



Machine Learning

机器学习经典之作

作者：Mingkui Tan

组织：SMIL

更新：September 25, 2019

版本：0.1



Victory won't come to us unless we go to it. — M Moore

目录

1 简介	1
1.1 数据	1
1.1.1 数据的重要性	1
1.1.2 数据存取	3
1.2 机器学习	4
1.2.1 机器学习与传统编程的不同	4
1.2.2 算法分类	5
1.2.3 机器学习模型演示	6
2 线性回归	7
2.1 线性模型	7
2.2 损失函数	8
2.3 线性回归闭式解	11
2.4 岭回归	14
2.5 梯度下降法	18
2.6 梯度下降法有效性解释	19
2.7 拓展阅读	20
2.7.1 L1 正则化回归	20
2.7.2 多项式回归	21
2.7.3 损失函数的概率解释	23
2.7.4 广义线性模型	24
2.8 本章总结	26
3 线性分类、多分类	30
3.1 线性分类模型	30
3.2 线性判别函数和决策边界	32
3.3 二分类	32
3.3.1 支持向量机（SVM）	34
3.3.2 感知器	39
3.3.3 参数平均感知器	43
3.4 多分类	44
3.4.1 感知器用于多分类	46

3.4.2 多分类感知器的收敛性	46
3.5 更多	47
3.6 章节总结	47
4 逻辑回归	50
4.1 简介	50
4.2 逻辑函数	51
4.3 代价函数	53
4.4 Softmax	55
4.5 简介	55
4.6 多分类	55
4.7 Softmax 与逻辑回归	57
5 梯度下降 (Gradient Descent)	58
5.1 梯度下降 (Gradient Descent)	58
5.2 梯度下降的理论基础	59
5.3 随机梯度下降 (Stochastic Gradient Descent)	59
5.4 批量随机梯度下降 (Minibatch SGD)	61
5.5 例子	62
5.6 其他方法	63
5.6.1 Momentum	63
5.6.2 Adagrad	64
5.6.3 RMSProp	65
5.6.4 Adam	65
6 过拟合、正则化和验证	67
6.1 过拟合 – Overfitting	67
6.1.1 训练误差和泛化误差	67
6.1.2 过拟合与欠拟合	68
6.1.3 如何减少过拟合	68
6.2 正则化 – Regularization	69
6.2.1 什么是正则化	69
6.2.2 L1 和 L2 正则化	70
6.3 验证 – Validation	71
6.3.1 保留交叉验证	71
6.3.2 K 折交叉验证	72



7 决策树和随机森林	74
7.1 决策树	74
7.1.1 决策树的概念	74
7.1.2 决策树的生成	74
7.1.3 选择属性	76
7.1.4 剪枝	82
7.1.5 小结	83
7.2 随机森林	84
7.2.1 Bagging 算法	84
7.2.2 随机森林	84
8 AdaBoost、梯度提升决策树和 XGBoost	88
8.1 AdaBoost	88
8.2 梯度提升决策树	92
8.2.1 分类与回归树	92
8.2.2 梯度提升（Gradient Boosting, GB）	95
8.2.3 梯度提升决策树（Gradient Boosting Decision Tree, GBDT）	97
8.3 XGBoost	97
8.3.1 理论储备	98
8.3.2 XGBoost 基本思想	98
8.3.3 XGBoost 算法实现	101
8.3.4 XGBoost 代码实现	104
8.3.5 advanced topic	104
9 深度神经网络	107
9.1 神经网络的结构	108
9.1.1 神经元模型	108
9.1.2 感知机与多层网络	109
9.2 神经网络两大特性	110
9.2.1 激活函数实现去线性化	110
9.2.2 多层解决异或问题	110
9.2.3 从数学的视角上理解神经网络的功能	110
9.3 神经网络的优化	111
9.3.1 反向传播算法	112
10 卷积神经网络	113
10.1 “猫”脸识别	114
10.2 卷积	114



10.2.1 一维卷积	114
10.2.2 二维卷积	115
10.2.3 卷积的变种	116
10.2.4 图像	119
10.3 网络结构	120
10.3.1 用卷积代替全连接	120
10.3.2 卷积层	121
10.3.3 激活函数	121
10.3.4 汇聚层	122
10.3.5 典型的卷积网络结构	124
10.4 参数学习	125
10.4.1 误差项的计算	125
10.5 提升技巧	127
10.5.1 数据增强	127
10.5.2 预处理	127
10.5.3 训练技巧	127
10.5.4 正则化	128
10.6 几种典型的卷积神经网络	128
10.6.1 LeNet-5	128
10.6.2 AlexNet	129
10.6.3 Inception 网络	130
10.6.4 残差网络	131
10.7 训练过程注意事项	132
10.8 其他卷积方式	132
10.8.1 空洞卷积	132
10.8.2 转置卷积	133
10.9 Advanced Topic	135
10.9.1 互相关	135
10.9.2 卷积的数学性质	136
10.9.3 连接表	137
10.10 总结	137
11 RNN & LSTM	139
11.1 RNN	139
11.1.1 为什么需要循环神经网络	139
11.1.2 不同类型的循环神经网络	140
11.1.3 参数更新	144



11.1.4 长期记忆中的问题	149
11.2 LSTM	151
11.2.1 L 还是 S?	151
11.2.2 三门分立	152
11.2.3 内外有别	154
11.2.4 LSTM 循环单元中的数据流	155
11.2.5 LSTM-变体	156
11.3 章末总结	157
11.4 Advanced Topic	158
12 Attention Mechanism	160
12.1 为什么需要注意力机制?	160
12.2 从仿生角度认识注意力	160
12.3 注意力评分函数	161
12.4 硬性注意力机制 (Hard Attention Mechanism)	162
12.5 软性注意力机制 (Soft Attention Mechanism)	163
12.6 软硬结合: 局部注意力机制 (Local Attention Mechanism)	164
12.7 自注意力机制 (Self-attention Mechanism)	166
12.7.1 键值对注意力 (Key-Value Attention)	166
12.7.2 多头注意力 (Multi-head Attention)	167
12.7.3 Attention Is All You Need	168
12.8 章末总结	170
12.9 Advanced Topic	171
13 聚类	173
13.1 引言	173
13.2 聚类方法与性能度量指标	173
13.3 划分聚类	174
13.3.1 K-Means	175
13.4 层次聚类	181
13.5 密度聚类	183
13.6 Advanced Topics	184
13.6.1 Birch 聚类	184
14 降维	190
14.1 降维的必要性	190
14.2 特征选择	191
14.2.1 过滤法	191



14.2.2 包装法	193
14.2.3 嵌入法	193
14.3 特征提取	194
14.3.1 矩阵与线性变换	195
14.3.2 主成分分析法 (Principal Component Analysis)	196
14.3.3 算法实例	198
14.3.4 延伸话题	200
15 推荐系统	209
15.1 概述	209
15.1.1 什么是推荐系统	209
15.1.2 发展历史	209
15.1.3 推荐系统的描述方式	210
15.2 用户画像	212
15.2.1 定性分析用户画像	212
15.2.2 定量分析用户画像	212
15.3 算法的分类	213
15.3.1 基于内容的推荐	213
15.3.2 基于协同过滤的推荐系统	217
15.4 实际应用	225
15.5 前沿方向	226
15.5.1 基于 AE 的协同过滤	226
15.5.2 基于 RNN 的新闻推荐	226
15.5.3 基于 Transformer 的电商推荐	226



第 1 章 简介

过去十年，机器学习领域经历了翻天覆地的变化。从一个纯粹的学术和研究方向开始，我们已经看到了机器学习在各个领域都有着广泛的应用，如零售，技术，医疗保健，科学等等。在 21 世纪，数据科学和机器学习的目标已经转变为解决现实问题，实现复杂任务的自动化。日新月异的科技带给人们许多便捷的生活体验，将人类的科技生活提高到了新的高度。

在讨论机器学习算法之前，我们先来看看人是怎么开始学习的：在面对一个具体的问题时，人会首先通过已有的经验和当前的信息作出反应动作，然后按照过程中的一些反馈来修改自己的经验，并不断重复这个过程。

机器学习就是通过程序让计算机“学会”人的学习过程。换句话说，机器学习算法是使得计算机变得像人一样“智能”的工具。机器学习算法试图从数据中学习潜在模式和关系，而不是写死规则。

1.1 数据

1.1.1 数据的重要性

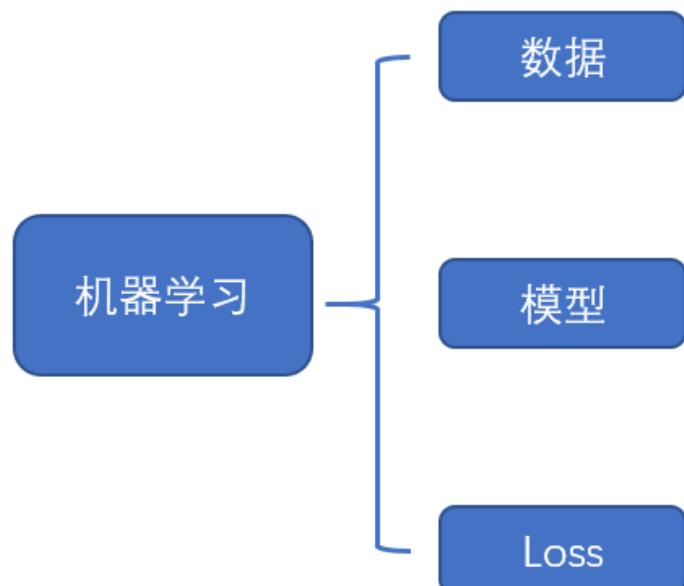


图 1.1: 机器学习三要素

如图1.1所示，机器学习三要素由数据，模型，loss 函数组成，而数据是机器学习的基石，没有数据我们无法训练模型，自然也就没法知道我们的模型是不是能够满足我们的需求。

大数据时代，数据量呈现爆炸式增长，怎么样在众多的数据中获取我们想得到的数据成为了机器学习的首要问题。现实世界中存在许多公开的数据集，这些数据集通常是经过清理的数据集，比较标准，一般作为公开的标准来衡量新算法的有效性。比如图像识别中的 MNIST 数据集(图1.2)，自然语言处理中的 SQuAD 数据集(图1.3)，语音识别中的 TED-LIUM 数据集(图1.4)等等。



图 1.2: MNIST 数据集

Passage Segment

...The European Parliament and the Council of the European Union have powers of amendment and veto during the legislative process...

Question

Which governing bodies have veto power?

图 1.3: SQuAD 数据集

根据数据的不同我们面临的机器学习问题也是不同的。图1.5显示了一些比较常见的具体类别数据所对应的机器学习任务，在这些问题中数据力量不容小觑。新的机器学习算法就是通过优化不断刷新数据集能够达到的精度，向着数据集能够达到的极限努力，来实现现实应用中的精度提升。

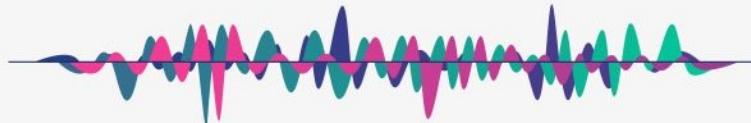


图 1.4: TED-LIUM 数据集

1.1.2 数据存取

有了数据我们怎么读取数据为我们所用是另外一个重要的问题。下面我们举几个常见的例子来说明怎么读取数据。

1.1.2.1 LIBSVM 数据格式

LIBSVM 是台湾大学林智仁 (Lin Chih-Jen) 教授等开发设计的一个简单、易于使用和快速有效的 SVM 模式识别与回归的软件包，这套库可以很方便的对数据做分类或回归。由于 LIBSVM 程序小，运用灵活，输入参数少，并且是开源的，易于扩展，因此成为目前国内应用最多的 SVM 的库。这套库可以从 <http://www.csie.ntu.edu.tw/~cjlin/> 免费获得。我们使用他们公开的数据集作为数据存取的例子，数据集可以在 <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/> 上免费下载，我们以第一个分类数据集 a1a <https://www.csie.ntu.edu.tw/~cjlin/libsvm-tools/datasets/binary.html> 为例进行演示。

首先我们下载数据集 a1a 和 a1a.t，其中 a1a 是训练数据集，a1a.t 是检验测试数据集。使用这两个文件之前，我们需要对这两个文件的数据存储格式进行了解。打开数据集文件 a1a 如下图1.6所示，数据被分为 3 个字段，label, index, value。label 指的是当前这个样本的类别标签，+1 是正类，-1 是负类。index 指的是当前样本的特征索引，这个网站上的数据特征是从 1 开始计数的（意思是），2 指第二个特征，6 指第六个特征。value 指的是当前该样本某个索引下特征对应的值，2 : 1 就是指当前样本的第二个特征的值为 1。训练数据集和测试数据集的存储情况是相同的。这样的存储格式我们称之为稀疏存储，在给定特征数量的情况下，默认将没有列出来的索引对应的特征值置为 0，因为在实际情况中为 0 的数据占多数，这样的存储格式能够节省很大的存储空间。

读取 LIBSVM 数据集我们使用 scikit-learn 中的方法 `load_svmlight_file` 来进行读取，记得要指定 `n_feature` 的值，不然可能会出错。

