \ell(H), h \rangle. \quad (20.3)$$

 **注 20.1.** 公式20.3: $\ell(H_{T+1}) \approx \ell(H_T) + \alpha \langle \nabla \ell(H_T), h \rangle$.

泰勒估计20.3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20.3 来寻找最优的弱学习器 h ：

$$\operatorname{argmin}_{h \in \mathbb{H}} \ell(H + \alpha h) \approx \operatorname{argmin}_{h \in \mathbb{H}} \langle \nabla \ell(H), h \rangle = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n \frac{\partial \ell}{\partial [H(\mathbf{x}_i)]} h(\mathbf{x}_i) \quad (20.4)$$


参考公式20.1，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)， $\frac{\partial \ell}{\partial H}(\mathbf{x}_i) = \frac{\partial \ell}{\partial [H(\mathbf{x}_i)]}$ 。

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} ：

$$h_{t+1} = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n \underbrace{\frac{\partial \ell}{\partial [H(\mathbf{x}_i)]}}_{r_i} h(x) \quad (20.5)$$

我们就可以进行 boosting。

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

 **注 20.2.** $\sum_{i=1}^n r_i h(\mathbf{x}_i) < 0$ 意味着 loss 会下降，推导如下：

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

20.3 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  $H_T$ 

```

20.4 案例 1: 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 20.3. 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 loss(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, r 表示一个由 y 指向 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  $H$ 

```

20.5 案例 1: AdaBoost

已知条件:

- ▶ 设置: 分类任务 ($y_i \in \{+1, -1\}$)
- ▶ 弱学习器: $h \in \mathbb{H}, h(\mathbf{x}) \in \{-1, +1\}, \forall x$
- ▶ 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 $\ell(H) = \sum_{i=1}^n e^{-y_i H(\mathbf{x}_i)}$

20.5.1 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)

$$h(\mathbf{x}) = \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i=1}^n r_i h(\mathbf{x}_i) \quad \left(\text{替换: } r_i = e^{-H(\mathbf{x}_i)y_i} \right) \quad (20.6)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} - \sum_{i=1}^n y_i e^{-H(\mathbf{x}_i)y_i} h(\mathbf{x}_i) \quad \left(\text{替换: } w_i = \frac{1}{Z} e^{-H(\mathbf{x}_i)y_i} \right) \quad (20.7)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} - \sum_{i=1}^n w_i y_i h(\mathbf{x}_i) \quad \left(h(\mathbf{x}_i)y_i = 1 \iff h(\mathbf{x}_i) = y_i \right) \quad (20.8)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i - \sum_{i:h(\mathbf{x}_i) = y_i} w_i \quad \left(\sum_{i:h(\mathbf{x}_i) = y_i} w_i = 1 - \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i \right) \quad (20.9)$$

$$= \operatorname{argmin}_{h \in \mathbb{H}} \sum_{i:h(\mathbf{x}_i) \neq y_i} w_i \quad \left(\text{分类错误的加权和} \right) \quad (20.10)$$

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 $\epsilon < 0.5$ 。

20.5.2 寻找步长

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：

$$\alpha = \operatorname{argmin}_{\alpha} \ell(H + \alpha h) \quad (20.11)$$

$$= \operatorname{argmin}_{\alpha} \sum_{i=1}^n e^{-y_i [H(\mathbf{x}_i) + \alpha h(\mathbf{x}_i)]} \quad (20.12)$$

对 α 求导并令其为 0，我们有如下推导过程：

$$\sum_{i=1}^n y_i h(\mathbf{x}_i) e^{-y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i)} = 0 \quad \left(y_i h(\mathbf{x}_i) \in \{+1, -1\} \right) \quad (20.13)$$

$$- \sum_{i:h(\mathbf{x}_i)y_i=1} e^{\underbrace{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))}_1} + \sum_{i:h(\mathbf{x}_i)y_i=-1} e^{\underbrace{-(y_i H(\mathbf{x}_i) + \alpha y_i h(\mathbf{x}_i))}_{-1}} = 0 \quad \left(w_i = \frac{1}{Z} e^{-y_i H(\mathbf{x}_i)} \right) \quad (20.14)$$

$$- \sum_{i:h(\mathbf{x}_i)y_i=1} w_i e^{-\alpha} + \sum_{i:h(\mathbf{x}_i)y_i=-1} w_i e^{\alpha} = 0 \quad \left(\epsilon = \sum_{i:h(\mathbf{x}_i)y_i=-1} w_i \right) \quad (20.15)$$

$$-(1 - \epsilon) e^{-\alpha} + \epsilon e^{\alpha} = 0 \quad (20.16)$$

$$e^{2\alpha} = \frac{1 - \epsilon}{\epsilon} \quad (20.17)$$

$$\alpha = \frac{1}{2} \ln \frac{1 - \epsilon}{\epsilon} \quad (20.18)$$

我们发现最优的步长大小是可以求解 (in closed form) 的，因此 AdaBoost 收敛速度很快！

20.5.3 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：

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

AdaBoost 的算法流程：

 **注 20.4.** 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 20.5. 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 $\alpha = 0$ 。

```

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 ( $H_T$ )

```

20.5.4 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！

20.6 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，它定义了一个算法家族，如梯度 Boosting，AdaBoost，LogitBoost 和很多其他的，其中梯度 Boosting 是排序问题 (learning to rank) 最好的方法之一！