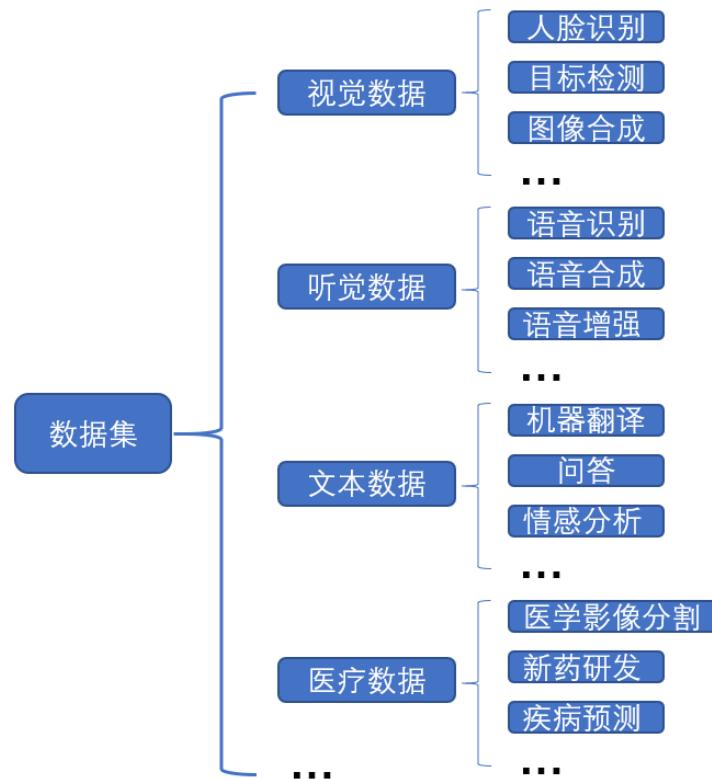


图 1.5: 机器学习任务

```

[[label] [index1]:[value1] [index2]:[value2] ...]
[[label] [index1]:[value1] [index2]:[value2] ...]

-1 2:1 6:1 18:1 20:1 37:1 42:1 48:1 64:1 71:1 73:1 74:1 76:1 81:1 83:1
+1 5:1 11:1 15:1 32:1 39:1 40:1 52:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
-1 5:1 16:1 30:1 35:1 41:1 64:1 67:1 73:1 74:1 76:1 80:1 83:1
-1 5:1 6:1 15:1 20:1 37:1 40:1 50:1 63:1 67:1 73:1 75:1 76:1 80:1 83:1
-1 5:1 7:1 16:1 29:1 39:1 40:1 48:1 63:1 67:1 73:1 74:1 76:1 78:1 83:1
  
```

图 1.6: LIBSVM 数据集示例

存储 LIBSVM 数据集我们使用 scikit-learn 中的方法 `dump_svmlight_file` 来进行存储，因为它默认特征索引是从零开始计数的，为了验证我们读取后再存储的数据没有差别，我们将 `zero_based` 设置为 `False`。

1.2 机器学习

1.2.1 机器学习与传统编程的不同

传统编程的公式：规则 + 数据 = 答案

机器学习的公式：答案 + 数据 = 规则

核心都是规则，最终目标是答案。世界都在数字化，能用传统编程方式模拟的规则都是一些简单的规则，而机器学习提供了一种探究复杂规则得途径。可以

解决传统编程无法解决的一些问题。所以机器学习只是特定领域的一种解决方案。而现实中的解决方案大部分还是由传统编程解决的。

世界上的任何东西都是有规则的，都可以用数学表示。机器学习通过穷举的方式使规则接近完美（监督学习），大数据的出现更加增强了这个效果。

1.2.2 算法分类

在机器学习中有两大类算法，一种是有监督机器学习算法，一种是无监督机器学习算法。

1.2.2.1 有监督机器学习算法

有监督机器学习算是是有特征（feature）和每个特征所对应的标签（label）的。举例理解：高考试卷是有标准答案的，在做题的过程中，我们可以通过对照标准答案（label），分析我们出错的原因（loss）。对照标签进行学习的过程就是监督学习。

有监督机器学习算法有两类基本的问题，回归和分类。

通过房地产市场的数据，预测一个给定面积的房屋的价格就是一个回归问题。这里我们可以把价格看成是面积的函数，它是一个连续的输出值。

给定医学数据，通过肿瘤的大小来预测该肿瘤是恶性瘤还是良性瘤，这就是一个分类问题，它的输出是 0 或者 1 两个离散的值。（0 代表良性，1 代表恶性）。分类问题的输出可以多于两个，比如在该例子中可以有 0,1,2,3 四种输出，分别对应 {良性，第一类肿瘤，第二类肿瘤，第三类肿瘤}。

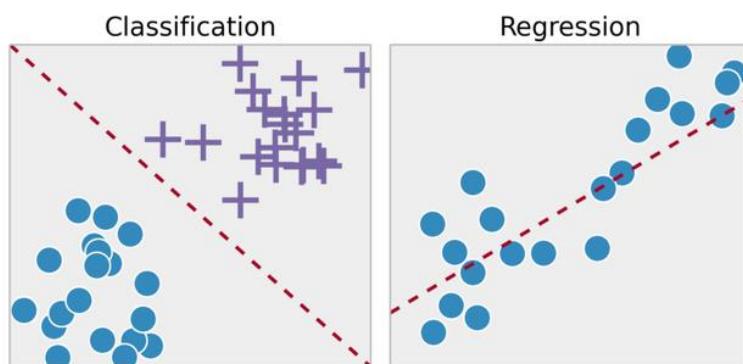


图 1.7: 有监督的分类和回归实例

1.2.2.2 无监督机器学习算法

无监督机器学习中数据没有被标记，也就是说没有对应的标签（label）。样本数据类别未知，需要根据样本间的相似性对样本集进行分类。举例理解：乒乓

球和篮球放在一起我们将其分开，依据就是直径大小的区别，我们并不用知道哪一种是乒乓球，哪一种是篮球，只需要分开就行了。

无监督学习中最常见的任务是聚类，表示学习和密度估计。在所有这些情况下，我们希望了解我们数据的内在结构，而不使用显式提供的标签。一些常用算法包括 k 均值聚类、主成分分析和自动编码器。由于没有提供标签，因此没有具体的方法来比较大多数无监督学习方法中的模型性能。

Google News 通过搜集网上的新闻，并且根据新闻的主题将新闻分成许多簇，然后将在同一个簇的新闻放在一起。当打开各个新闻链接的时候，展现的都是和这个类别有关的新闻，比如财经类，体育类。

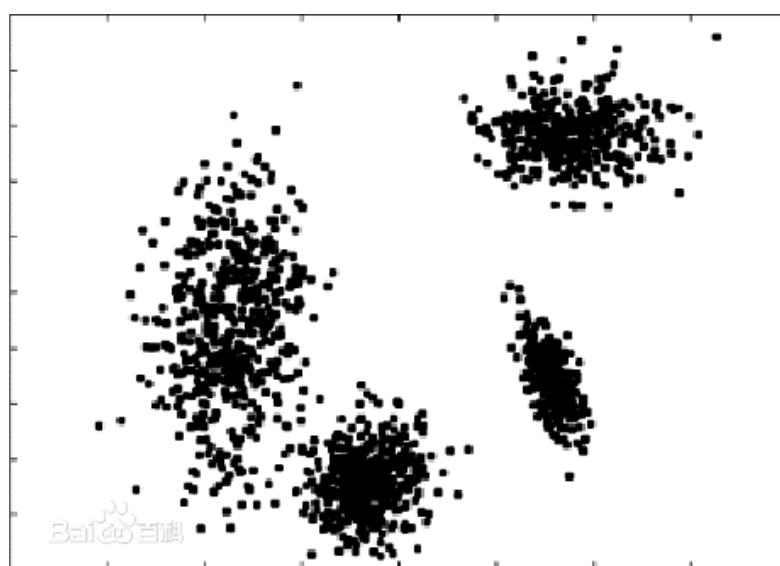


图 1.8: 无监督的聚类实例

1.2.3 机器学习模型演示

等待资源中。

第 2 章 线性回归

2.1 线性模型

坐在电脑前的小明，眼神凝重，握着鼠标的手来回移动，似乎在犹豫着什么。原来小明在考虑究竟要不要选机器学习这门课，毕竟近些年机器学习、人工智能的热度挺火的。富有好奇心的小明当然也有着一窥机器学习的想法。但是小明本身也成绩平平，要是最后没有通过机器学习的考核，或者这门课分数很低，这些都不是小明想要的。这该怎么办呢？要是能预测一下自己的机器学习成绩就好了。突然小明想到中学时期学过这样的一次函数

$$f(x) = kx + b \quad (2.1)$$

其中 k 是这个线性函数的斜率， b 为截距，也就是偏置项

也就是说，某数学函数或数量关系的函数图形呈现一条直线或线段，这样的关系就是一个线性关系。同时，这条公式也能代表一个模型，只有一个自变量的线性模型。但是，这个只有一个自变量的线性模型过于简单了，小明需要通过几门课程的成绩预测自己的机器学习成绩，这将如何改进呢？当我们需要表示多个自变量对一个因变量的关系时，我们需要更一般的线性模型。即

$$\mathbf{x} = (x_1, x_2, \dots, x_d)^\top \quad (2.2)$$

$$\mathbf{w} = (w_1, w_2, \dots, w_d)^\top \quad (2.3)$$

$$f(\mathbf{x}) = w_1x_1 + w_2x_2 + \dots + w_dx_d + b \quad (2.4)$$

$$\begin{aligned} &= \sum_{i=1}^d w_i x_i + b \\ &= \mathbf{w}^\top \mathbf{x} + b \end{aligned}$$

其中 \mathbf{x} 是指包含 d 个自变量的列向量，我们称之为特征向量。关于 \mathbf{w} ，我们不用斜率来命名，用更一般的权重来命名，可以表示每个自变量的重要程度，则 \mathbf{w} 是包含 d 个权重的列向量，我们称之为权重向量。 b 为偏置项。

为了使本章公式简洁以及易于推导，我们将线性模型简写成

$$\hat{\mathbf{x}} = (1, x_1, x_2, \dots, x_d)^\top \quad (2.5)$$

$$\hat{\mathbf{w}} = (b, w_1, w_2, \dots, w_d)^\top \quad (2.6)$$

$$f(\hat{\mathbf{x}}) = \hat{\mathbf{w}}^\top \hat{\mathbf{x}} \quad (2.7)$$

其中 $\hat{\mathbf{w}}$ 是增广权重向量， $\hat{\mathbf{x}}$ 是增广特征向量。

在保证本章后续符号表示没有歧义的情况下，我们使用 \mathbf{w} 代表增广权重向量， \mathbf{x} 代表增广特征向量。那么线性模型最后写为

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \quad (2.8)$$

线性模型是机器学习中最简单的模型，但是其中也包含了机器学习的一些重要思想。而线性回归要做的是通过一个用于训练的包含 n 个样本的数据集 $\mathcal{T} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_n, y_n)\}$, $\mathbf{x}_i \in \mathbb{R}^{n \times 1}$, $y_i \in \mathbb{R}$, 试图学得一个线性模型

$$f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i \quad (2.9)$$

可以使得 $f(\mathbf{x}_i) \approx y_i$ 。那如果我们要用这个线性模型帮助小明预测他的机器学习成绩，首先就需要收集以前师兄师姐的各门课程成绩作为 \mathbf{x}_i ，对应的机器学习成绩作为 y_i 构造训练集，通过这些数据学得一个可以预测机器学习成绩的线性模型。那么首先要做就是找到合适的 \mathbf{w} 使得线性模型尽可能预测准训练集中各个师兄师姐的机器学习成绩，换句话说就是让预测值与真实值的误差尽可能小。只有这样，预测小明的机器学习成绩才可能比较准确。如何寻找合适的 \mathbf{w} ，使得的训练集中每一个样本 \mathbf{x}_i ，经过线性模型的映射之后得到的预测值 $f(\mathbf{x}_i)$ ，与真实值 y_i 的差异都尽可能小呢？这是一个最优化问题，那么首先就需要一个函数可以衡量所有预测值与真实值的差异的总和，即损失函数。

2.2 损失函数

关于损失函数，我们可以很天然地想到使用差值函数作为描述两个值之间的差异，即该损失函数如下

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i) \quad (2.10)$$

但是在线性回归中，大量的样本通过线性模型的映射后，得到的预测值可能大于对应的真实值，也可能小于对应的真实值。如果使用差值函数作为损失函数，

对每个预测值与真实值的差值求和后，正负值抵消，无法衡量整体预测值与真实值的差异。

于是，我们很自然地进一步改进为对每个预测值与真实值的差值取绝对值后求和，那么这个绝对值损失函数表示为

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n |y_i - \mathbf{w}^\top \mathbf{x}_i| \quad (2.11)$$

但是，在线性回归中，绝对值损失函数并不算是一个上佳的损失函数，因为寻找合适的 \mathbf{w} 需要损失函数最优化，即最小化。而绝对值损失函数存在不可导点，必然会对优化过程存在影响。那么我们也容易想到另一个损失函数——平方损失函数，即

$$\mathcal{L}(\mathbf{w}) = \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \quad (2.12)$$

平方损失函数有着很好的几何意义，对应了我们常用的距离，即欧几里得距离。所以在线性回归中，使用平方损失函数的意义就是希望找到一条直线，使得所有样本到直线上的欧几里得距离之和最小。况且，平方损失函数处处可导，且导函数的形式也较为简单。相比于四次方、六次方等高次方损失函数，平方损失函数的函数结构简单，便于计算。所以对于线性回归来说，这是一个上佳的损失函数。

在不影响平方损失函数的使用效果的情况下，为了方便平方损失函数后续的求导化简，我们把平方损失函数写成如下形式

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \quad (2.13)$$

为了方便本章后续的公式推导，我们需要把训练集中的每个样本中的 \mathbf{x}_i 综合起来作为整体设计成一个矩阵，每一个对应的 y_i 作为整体也要设计成一个向



量。

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \quad (2.14)$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad (2.15)$$

其中 $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$, $\mathbf{y} \in \mathbb{R}^{n \times 1}$, 因为 \mathbf{X} 中有 d 个特征元素和一个增广后的元素 1, 并且有 n 个样本, 所以 \mathbf{X} 为 n 行 $d+1$ 列, \mathbf{y} 为 n 行 1 列。

所以平方损失函数可以化成如下形式

$$\begin{aligned} \mathcal{L}(\mathbf{w}) &= \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\ &= \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{x}_i^\top \mathbf{w})^2 \\ &= \frac{1}{2} \begin{bmatrix} y_1 - \mathbf{x}_1^\top \mathbf{w} \\ \vdots \\ y_n - \mathbf{x}_n^\top \mathbf{w} \end{bmatrix}^\top \begin{bmatrix} y_1 - \mathbf{x}_1^\top \mathbf{w} \\ \vdots \\ y_n - \mathbf{x}_n^\top \mathbf{w} \end{bmatrix} \\ &= \frac{1}{2} \left(\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \mathbf{w} \right)^\top \left(\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} \mathbf{x}_1^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} \mathbf{w} \right) \\ &= \frac{1}{2} (\mathbf{y} - \mathbf{X}\mathbf{w})^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) \end{aligned} \quad (2.16)$$

如果读者读到这里, 对于损失函数的选取还存在疑问, 不妨阅读本章拓展阅读部分, 该部分会从概率的角度阐述为何选取平方损失函数作为线性回归的损失函数。而下一节我们会介绍在使用平方损失函数的情况下, 如何寻找 \mathbf{w} , 使得损失函数尽可能小, 从而求解线性回归。

2.3 线性回归闭式解

如何寻找合适的 \mathbf{w} , 使得损失函数 $\mathcal{L}(\mathbf{w})$ 最小化? 根据最优化理论, 可列如下等式

$$\mathbf{w}^* = \arg \min_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \quad (2.17)$$

上式中 \mathbf{w}^* 表示最优的 \mathbf{w} 。

一般来说, 优化平方损失函数 $\mathcal{L}(\mathbf{w})$ 需要知道 $\mathcal{L}(\mathbf{w})$ 对于 \mathbf{w} 的导数, 这个对向量求导的导数是一个包含所有向量元素偏导数的向量, 即梯度。

为了简化梯度计算, 令

$$\mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{w} \quad (2.18)$$

那么 $\mathcal{L}(\mathbf{w})$ 对 \mathbf{w} 的梯度为

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \frac{\partial (\frac{1}{2} \mathbf{a}^\top \mathbf{a})}{\partial \mathbf{a}} \\ &= \frac{1}{2} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \frac{\partial \mathbf{a}^\top \mathbf{a}}{\partial \mathbf{a}} \\ &= \frac{1}{2} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} (2\mathbf{a}) \\ &= \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \\ &= \frac{\partial (\mathbf{y} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= -\frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= -\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) \\ &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \end{aligned} \quad (2.19)$$

上式中的 $\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$ 与 \mathbf{w} 的维度相同.

易知平方损失函数 $\mathcal{L}(\mathbf{w})$ 是凸二次函数, 当其梯度为 $\mathbf{0}$ 时, $\mathcal{L}(\mathbf{w})$ 取得最小值.

$$\begin{aligned} \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) = \mathbf{0} \\ \Rightarrow \mathbf{X}^\top \mathbf{X}\mathbf{w} &= \mathbf{X}^\top \mathbf{y} \end{aligned} \quad (2.20)$$

上式中的 $\mathbf{0}$ 指的是与 $\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$ 维度相同的全零向量。

若 $\mathbf{X}^\top \mathbf{X}$ 为可逆矩阵或满秩矩阵时，可以解得 \mathbf{w}^*

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} \quad (2.21)$$

这就是线性回归的闭式解，利用这个 \mathbf{w}^* 就可以顺利地预测小明的机器学习成绩了，而这种方法也叫最小二乘估计（Least-squares Estimation），二乘即平方。

下面我们通过一个小算例来说明线性回归的闭式解，小明通过问询得到十位师兄师姐的 C++(上) 成绩、线性代数成绩和机器学习成绩，如表2.1所示。

表 2.1：十位师兄师姐的成绩

C++(上)	线性代数	机器学习
78	66	77
70	93	86
61	71	60
73	66	69
79	81	70
93	95	88
74	77	72
90	85	88
66	64	70
81	90	91

如果利用前六位师兄师姐的成绩构造训练集去进行线性回归，则可令

$$\mathbf{X} = \begin{bmatrix} 1 & 78 & 66 \\ 1 & 70 & 93 \\ 1 & 61 & 71 \\ 1 & 73 & 66 \\ 1 & 79 & 81 \\ 1 & 93 & 95 \end{bmatrix} \quad (2.22)$$

$$\mathbf{y} = \begin{bmatrix} 77 \\ 86 \\ 60 \\ 69 \\ 70 \\ 88 \end{bmatrix} \quad (2.23)$$

根据公式2.21，此时可以得到 \mathbf{w}^* 为

$$\mathbf{w}^* = \begin{bmatrix} 7.74 \\ 0.42 \\ 0.45 \end{bmatrix} \quad (2.24)$$

在这个数据集中，我们可以看到线性代数成绩在模型中的权重是要比 C++(上) 成绩大的，即线性模型在这个数据集中认为线性代数成绩比 C++(上) 成绩重要。

于是， \mathbf{X} 经过这个线性模型的映射后，为

$$\mathbf{X}\mathbf{w}^* = \begin{bmatrix} 70.26 \\ 79.09 \\ 65.39 \\ 68.16 \\ 77.45 \\ 89.65 \end{bmatrix} \quad (2.25)$$

根据公式2.16，可得训练集误差为 $\mathcal{L}_{train} = 90.53$ 。尽管训练集误差 \mathcal{L}_{train} 已经降到最小了，由于样本还是比较少，以及线性模型本身的局限性， $\mathbf{X}\mathbf{w}^*$ 对 \mathbf{y} 的预测只能说差强人意吧。

剩下四位师兄师姐的成绩作为训练集，相关计算如下

$$\mathbf{X}_{test} = \begin{bmatrix} 1 & 74 & 77 \\ 1 & 90 & 85 \\ 1 & 66 & 64 \\ 1 & 81 & 90 \end{bmatrix} \quad (2.26)$$

$$\mathbf{y}_{test} = \begin{bmatrix} 72 \\ 88 \\ 70 \\ 91 \end{bmatrix} \quad (2.27)$$

$$\mathbf{X}_{test}\mathbf{w}^* = \begin{bmatrix} 73.55 \\ 83.87 \\ 64.32 \\ 82.35 \end{bmatrix} \quad (2.28)$$

$$\mathcal{L}_{test} = 63.21 \quad (2.29)$$

其中 \mathcal{L}_{test} 表示代表训练集误差。

而小明 C++(上) 成绩和线性代数成绩如表2.2所示，则通过该线性模型预测

他的机器学习成绩为 77.74。

表 2.2: 小明两门课程的成绩

C++ (上)	线性代数
84	77

但我们也知道，在不少情况下 $\mathbf{X}^T \mathbf{X}$ 是不可逆的，比如 \mathbf{X} 的行数少于列数，即训练集的样本数过少。比如小明想通过六门课程的成绩预测机器学习的成绩，但是只收集到三个师兄师姐的六门课程及机器学习成绩。这时，可能会存在多个 \mathbf{w}^* ，那么下面我们介绍岭回归来解决这个问题。

2.4 岭回归

从上一节我们知道，存在 $\mathbf{X}^T \mathbf{X}$ 不可逆的情况。或者就算 $\mathbf{X}^T \mathbf{X}$ 是可逆的，但存在极小的特征值， $\mathbf{X}^T \mathbf{X}$ 接近于奇异矩阵。这时，特征变量间存在较大的线性相关关系，使用传统的线性回归模型并不能很好地估计 \mathbf{w} ，对于不同的训练集估计方差会变得很大，这表现为存在很大的正权重系数项，然后被另一个同样大的与之相关的负权重系数项抵消，那么整个线性模型对输入变量中的噪音非常敏感。如果小明的各门成绩输入到如此敏感的线性模型去预测机器学习成绩时，可能得到的成绩会低于 0 分，或者高于 100 分，不符合实际。而如果我们去限制参数的大小，理所当然地会使线性模型对噪音的敏感度降低。这也就是岭回归（Ridge Regression）的基本思想。

岭回归实际上是一种改良的最小二乘估计法，通过损失模型部分的解释性、降低回归精度使得权重系数更为可靠及更符合实际。而岭回归的具体操作是让平方损失函数 $\mathcal{L}(\mathbf{w})$ 增加一个惩罚项，通过限制 \mathbf{w} 的增长，从而使线性模型对输入的噪音的敏感程度降低。公式如下

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{1}{2}\lambda\mathbf{w}^T\mathbf{w} \quad (2.30)$$

式中的 λ 是惩罚系数， $\lambda > 0$ ，其值越大，对 \mathbf{w} 的限制能力越强，而 $\frac{1}{2}\lambda$ 中的 $\frac{1}{2}$ 则是为了后续的求导化简。

同样地，为了简化梯度计算，令 $\mathbf{a} = \mathbf{y} - \mathbf{X}\mathbf{w}$, $\mathcal{L}(\mathbf{w})$ 对 \mathbf{w} 的梯度为

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{2} \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \frac{\partial \mathbf{a}^\top \mathbf{a}}{\partial \mathbf{a}} + \frac{1}{2} \lambda \frac{\partial \mathbf{w}^\top \mathbf{w}}{\partial \mathbf{w}} \\ &= \frac{1}{2} \left(2 \frac{\partial \mathbf{a}}{\partial \mathbf{w}} \mathbf{a} + 2 \lambda \mathbf{w} \right) \\ &= \frac{\partial (\mathbf{y} - \mathbf{X}\mathbf{w})}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w} \\ &= -\frac{\partial \mathbf{X}\mathbf{w}}{\partial \mathbf{w}} (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w} \\ &= -\mathbf{X}^\top (\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \mathbf{w} \\ &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w}\end{aligned}\tag{2.31}$$

让 $\mathcal{L}(\mathbf{w})$ 的梯度为 $\mathbf{0}$ ，则

$$\begin{aligned}\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} &= \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) + \lambda \mathbf{w} = \mathbf{0} \\ \Rightarrow (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})\mathbf{w} &= \mathbf{X}^\top \mathbf{y}\end{aligned}\tag{2.32}$$

式中的 \mathbf{I} 是单位矩阵，除了对角线的元素为 1，其余部分皆为 0，该式中的 \mathbf{I} 与 $\mathbf{X}^\top \mathbf{X}$ 维度相同。

岭回归可以解决 $\mathbf{X}^\top \mathbf{X}$ 不可逆的问题，因为 $\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}$ 必然是可逆矩阵，证明如下。

首先，根据线性代数的奇异值分解（Singular Value Decomposition），可以把 \mathbf{X} 分解成如下形式

$$\mathbf{X} = \mathbf{U}\Sigma\mathbf{V}^\top\tag{2.33}$$

其中 $\mathbf{X} \in \mathbb{R}^{n \times (d+1)}$, $\mathbf{U} \in \mathbb{R}^{n \times n}$ 为正交矩阵， $\mathbf{V} \in \mathbb{R}^{(d+1) \times (d+1)}$ 为正交矩阵，而 $\Sigma \in \mathbb{R}^{n \times (d+1)}$ 则是非负实数对角矩阵。

根据正交矩阵的性质，可化简 $\mathbf{X}^\top \mathbf{X}$ 为

$$\begin{aligned}\mathbf{X}^\top \mathbf{X} &= \mathbf{V}\Sigma^\top \mathbf{U}^\top \mathbf{U}\Sigma\mathbf{V}^\top \\ &= \mathbf{V}\Sigma^\top \Sigma\mathbf{V}^\top\end{aligned}\tag{2.34}$$

那么

$$\begin{aligned}
 \mathbf{X}^T \mathbf{X} + \lambda \mathbf{I} &= \mathbf{V} \Sigma^T \Sigma \mathbf{V}^T + \lambda \mathbf{I} \\
 &= \mathbf{V} \Sigma^T \Sigma \mathbf{V}^T + \lambda \mathbf{V} \mathbf{V}^T \\
 &= (\mathbf{V} \Sigma^T \Sigma + \lambda \mathbf{V}) \mathbf{V}^T \\
 &= (\mathbf{V} \Sigma^T \Sigma + \lambda \mathbf{V} \mathbf{I}) \mathbf{V}^T \\
 &= \mathbf{V} (\Sigma^T \Sigma + \lambda \mathbf{I}) \mathbf{V}^T
 \end{aligned} \tag{2.35}$$

易知, 存在 $\lambda > 0$, 使得 $\Sigma^T \Sigma + \lambda \mathbf{I}$ 是对角线上皆为非 0 元素的对角矩阵, $\Sigma^T \Sigma + \lambda \mathbf{I}$ 可逆, 且其逆矩阵是对角元素皆为原对角元素倒数的对角矩阵。故 $\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I}$ 为可逆矩阵。

所以, \mathbf{w} 在岭回归下的最优估计 \mathbf{w}^* 是

$$\mathbf{w}^* = (\mathbf{X}^T \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \tag{2.36}$$

最后我们通过一个小算例, 结束本小节的内容。小明得到了六位师兄师姐的六门成绩及机器学习成绩, 如表2.3所示。

表 2.3: 六位师兄师姐的成绩

C++ (上)	C++(下)	数学分析 (一)	数学分析 (二)	线性代数	概率论	机器学习
78	89	68	62	66	73	77
70	77	87	95	93	77	86
61	64	60	62	71	71	60
73	56	49	66	66	68	69
79	81	73	74	81	51	70
93	85	100	100	95	97	88

那么利用前三位师兄师姐的成绩构造训练集去进行岭回归, 则可令

$$\mathbf{X} = \begin{bmatrix} 1 & 78 & 89 & 68 & 62 & 66 & 73 \\ 1 & 70 & 77 & 87 & 95 & 93 & 77 \\ 1 & 61 & 64 & 60 & 62 & 71 & 71 \end{bmatrix} \tag{2.37}$$

$$\mathbf{y} = \begin{bmatrix} 77 \\ 86 \\ 60 \end{bmatrix} \tag{2.38}$$

此时根据公式2.36, 设 $\lambda = 0.1$, 得到 \mathbf{w}^* 为

$$\mathbf{w}^* = \begin{bmatrix} -0.006 \\ 0.19 \\ 0.43 \\ 0.39 \\ 0.33 \\ -0.07 \\ -0.27 \end{bmatrix} \quad (2.39)$$

于是, \mathbf{X} 经过这个线性模型的映射后, 为

$$\mathbf{X}\mathbf{w}^* = \begin{bmatrix} 76.998 \\ 85.998 \\ 60.005 \end{bmatrix} \quad (2.40)$$

根据公式2.30, 可得训练集误差 $\mathcal{L}_{train} = 0.028$ 。利用后三个师兄师姐的成绩作为测试集, 相关计算如下

$$\mathbf{X}_{test} = \begin{bmatrix} 1 & 73 & 56 & 49 & 66 & 66 & 68 \\ 1 & 79 & 81 & 73 & 74 & 81 & 51 \\ 1 & 93 & 85 & 100 & 100 & 95 & 97 \end{bmatrix} \quad (2.41)$$

$$\mathbf{y}_{test} = \begin{bmatrix} 69 \\ 70 \\ 88 \end{bmatrix} \quad (2.42)$$

$$\mathbf{X}_{test}\mathbf{w}^* = \begin{bmatrix} 56.98 \\ 84.58 \\ 95.15 \end{bmatrix} \quad (2.43)$$

$$\mathcal{L}_{test} = 204.11 \quad (2.44)$$

其中 \mathcal{L}_{test} 为测试集误差。

小明的六门成绩如表2.4所示, 则通过该线性模型预测他的机器学习成绩为 75.17。

表 2.4: 小明的六门成绩

C++ (上)	C++(下)	数学分析 (一)	数学分析 (二)	线性代数	概率论
84	74	67	79	77	77

由于训练集的样本过少, 样本没有进行归一化, λ 的惩罚力度不够, 训练集

误差和测试集误差相差很大，这是很明显的过拟合现象，至于过拟合的详细解释，读者可以自行查阅后续章节进行学习。

2.5 梯度下降法

我们根据多元微积分的知识可以知道，某一标量场某点的方向导数的最大值是其梯度的模，当且仅当沿着梯度的方向取到，也就是说梯度的方向是该函数该点瞬时变化率最大的方向，即增长最快的方向。反过来，梯度的负方向也就是函数该点下降最快的方向。线性回归需要最小化 $\mathcal{L}(\mathbf{w})$ ，那我们需要得到 $\mathcal{L}(\mathbf{w})$ 每一点的梯度，然后用 \mathbf{w} 沿着梯度的负方向进行更新，从而实现了对 $\mathcal{L}(\mathbf{w})$ 的优化。而这就是梯度下降法的精髓所在。

为了直观理解梯度下降法，我们不妨把梯度下降过程理解成从山上某一点（初始点）下到山谷（局部最低点）的过程。一开始我们并不知道什么路径可以下山，只能观望到所站位置的周围，对周围环境进行足够地观察后，我们决定从最陡峭向下的方向（梯度的负方向）下山，步长（学习率）决定了我们下一刻的位置在哪。就这样走一步看一步，直到我们走到一个四处平坦（梯度为 0）的位置，我们就完成下到山谷的过程了。如果我们的步长过大的话，很可能直接从山坡的一边迈步到山坡的另一边，错过了山谷，然后又从山坡新的一边迈步回原来的一边，呈现“之”字形下山，也可能来回迈步无法到达山谷。但步长过小的话，下山又太慢了。

通过上面的理解我们不难看出，寻找一个合适的学习率很重要，学习率太小，整个梯度下降过程所需要的时间太长了，学习率太大可能梯度下降过程无法收敛。而梯度下降不一定能找到函数的最小值，而是找到一个局部最小值。若所需优化的函数是凸函数的话，恰好局部最小值就是全局最小值。下面，我们给出梯度下降法的具体运行过程

Algorithm 1: 梯度下降法

输入：

训练集 \mathcal{T}

学习率 η

迭代次数 t

输出：

```

for  $i = 1 \dots t$  do
     $\mathbf{w} := \mathbf{w} - \eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}$ 
end for
return  $\mathbf{w}$ 

```

回到线性回归，所幸平方损失函数是凸函数，这意味着通过合适的学习率 η ，

我们可以找到 \mathbf{w}^* , 使得 $\mathcal{L}(\mathbf{w})$ 取得最小值. 更新规则如下

$$\mathbf{w} := \mathbf{w} - \eta \mathbf{X}^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \quad (2.45)$$

上式的 $:=$ 代表赋值。

至此, 梯度下降法介绍完毕。如果读者对梯度下降法的有效性还存疑的话, 不妨继续阅读下一小节, 这将会看到较为严谨的代数解释。对此没有兴趣的读者可以跳过下一小节, 这并不影响后续章节的阅读。

2.6 梯度下降法有效性解释

因为 $\mathcal{L}(\mathbf{w})$ 是连续可微的, 根据泰勒展开公式, 当 $\Delta\mathbf{w} \rightarrow \mathbf{0}$ 时, $\mathcal{L}(\mathbf{w})$ 的一阶泰勒展开为

$$\mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) = \mathcal{L}(\mathbf{w}) + \Delta\mathbf{w}^\top \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \quad (2.46)$$

设学习率 $\eta > 0$, 不妨令

$$\Delta\mathbf{w} = -\eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \quad (2.47)$$

易知, 当 $\eta \rightarrow 0$ 时, $\Delta\mathbf{w} \rightarrow \mathbf{0}$, 那么 $\mathcal{L}(\mathbf{w})$ 的一阶泰勒展开改写为

$$\mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) = \mathcal{L}(\mathbf{w}) - \eta \left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \right)^\top \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \quad (2.48)$$

又知

$$\left(\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \right)^\top \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} \geq 0 \quad (2.49)$$

故可以找到一个很小的正数 η , 使得

$$\mathcal{L}(\mathbf{w} + \Delta\mathbf{w}) \leq \mathcal{L}(\mathbf{w}) \quad (2.50)$$

$$\Rightarrow \mathcal{L}(\mathbf{w} - \eta \frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}}) \leq \mathcal{L}(\mathbf{w}) \quad (2.51)$$

这就比较严谨地解释了为什么梯度下降法是有效的, 同时也指出了为什么学习率 η 过大会使梯度下降无法收敛到局部最优解。

2.7 拓展阅读

2.7.1 L1 正则化回归

L1 正则化回归 (L1 Regularization regression)，也称 Lasso 回归，是最小绝对值收敛和选择算子 (Least absolute shrinkage and selection operator) 回归的简称。它在普通的线性回归模型上与岭回归类似地构造一个惩罚函数使得能够压缩权重系数。岭回归按不同比例压缩每个权重系数但始终保留每个权重系数，而 L1 正则化回归在惩罚系数比较大的时候能够使部分权重系数精确地缩减为 0，即不保留部分特征。而 L1 正则化回归的这种思想在数据时代的今天是很有意义的，很多时候我们并不缺少数据，反而是数据过多，真正有用的数据比较少。在成百上千个变量中，起决定作用的数据只有有限的几个。而 L1 正则化回归的具体操作类似岭回归，对损失函数的增加了一个惩罚项，公式如下。

$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\|\mathbf{w}\|_1 \quad (2.52)$$

$$\|\mathbf{w}\|_1 = \sum_{i=1}^{d+1} |w_i| \quad (2.53)$$

其中 λ 是惩罚系数， $\|\mathbf{w}\|_1$ 是 \mathbf{w} 的 L1 范数，等于列向量 \mathbf{w} 的每个分量的绝对值之和。而这也揭示了何为 L1 正则化。同理，岭回归也叫 L2 正则化回归。

下面，我们略微介绍一下该损失函数推导。我们知道普通线性回归需要优化的问题如下

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) \quad (2.54)$$

而 L1 正则化回归对此优化问题改进成如下形式

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) \quad \text{s. t. } \|\mathbf{w}\|_1 \leq t \quad (2.55)$$

其中 t 是一个不依赖于 \mathbf{w} 的常数， $t > 0$ 。这是一个非线性的规划问题，带有不等式约束。利用拉格朗日乘数法 (Lagrange multiplier) 及 KKT 条件 (Karush-Kuhn-Tucker Conditions)，我们可以把该优化问题等价成无约束形式，如下

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda\|\mathbf{w}\|_1 - \lambda t \quad (2.56)$$

其中 $\lambda > 0$ ，和 t 一样不依赖于 \mathbf{w} 。末尾的常数项不影响优化过程，可以去掉。故

优化问题最后化为

$$\min_{\mathbf{w}} \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda \|\mathbf{w}\|_1 \quad (2.57)$$

这就略微解释了公式2.52的由来，而在岭回归部分为了讲述的连贯性，没有涉及到这部分内容。至于拉格朗日乘数法和KKT条件的数学过程略为高深，篇幅有限，就不予展示了，有兴趣的读者可以自行查阅。

L1正则化回归的求解方法有多种，该部分不是本文重点，有兴趣的读者可以查阅参考文献。

2.7.2 多项式回归

多项式回归（Polynomial Regression），回归函数是特征变量多项式的回归。在现实生活中，很多变量与变量的关系不是线性的，例如在匀加速运动中，位移与时间的关系。而普通线性回归的局限性在于假设变量间是线性关系，即使尽可能去拟合，但对于本身是非线性关系的变量间，回归的效果并不好。根据微积分的知识，我们知道连续函数可以用多项式去逼近，故多项式回归还是有着较为广泛的应用。下面我们通过一元函数的拟合例子来解释多项式回归，假设因变量和自变量的数据如图2.1所示。

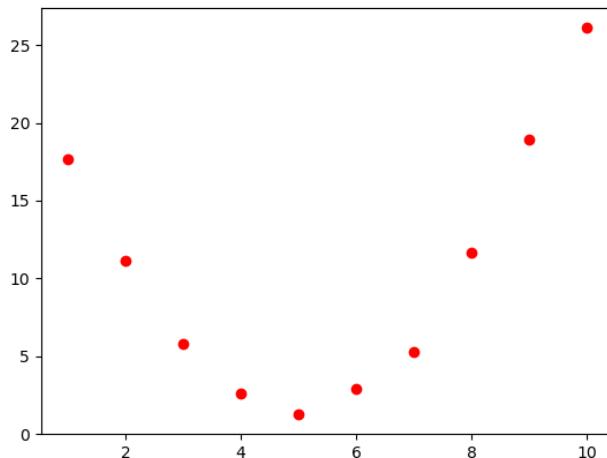


图 2.1: 因变量与自变量数据

如果我们使用，普通的线性回归对这些数据进行拟合，得到的拟合图像如图2.2所示。

我们直接的观察都可以很容易看出，自变量和因变量之间没什么线性关系，不妨先假设他们之间存在二次关系。即假设一个线性模型如下

$$f(x) = w_1x + w_2x^2 + b \quad (2.58)$$

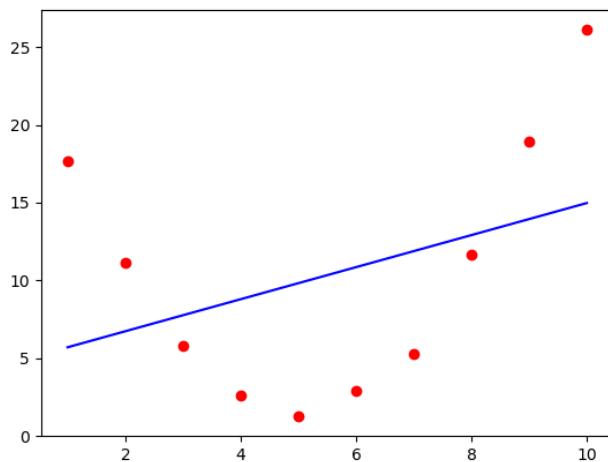


图 2.2: 普通线性回归拟合图像

而当我们把 x^2 当成另一个新的特征变量时，这本质上就可以看成是线性回归了。

$$\mathbf{x} = (1, x, x^2)^T \quad (2.59)$$

$$\mathbf{w} = (b, w_1, w_2)^T \quad (2.60)$$

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} \quad (2.61)$$

如此这般，该多项式回归的拟合图像如图2.3所示。

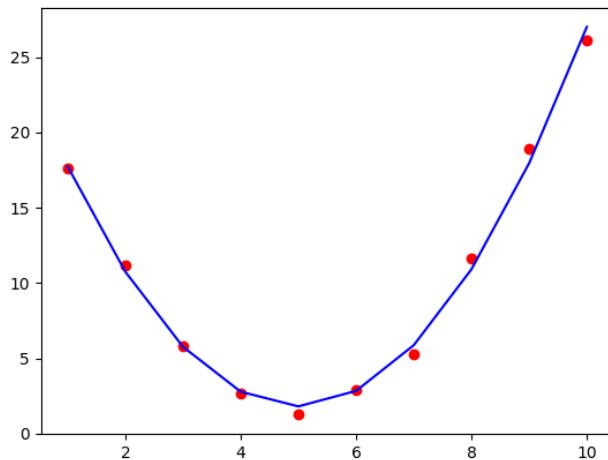


图 2.3: 多项式回归拟合图像

我们可以看到拟合的效果比起普通的线性回归好很多。那么对于多个特征变

量如何进行多项式回归呢，我们用二元二次多项式回归作为例子。

$$\mathbf{x} = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2)^\top \quad (2.62)$$

$$\mathbf{w} = (b, w_1, w_2, w_3, w_4, w_5)^\top \quad (2.63)$$

$$\begin{aligned} f(\mathbf{x}) &= w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2 \\ &= \mathbf{w}^\top \mathbf{x} \end{aligned} \quad (2.64)$$

更高次项的多项式回归如此类推，但可以发现随着特征变量的个数、最高次项增加，“真实”的特征向量的数量急剧增加，计算量剧增。而且太高次的多项式回归会导致过拟合问题，详细的相关内容读者可自行查阅后续章节。

2.7.3 损失函数的概率解释

在本小节中，我们首先会给出一些概率的基本假设，然后基于这些假设推出选择平方损失函数作为线性回归的损失函数是一个很自然的事情。首先我们假设训练集 \mathcal{T} 每个样本和线性模型间存在这样的关系。

$$y_i = f(\mathbf{x}_i) + \epsilon_i = \mathbf{w}^\top \mathbf{x}_i + \epsilon_i \quad (2.65)$$

这里的 ϵ_i 是误差项，误差项存在是因为模型本身的缺陷（例如 \mathbf{x} 与 y 存在很明显的非线性关系），或者是噪音。

这个误差项应该是对结果有微小影响的众多独立随机变量叠加产生的，中心极限定理指出这种误差项近似服从正态分布。故假设该误差项是独立同分布的，且服从均值为 0，方差为 σ^2 的正态分布，即

$$\epsilon_i \sim \mathcal{N}(0, \sigma^2) \quad (2.66)$$

那么这样就可以写出 ϵ_i 的概率密度函数为

$$p(\epsilon_i) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{\epsilon_i^2}{2\sigma^2}\right) \quad (2.67)$$

因 $\epsilon_i = y_i - \mathbf{w}^\top \mathbf{x}_i$ ，假设 y_i 为一个随机变量，故

$$p(y_i|\mathbf{x}_i; \mathbf{w}) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \quad (2.68)$$

这里的 $p(y_i|\mathbf{x}_i; \mathbf{w})$ 表示为对于给定输入为 \mathbf{x}_i 时输出正好等于 y_i 的条件概率密度函数，其中 \mathbf{w} 为该分布的参数。

我们知道线性回归的目标是让全部样本的预测值与真实值之间的差异尽可

能小。从概率的角度来说，就是在参数 \mathbf{w} 未知的情况下，给定所有样本的 \mathbf{x}_i ，让对应的 y_i 同时发生的概率最大，即概率积最大。而这个概率积就是 \mathbf{w} 在训练集 \mathcal{T} 下的似然函数

$$\mathbb{L}(\mathbf{w}) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \quad (2.69)$$

为了区分损失函数的符号 \mathcal{L} ，上式用了 \mathbb{L} 表示似然函数。

在得出似然函数的形式后，怎么样得到 \mathbf{w} 的最佳估计呢？这里就要用到最大似然估计，它告诉我们， \mathbf{w} 的最佳估计能让似然函数 $\mathbb{L}(\mathbf{w})$ 取得最大值。我们可以看到 $\mathbb{L}(\mathbf{w})$ 的函数结构比较复杂，且容易发生数值下溢。而我们对其取对数后，函数结构较为简单且单调性不变。所以我们可以用对数似然函数 $\ln \mathbb{L}(\mathbf{w})$ 来代替似然函数 $\mathbb{L}(\mathbf{w})$ 。

$$\begin{aligned} \ln \mathbb{L}(\mathbf{w}) &= \ln \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^n \ln \left(\frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y_i - \mathbf{w}^\top \mathbf{x}_i)^2}{2\sigma^2}\right) \right) \\ &= n \ln \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \cdot \frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \end{aligned} \quad (2.70)$$

而我们对 \mathbf{w} 的估计不依赖于 σ ，故为了让 $\ln \mathbb{L}(\mathbf{w})$ 取得最大值，也就意味着要让 $\frac{1}{2} \sum_{i=1}^n (y_i - \mathbf{w}^\top \mathbf{x}_i)^2$ 取得最小值，而这也正是平方损失函数 $\mathcal{L}(\mathbf{w})$ 本身。所以，线性回归选取平方损失函数作为损失函数是一种很自然的做法。

2.7.4 广义线性模型

通过2.7.3小节的介绍我们可以知道，线性回归的线性模型本质等同于假设训练集 \mathcal{T} 中的每个样本都符合正态分布，即

$$y_i | \mathbf{x}_i; \mathbf{w} \sim \mathcal{N}(\mathbf{w}^\top \mathbf{x}_i, \sigma^2) \quad (2.71)$$

在不引起歧义的情况下，下面我们用 \mathbf{x}, y 泛指样本的特征向量和标签值。我们知道，正态分布只是一个假设，现实中很多事物的分布并不符合正态分布。并且标签值 $y_i \in (-\infty, \infty)$ ，难以扩展成其他形式，通过这样假设得到的线性模型，有较大的局限性。下面我们介绍广义线性模型，来降低局限性。在广义线性模型中，定义 y 符合指数族分布（Exponential family distributions），即

$$y | \mathbf{x}; \mathbf{w} \sim \text{ExponentialFamily}(\eta) \quad (2.72)$$

其中 ExponentialFamily 即是指数族分布，泛指正态分布、泊松分布、指数分布、伯努利分布等分布。 η 是该分布的自然参数 (Natural parameter)，且广义线性模型假设 $\eta = \mathbf{w}^\top \mathbf{x}$ ，说是假设，倒不如用设计二字描述更为准确。指数族的通用概率分布表示如下

$$p(y|\eta) = b(y) \exp(\eta T(y) - a(\eta)) \quad (2.73)$$

其中 $T(y)$ 叫做充分统计量 (Sufficient statistic)，通常情况下 $T(y) = y$ 。而 $a(\eta)$ 是对数分割函数 (Log partition function)。 $\exp(-a(\eta))$ 扮演了归一化常数 (Normalization constant) 的作用，保证 $p(y|\eta)$ 求和或者积分为 1。 $b(\cdot)$ 只是代表指数外与 y 相关的函数。

那么在我们知道指数族分布之后，如何构造具体的广义线性模型呢？我们知道构造线性模型是想构造 $f(\mathbf{x})$ 尽可能地去预测 y ，从概率的角度来说就是， $f(\mathbf{x})$ 即为 y 分布的期望。下面我们用正态分布作为例子，去构造广义线性模型，而这也就是普通的线性模型。

$$y|\eta \sim \mathcal{N}(\mu, \sigma^2) \quad (2.74)$$

其中 μ 为均值，即期望。 σ^2 为方差。那么可以把该概率分布变换为通用的指数族概率分布形式，即

$$\begin{aligned} p(y|\eta) &= \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(-\frac{y^2}{2\sigma^2}\right) \cdot \exp\left(\frac{\mu}{\sigma^2}y - \frac{\mu^2}{2\sigma^2}\right) \end{aligned} \quad (2.75)$$

那么我们很容易看出 $\eta = \frac{\mu}{\sigma^2}$ ，那么我们定义一个 μ 到 η 的链接函数 (Link function) $g(\mu)$ 如下

$$\begin{aligned} g(\mu) &= \frac{\mu}{\sigma^2} \\ &= \eta \end{aligned} \quad (2.76)$$

那么这个线性模型 $f(\mathbf{x})$ 的构造就呼之欲出了。

$$\begin{aligned}
 f(\mathbf{x}) &= E(y|\eta) \\
 &= \mu \\
 &= g^{-1}(\eta) \\
 &= \sigma^2 \eta \\
 &= \sigma^2 \mathbf{w}^\top \mathbf{x}
 \end{aligned} \tag{2.77}$$

其中 E 表示这个分布的期望函数， $g^{-1}(\cdot)$ 为响应函数 (Response function)，是 $g(\cdot)$ 的反函数。

对于给定 \mathbf{x} ，认为 σ^2 是常量，完全可以由 \mathbf{w} 体现。故该线性模型最后为

$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} \tag{2.78}$$

同理，其他指数族分布同样如此构造广义线性模型，如通过伯努利分布 (Bernoulli distribution) 构造的广义线性模型，就是 logistic 回归的模型，对具体构造推导感兴趣的读者可以自行查阅相关文献。而损失函数的构造主要通过极大似然法，在2.7.3小节中，我们有详细介绍，不再赘述。

2.8 本章总结

在本章中，我们介绍了线性回归及相关内容。首先我们提出了基本的线性模型 $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$ ，以及线性回归的目的。为了达到线性回归的目的，我们提出了损失函数 $\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w})$ 。为了最小化损失函数，可以得到线性回归的闭式解 $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$ 。为了解决 $\mathbf{X}^\top \mathbf{X}$ 可能不可逆等问题，我们介绍了岭回归。在岭回归中，主要是把损失函数转换成 $\mathcal{L}(\mathbf{w}) = \frac{1}{2}(\mathbf{y} - \mathbf{X}\mathbf{w})^\top(\mathbf{y} - \mathbf{X}\mathbf{w}) + \frac{1}{2}\lambda \mathbf{w}^\top \mathbf{w}$ ，相应地，闭式解为 $\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$ 。最后，我们在拓展阅读中介绍了 L1 正则化回归、多项式回归、损失函数的概率解释以及广义线性模型。对线性回归尚存疑问的读者，可以查阅本章末尾的参考文献。

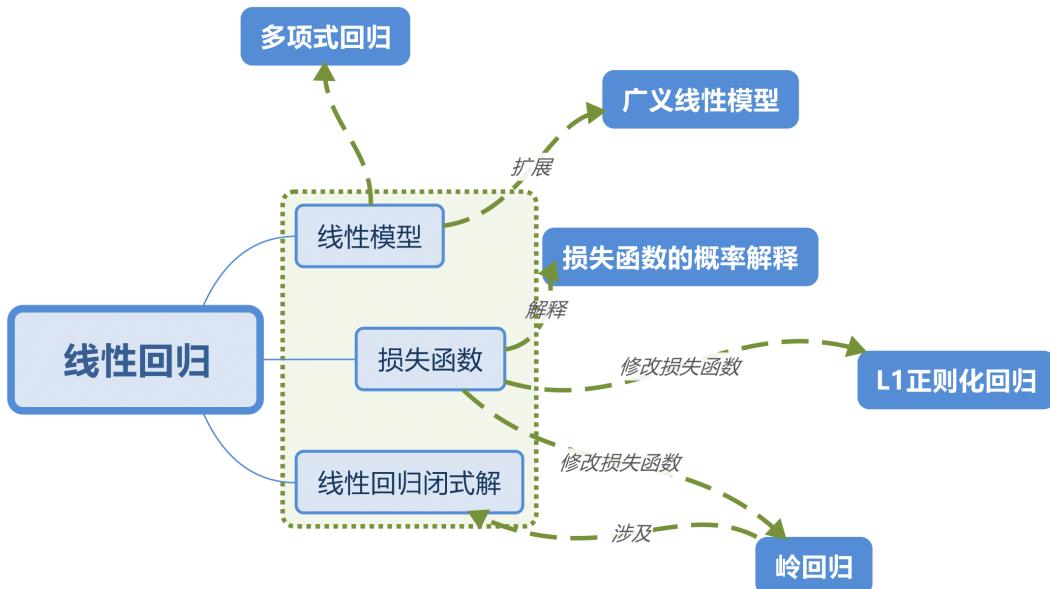


图 2.4: 线性回归知识框架

高斯与最小二乘法

18 世纪末，天文学的发展以测量和观测为基础。1794 年，高斯利用了最小二乘法解决了多余观测问题，当时他只有 17 岁。他通过实验得到两组数据，目的是用一个简单的式子来表示这些数据之间的关系。在数据分析的过程中，他首先把这些点绘制在二维平面中，发现这些点差不多在一条直线上，因此自然想到用一条直线，也就是一个线性函数来表达这些数之间的关系，求解的过程相当于求方程组中的待定系数。高斯想到，这个拟合过程毕竟是估计，不是真实的。要是用来做预测的话，必然存在一定的误差，这个误差如果能控制在相对小范围内的话，那么结果相对就好一些。所以，他找到了我们现在广为应用的残差平方和取最小值的方法，得到了待定系数的最小二乘估计。

在 1801 年，意大利著名天文学家朱赛普皮亚齐发现了第一颗小行星——谷神星。经过 40 多天的持续观察后，发现谷神星运转到太阳的背后，使得皮亚齐无法观测到谷神星的位置。在此之后，全世界的科学家利用皮亚齐的观测数据开始寻找谷神星，但是利用大多数人计算的结果来寻找谷神星，都没有找到。时年 24 岁的高斯也计算了谷神星的位置和运行轨道函数。奥地利天文学家海因里希奥尔伯斯根据高斯计算出来的轨道结果重新发现了谷神星。这一神奇的计算结果，得到了全世

界的瞩目。

高斯使用的最小二乘法的方法，最早发表于1809年他的著作《天体运动论》中。法国科学家勒让德于1806年独立发现“最小二乘法”。但因名气太小而不为人所知。勒让德曾与高斯为“谁最早创立最小二乘法原理”发生争执。根据普拉克特记载，普通最小二乘是勒让德“确定彗星轨道的新方法”。阿比意力挺高斯，声称高斯是1795年发明的，认为勒让德是在1806年首次发表的。最终在1829年，高斯提供了最小二乘法的优化效果强于其他方法的证明，因此被称为“高斯—莫卡夫”定理。最小二乘法的基本概念区别于以往任何的求得极值的方法，它的运用是相对灵活的。所用的具体的极小化方法可以是十分复杂的，要利用非线性规划的算法；也可以是相对地简单，只要用中学生的计算水平就可以解决了。所以最小二乘法在实践中得以广泛使用。

参考文献

- [1] Gene H Golub, Per Christian Hansen, and Dianne P O'Leary. Tikhonov regularization and total least squares. *SIAM Journal on Matrix Analysis and Applications*, 21(1):185–194, 1999.
- [2] Peter McCullagh. *Generalized linear models*. Routledge, 2018.
- [3] John Ashworth Nelder and Robert WM Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A (General)*, 135(3):370–384, 1972.
- [4] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 329. John Wiley & Sons, 2012.
- [5] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society: Series B (Methodological)*, 58(1):267–288, 1996.
- [6] 周志华. 机器学习. Qing hua da xue chu ban she, 2016.
- [7] 邱锡鹏. 神经网络与深度学习.
- [8] 霍建新. 高斯的“大手笔”. 中国统计, (5):57–58, 2010.

第3章 线性分类、多分类



图 3.1: 章节结构

引言 1 选课的烦恼 终于，小明到了大二了，终于不在只能学学校安排的专业选修课了，他可以开始体验到自己掌握自己该学什么课了，可以根据自己的兴趣选择自己想上的课程，但是也不能随便选择，因为可能这门课虽然很感兴趣，但是并不适合小明学，他可能会在这门课上栽跟头，无法拿到理想的分数，甚至挂科，这会极大的损害小明的自信心，所以小明希望能够有一个工具帮助他判断他想选的这门课程是否适合他。

另外，大二下学期了，小明也面临着选择专业方向的苦恼，学院有着几个不同的方向：智能软件、数字媒体，嵌入式系统、移动开发、数据挖掘，选择了不同的方向，就必须把这个方向的选修课全修了才能顺利毕业，尽管辅导员请了各个专业的代表老师来给他们做各个专业方向的特点，课程，和主要培养目标，他还是有点摸不准他到底适合哪一个方向，所以希望能够有一个工具能够给他推荐一个专业方向作为参考。

而上述问题实际可以看作我们本章要讲的二分类问题和多分类问题，都可以利用线性分类模型解决。相信学完这一章你自己也能写一个工具来帮助小明选课了。

3.1 线性分类模型

线性模型是机器学习中最为简单的一个数学模型，我们知道线性方程形如： $y = 2x + 5$ ，其任意变量都是 1 次幂，其表现在坐标轴图上则为一条直线，线性模型实际上就是一个线性组合函数，不仅限于 2 维数据，在高维数据上，其图像表现为一个线性超平面。

给定一个 d 维样本 $[x_1, \dots, x_d]^T$, 这里的样本特征可以是小明之前所学 d 门课程的成绩, 其线性组合函数为:

$$f(\mathbf{x}, \mathbf{w}) = w_1x_1 + w_2x_2 + \dots + w_dx_d + b \quad (3.1)$$

$$= \mathbf{w}^T \mathbf{x} + b \quad (3.2)$$

其中 $\mathbf{w} = [w_1, \dots, w_d]^T$ 为 d 维的权重向量, w_i 表示第 i 门课程对结果的影响力大小, b 为偏置。在分类问题中, 由于输出目标 y 是一些离散的标签, 如“是、否”合适小明, 智能软件、数字媒体, 嵌入式系统、移动开发、数据挖掘”, 而 $f(\mathbf{x}, \mathbf{w})$ 的值域为实数, 因此无法直接用 $f(\mathbf{x}, \mathbf{w})$ 来进行预测, 需要引入一个非线性的决策函数 (decision function) $g(\cdot)$ 来预测输出目标, 将 $f(\mathbf{x}, \mathbf{w})$ 的值映射到离散空间上:

$$y = g(f(\mathbf{x}, \mathbf{w})) \quad (3.3)$$

其中 $f(\mathbf{x}, \mathbf{w})$ 也称为判别函数 (discriminant function)。

对于二分类问题, $g(\cdot)$ 可以是符号函数 (sign function):

$$g(f(\mathbf{x}, \mathbf{w})) = \text{sgn}(f(\mathbf{x}, \mathbf{w})) \quad (3.4)$$

$$\triangleq \begin{cases} +1 & \text{if } f(\mathbf{x}, \mathbf{w}) > 0 \\ -1 & \text{if } f(\mathbf{x}, \mathbf{w}) < 0 \end{cases} \quad (3.5)$$

$f(\mathbf{x}, \mathbf{w}) = 0$ 时不进行预测。公式3.5定义了一个典型的两类分类问题的决策函数, 其结构如图3.2所示。

在本章, 我们主要介绍两种不同线性分类模型: 感知器和支持向量机, 这些模型区别主要在于使用了不同的损失函数。

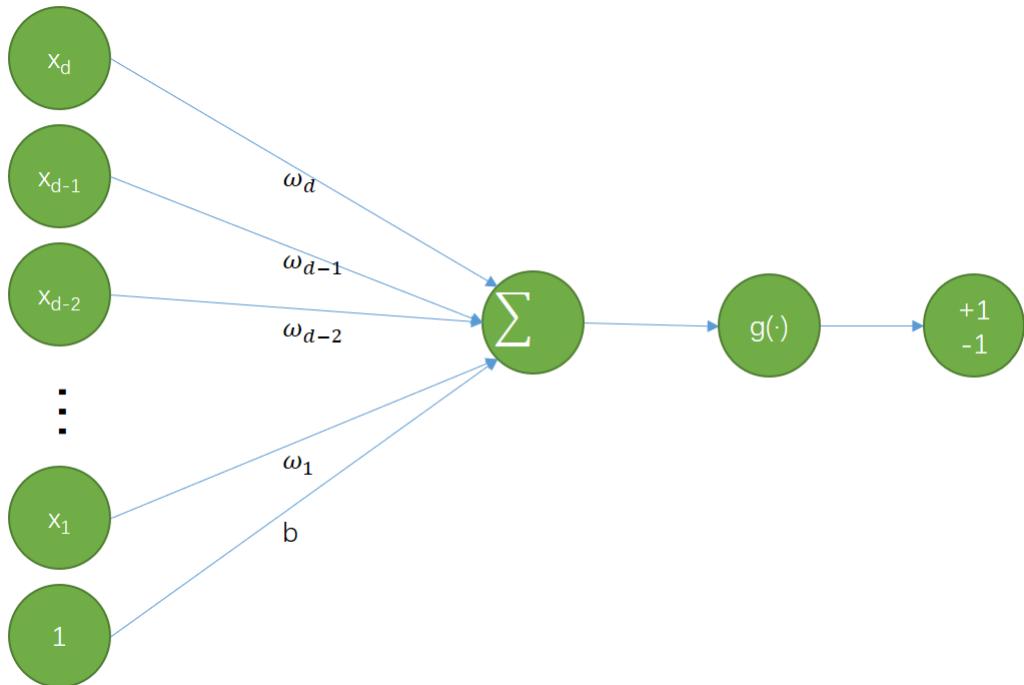


图 3.2: 二分类问题决策函数结构

3.2 线性判别函数和决策边界

通过上部分的介绍，我们知道了一个线性分类模型（Linear Classification Model）或线性分类器（Linear Classifier），是由一个（或多个）线性的判别函数 $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + b$ 和非线性的决策函数 $g()$ 组成。

3.3 二分类

二分类（Binary Classification）的类别标签 y 只有两种取值，通常可以设为 $\{+1, -1\}$ ，这两个标签可以在现实问题中代表各种不同的意义，如可以在我们的问题中， $+1$ 代表这门课适合小明， -1 则代表不适合，或者反过来。在二分类中，我们只需要一个线性判别函数 $f(\mathbf{x}, \mathbf{w}) = \mathbf{w}^T \mathbf{x} + b$ 。特征空间 \mathbb{R}^d 中所有满足 $f(\mathbf{x}, \mathbf{w}) = 0$ 的点组成用一个分割超平面（hyperplane），称为决策边界（decision boundary）或决策平面（decision surface）。决策边界将特征空间一分为二，划分成两个区域，每个区域对应一个类别。

所谓“线性分类模型”就是指其决策边界是线性超平面。在特征空间中，决策平面与权重向量 \mathbf{w} 正交。特征空间中每个样本点到决策平面的有向距离（signed distance）为：

$$\gamma = \frac{f(\mathbf{x}, \mathbf{w})}{\|\mathbf{w}\|} \quad (3.6)$$

γ 也可以看作是点 \mathbf{x} 在 \mathbf{w} 方向上的投影。

图3.3给出了一个二维数据的线性决策边界示例，其中样本特征向量是二维的， $\mathbf{x} = [x_1, x_2]$ ，权重向量对应的也是二维的 $\mathbf{w} = [w_1, w_2]$ ，给定 N 个样本的训练

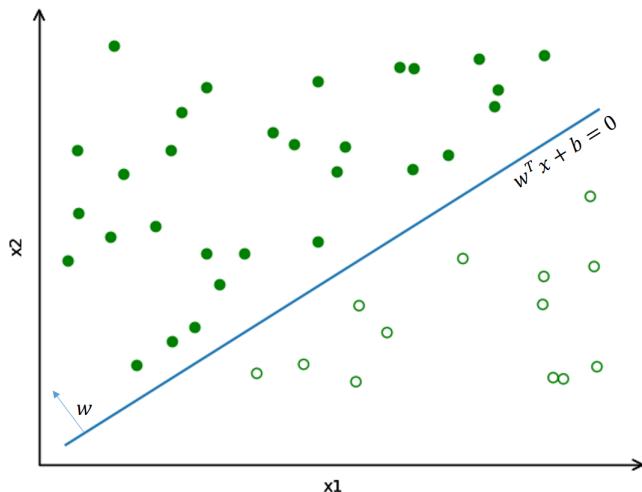


图 3.3: 二维数据线性决策边界

集 $\mathcal{D} = (\mathbf{x}^{(i)}, y^{(i)})_{i=1}^N$ ，其中 $y^{(i)} \in \{+1, -1\}$ ，线性模型试图学习到权重参数 \mathbf{w}^* ，使得对于每个样本 $(\mathbf{x}^{(i)}, y^{(i)})$ 尽量满足：

$$\begin{aligned} f(\mathbf{x}^{(i)}, \mathbf{w}^*) &> 0 \quad \text{if } y^{(i)} = 1, \\ f(\mathbf{x}^{(i)}, \mathbf{w}^*) &< 0 \quad \text{if } y^{(i)} = -1. \end{aligned} \quad (3.7)$$

上面公式3.7也可以合并，即参数 \mathbf{w}^* 尽量满足：

$$y^{(i)} f(\mathbf{x}^{(i)}, \mathbf{w}^*) > 0, \quad \forall i \in [1, N] \quad (3.8)$$

于是我们有了定义1

定义 1 两类线性可分 对于训练集 $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$ ，如果存在权重向量 \mathbf{w} ，对所有样本都满足 $y f(\mathbf{x}, \mathbf{w}) > 0$ ，那么训练集 \mathcal{D} 是线性可分的。

为了学习参数 \mathbf{w} ，我们需要定义合适的损失函数以及优化方法。对于二分类问题，最直接的损失函数为 0-1 损失函数，即：

$$\mathcal{L}_{01}(y, f(\mathbf{x}, \mathbf{w})) = I(y f(\mathbf{x}, \mathbf{w}) > 0) = \begin{cases} 1, & y f(\mathbf{x}, \mathbf{w}) > 0 \\ 0, & y f(\mathbf{x}, \mathbf{w}) \leq 0 \end{cases} \quad (3.9)$$

其中 $I()$ 为指示函数。但 0-1 损失函数的数学性质不好，其关于 \mathbf{w} 的导数为 0，从而导致无法优化 \mathbf{w} 。

3.3.1 支持向量机 (SVM)

支持向量机 (Support Vector Machine, SVM)¹是一个经典两类分类算法，在1964年由Vapnik和Alexey Y. Chervonenkis [10]提出，其找到的分割超平面具有更好的鲁棒性，因此广泛使用在很多任务上，并表现出了很强优势，因此可以用于解决我们的问题。

给定一个两类分类器数据集 $\mathcal{D} = (\mathbf{x}^{(i)}, y^{(i)})_{i=1}^N$ ，其中 $y^{(i)} \in \{+1, -1\}$ ，其中 $\mathbf{x}^{(i)}$ 表示第 i 个人的以往课程成绩组成的向量， $y^{(i)}$ 表示这个人这些课程成绩是否满意，如果两类样本是线性可分的，即存在一个超平面：

$$\mathbf{w}^T \mathbf{x} + b = 0 \quad (3.10)$$

将两类样本分开，那么对于每个样本都有 $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) > 0$ 。将 \mathbf{w} 规范化后数据集 \mathcal{D} 中每个样本 $\mathbf{x}^{(i)}$ 到分割超平面的距离为：

$$\gamma^{(i)} = \frac{\|\mathbf{w}^T \mathbf{x}^{(i)} + b\|}{\|\mathbf{w}\|} = \frac{y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} + b}{\|\mathbf{w}\|} \quad (3.11)$$

我们定义整个数据集 \mathcal{D} 中所有样本中到分割超平面的距离最短的那个点的距离为间隔 (Margin) γ

$$\gamma = \min_i \gamma^{(i)} \quad (3.12)$$

如果间隔 γ 越大，其分割超平面对两个数据集的划分越稳定，不容易受噪声点等因素影响。支持向量机的目标是寻找一个超平面 (\mathbf{w}, b) 使得 γ 最大，即：

$$\begin{aligned} & \max_{\mathbf{w}, b} \gamma \\ & \text{s.t. } \frac{y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)}{\|\mathbf{w}\|} \geq \gamma, \forall i \end{aligned} \quad (3.13)$$

显然各个样本中落在平行于分割超平面的边界的点就是我们要找的 γ ，我们假设这些点刚好落在 $\mathbf{w}^T \mathbf{x}^{(i)} + b = \pm 1$ 上，则公式3.13等价于

$$\begin{aligned} & \max_{\mathbf{w}, b} \frac{1}{\|\mathbf{w}\|} \\ & \text{s.t. } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 1, \forall i \end{aligned} \quad (3.14)$$

数据集中所有满足 $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) = 1$ 的样本点，也就是落在两个分类的样本点平行于切割面 $\mathbf{w}^T \mathbf{x} + b = 0$ 的边缘平面上的边界点都称为支持向量 (Support-Vector)，刚好在这些点的两边的都是属于他们同一类别的点。

对于一个线性可分的数据集，其分割超平面有很多个，但是间隔最大的超平

¹具体可参考 <https://zh.wikipedia.org/zh-hans/%E6%94%AF%E6%8C%81%E5%90%91%E9%87%8F%E6%9C%BA>

面是唯一的。图3.4给定了支持向量机的最大间隔分割超平面的示例，其蓝色样本点为支持向量。

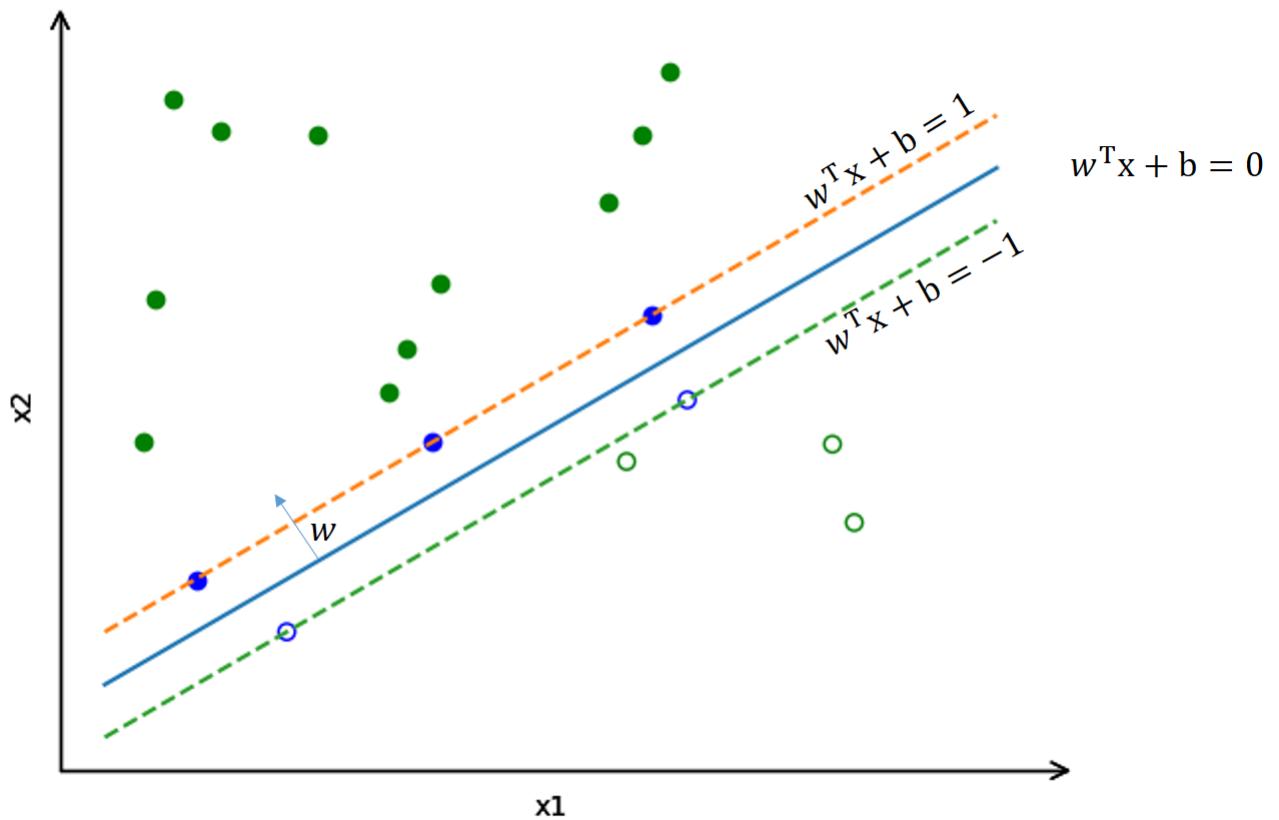


图 3.4: 最大间隔分割超平面

3.3.1.1 支持向量机参数优化

为了便于解决问题和参数优化，将公式3.14 的目标函数写为凸优化问题：

$$\begin{aligned} \min_{\mathbf{w}, b} \quad & \frac{1}{2} \|\mathbf{w}\|^2 \\ \text{s.t.} \quad & 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \leq 0, \quad \forall i \end{aligned} \tag{3.15}$$

带条件的优化问题，我们很难进行统一公式优化，所以我们需要使用拉格朗日乘数法以将其转换为不带条件的优化方程，公式3.15的拉格朗日函数为：

$$L(\mathbf{w}, b, \lambda) = \frac{1}{2} \|\mathbf{w}\|^2 + \sum_{i=1}^N \lambda_i (1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b)) \tag{3.16}$$

其中 $\lambda_1 \geq 0, \dots, \lambda_N \geq 0$ 为拉格朗日乘数。计算 $\Gamma(\mathbf{w}, b, \lambda)$ 关于 \mathbf{w} 和 b 的导数：

$$\begin{aligned}\frac{\partial \Gamma(\mathbf{w}, b, \lambda)}{\partial \mathbf{w}} &= \mathbf{w} - \sum_{i=1}^N \lambda_i y^{(i)} \mathbf{x}^{(i)} \\ \frac{\partial \Gamma(\mathbf{w}, b, \lambda)}{\partial b} &= \sum_{i=1}^N \lambda_i y^{(i)}\end{aligned}\tag{3.17}$$

并使其等于 0, 得：

$$\mathbf{w} = \sum_{i=1}^N \lambda_i y^{(i)} \mathbf{x}^{(i)},\tag{3.18}$$

$$0 = \sum_{i=1}^N \lambda_i y^{(i)}\tag{3.19}$$

将公式3.18 代入公式3.16, 并利用公式3.19, 得到拉格朗日对偶函数：

$$\Gamma(\lambda) = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_j \lambda_i y^{(j)} y^{(i)} (\mathbf{x}^{(j)})^T \mathbf{x}^{(i)} + \sum_{i=1}^N \lambda_i.\tag{3.20}$$

支持向量机的优化问题主要是凸优化问题，满足强对偶性，即优化问题可以通过最大化对偶函数 $\max_{\lambda \geq 0} \Gamma(\lambda)$ 来求解。对偶函数 $\Gamma(\lambda)$ 是一个凹函数，因此最大化对偶函数是一个凸优化问题，可以通过多种凸优化方法来进行求解，得到拉格朗日乘数的最优值 λ 。但由于其约束条件的数量为训练样本数量，一般的优化方法代价比较高，因此在实践中通常采用比较高效的优化方法，比如由微软研究院的 John Platt 在 1998 年提出的序列最小优化算法（Sequential minimal optimization, SMO）² [6] 等。

根据 KKT 条件³中的互补松弛条件，最优解满足 $\lambda_i^* (1 y^{(i)} (\mathbf{w}^{*T} \mathbf{x}^{(i)} + b^*)) = 0$ 。如果样本 $x^{(i)}$ 不在约束边界上， $\lambda_i^* = 0$ ，其约束失效；如果样本 $\mathbf{x}^{(i)}$ 在约束边界上， $\lambda_i^* \geq 0$ 。这些在约束边界上样本点称为支持向量（support vector），即离决策平面距离最近的点。

在计算出 λ^* 后，根据公式3.18 计算出最优权重 \mathbf{w}^* ，最优偏置 b^* 可以通过任选一个支持向量 (\tilde{x}, \tilde{y}) 计算得到。

$$b^* = \tilde{y} - \mathbf{w}^{*T} \tilde{\mathbf{x}}\tag{3.21}$$

²具体可参考 <https://pdfs.semanticscholar.org/59ee/e096b49d66f39891eb88a6c84cc89acba12d.pdf>

³具体参考 https://en.wikipedia.org/wiki/Karush%20-%20Kuhn%20-%20Tucker_conditions

最优参数的支持向量机的决策函数为：

$$f(\mathbf{x}) = \operatorname{sgn}(\mathbf{w}^*{}^T \mathbf{x} + b^*) \quad (3.22)$$

$$= \operatorname{sgn}\left(\sum_{i=1}^N \lambda_i^* y^{(i)} (\mathbf{x}^{(i)})^T \mathbf{x} + b^*\right) \quad (3.23)$$

支持向量机的决策函数只依赖 $\lambda_i^* > 0$ 的样本点，即支持向量。

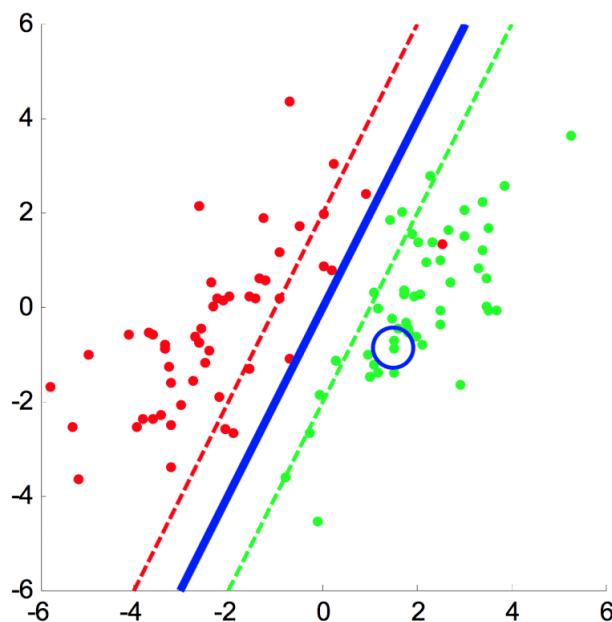
支持向量机的目标函数可以通过 SMO 等优化方法得到全局最优解，因此比其它分类器的学习效率更高。此外，支持向量机的决策函数只依赖与支持向量，与训练样本总数无关，分类速度比较快。

Q1: 不是所有的数据集都是严格线性可分的，可能会存在少许噪声点，造成的原因可能是数据集的标签值在标记时出了错，如将机器学习的课程推荐数据集中，一个分数高的同学，被数据标记员在标记时标记成了 -1，即不推荐，这样上面的向量机怎么也不能找到最大间隔的分割平面，可是其数据集又明显是线性可分问题，显然上述模型不足以解决这个问题，我们需要放松一下条件使得向量机能够支持分类这样的数据集。

3.3.1.2 软间隔

在上节的支持向量机的优化问题中，约束条件比较严格。如果训练集中的样本在特征空间中不是严格线性可分的话，就无法找到最优解。如图3.5

图 3.5：无法严格线性可分的情况



为了能够容忍部分不满足线性约束的样本，我们可以引入松弛变量 ξ ，将优

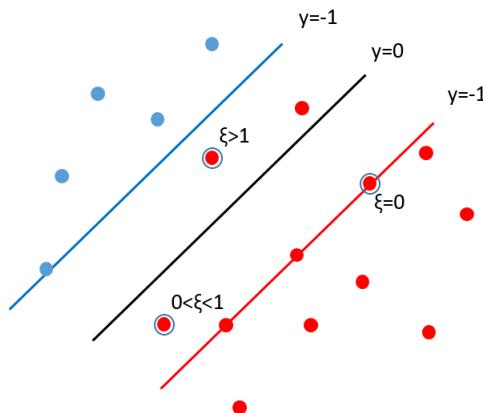
化问题变为：

$$\begin{aligned} \min_{w,b} \quad & \frac{1}{2} \|w\|^2 + C \sum_{i=1}^N \xi_i \\ \text{s.t.} \quad & 1 - y^{(i)}(w^T x^{(i)} + b) - \xi_i \leq 0, \quad \forall i \\ & \xi_i \geq 0, \quad \forall i \end{aligned} \quad (3.24)$$

ξ_i 可以理解为第 i 个样本点离其正确类别点的边界的距离，如图3.6所示：

1. 如果 $\xi_i = 0$ 则其刚好落在边界点，即其是支持向量，则分对了。
2. 如果 $0 < \xi_i < 1$ 则其在分割面的正确的一边，只是没在边界内，也能正确分类。
3. 如果 $\xi_i > 1$ 则其分在了分割面的另一个类别，明显是错误分类了。

图 3.6: ξ_i 的现实意义



其中参数 $C > 0$ 用来控制间隔和松弛变量惩罚的平衡。引入松弛变量的间隔称为软间隔（soft margin）[8]。公式 3.25 也可以表示为无条件函数：

$$\min_{w,b} L(w,b) = \sum_{i=1}^N \max(0, 1 - y^{(i)}(w^T x^{(i)} + b)) + \frac{1}{C} \cdot \frac{1}{2} \|w\|^2, \quad (3.25)$$

其中 $\max(0, 1 - y^{(i)}(w^T x^{(i)} + b))$ 称为 hinge 损失函数，如图3.7所示，其图像就像是一个铰链(hinge)， $\frac{1}{C}$ 可以看作是正则化系数。 $\frac{1}{C}$ 越大，则越关注模型复杂度，这样得到的模型复杂度低， $\frac{1}{C}$ 越小，则越关注间隔大小，可能会导致模型复杂度过高，出现过拟合现象(过拟合将会在第 6 章和欠拟合一同介绍)。软间隔支持向量机的参数学习和原始支持向量机类似，其最终决策函数也只和支持向量有关，即

满足 $1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) - \xi_i = 0$ 的样本。我们对公式3.25求导：

$$\frac{\partial L(\mathbf{w}, b)}{\partial \mathbf{w}} = \frac{1}{C} \|\mathbf{w}\| + \sum_{i=1}^N g_{\mathbf{w}}(\mathbf{x}_i) \quad (3.26)$$

$$g_{\mathbf{w}}(\mathbf{x}_i) = \begin{cases} -y_i \mathbf{x}_i & 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 0 \\ 0 & 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) < 0 \end{cases}$$

$$\frac{\partial L(\mathbf{w}, b)}{\partial b} = g_b(\mathbf{x}_i) = \begin{cases} -y_i & 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq 0 \\ 0 & 1 - y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) < 0 \end{cases} \quad (3.27)$$

Q2: 那如果数据集不是线性可分数据集，而是在当前的特征空间是非线性的，但是鉴于 SVM 的高效率高性能，那能不能继续用 SVM 算法进行分类呢？实际上是可以的，在当前特征空间不满足线性可分的数据，我们可以利用核技巧将其映射到线性可分的空间上，再用 SVM 进行求解。感兴趣的同学可以自行参阅“更多”小节的核技巧 [3] 部分的参考文献。

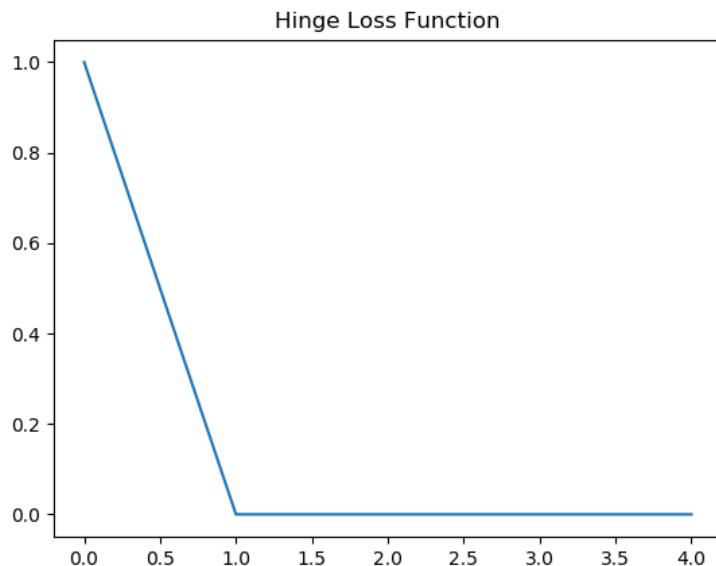


图 3.7: Hinge Loss 图像

3.3.2 感知器

感知器（Perceptron）⁴由 Frank Rosenblatt 于 1957 年 [7] 提出，是一种广泛使用的线性分类器。感知器可谓是最简单的人工神经网络，只有一个神经元。

感知器是对生物神经元的简单数学模拟，有与生物神经元相对应的部件，如权重（突触）、偏置（阈值）及激活函数（细胞体），输出为 +1 或 -1。

⁴具体可参考 <https://zh.wikipedia.org/wiki/%E6%84%9F%E7%9F%A5%E5%99%A8>

感知器是一种简单的两类线性分类模型，其分类准则与公式3.5相同。

$$\hat{y} = \text{sgn}(\mathbf{w}^T \mathbf{x}). \quad (3.28)$$

3.3.2.1 感知器参数优化

有 N 个样本的训练集: $(\mathbf{x}^{(i)}, y^{(i)})_i^N = 1$, 其中 $y^{(i)} \in \{+1, -1\}$, 感知器试图更新并学习到参数 \mathbf{w} , 使得对于每个样本 $(x^{(i)}, y^{(i)})$ 有

$$y^{(i)} \mathbf{w}^T \mathbf{x}^{(i)} > 0, \quad \forall i \in [1, N]. \quad (3.29)$$

感知器的学习算法是一种错误驱动的学习算法。先初始化一个权重向量 $\mathbf{w} \leftarrow 0$ (全零向量或者随机向量), 如果分错了一个样本 (\mathbf{x}, y) 时, 即 $y\mathbf{w}^T \mathbf{x} < 0$, 就用这个样本来更新权重。 $F(\mathbf{w}^T, \mathbf{x}) = y\mathbf{w}^T \mathbf{x}$ 关于 \mathbf{w}^T 的求导很简单, 为:

$$\frac{\partial F(\mathbf{w}^T, \mathbf{x})}{\partial \mathbf{w}^T} = y\mathbf{x} \quad (3.30)$$

更新权重:

$$\mathbf{w} \leftarrow \mathbf{w} - y\mathbf{x} \quad (3.31)$$

具体的感知器参数学习策略如算法2所示。

Algorithm 2: 两类感知算法

Input: 训练集 $\{\{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N\}$, 迭代 T 次

1 初始化: $\mathbf{w}_0 \leftarrow 0, k \leftarrow 0$;

2 **for** $t = 1 \rightarrow T$ **do**

3 随机对训练样本进行随机排序;

4 **for** $i = 1 \rightarrow N$ **do**

5 选取一个样本 $(\mathbf{x}^{(i)}, y^{(i)})$; 用公式3.19计算预测类别 $\hat{y}^{(i)}$;

6 **if** $\hat{y}^{(i)} \neq y^{(i)}$ **then**

7 $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + (\phi(\mathbf{x}^{(i)}, y^{(i)}) - \phi(\mathbf{x}^{(i)}, \hat{y}^{(i)}))$; $k = k + 1$;

8 **end**

9 **end**

10 **end**

Output: \mathbf{w}_k

实际上, 感知器的损失函数也很简单:

$$\mathcal{L}(\mathbf{w}; \mathbf{x}, y) = \max(0, -y\mathbf{w}^T \mathbf{x}). \quad (3.32)$$

图??给出了感知器参数学习的更新过程, 其中红色实心点为正例, 蓝色空心

点为负例。黑色箭头表示权重向量，红色虚线箭头表示权重的更新方向。

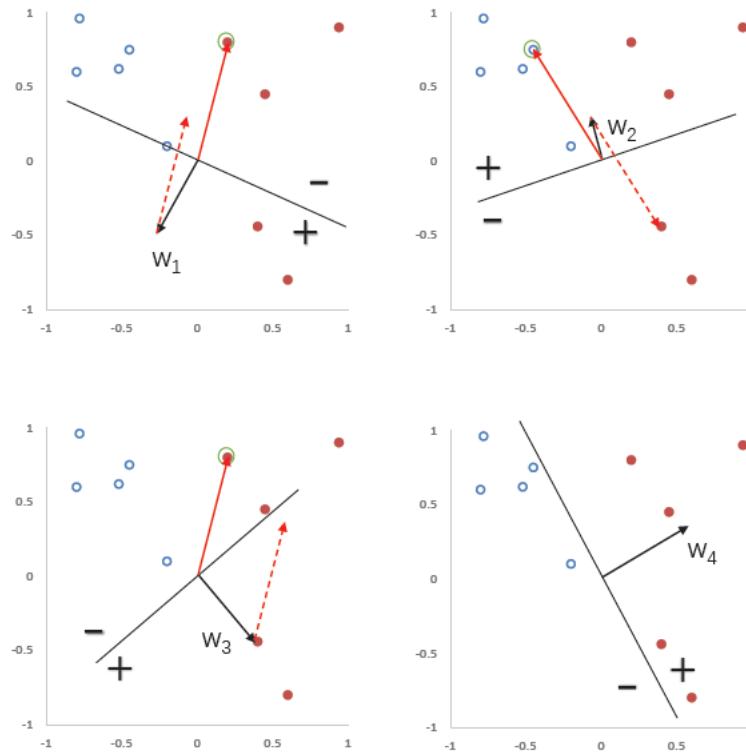


图 3.8: 感知器参数学习更新过程

3.3.2.2 感知器的收敛性

Novikoff 于 1963 证明了对于二分类问题，如果训练集是线性可分的，那么感知器算法可以在有限次迭代后收敛；如果训练集不是线性可分的，那么这个算法则不一定会收敛 [5]。

当数据集是二类线性可分时，对于训练集 $\mathcal{D} = (\mathbf{x}^{(i)}, y^{(i)})_{i=1}^N$ ，其中 $\mathbf{x}^{(i)}$ 为样本的增广特征向量， $y^{(i)} \in \{1, -1\}$ ，则必定存在一个权重向量也就是参数 \mathbf{w}^* ，并且 $\|\mathbf{w}^*\| = 1$ ，对所有 i 都有 $(\mathbf{w}^*)^T (\mathbf{x}^{(i)}) \geq 0$ ，那么也必定可以找到一个正的常数 $\gamma (\gamma > 0)$ 和权重向量，对所有 i 都满足 $(\mathbf{w}^*)^T (\mathbf{x}^{(i)}) \geq \gamma$ 。我们可以证明如下定理。

定理 1 感知器的收敛性 给定一个训练集 $\mathcal{D} = (\mathbf{x}^{(n)}, y^{(n)})_{n=1}^N$ ，假设 R 是训练集中最大的特征向量的模，

$$R = \max_n \|\mathbf{x}^{(n)}\|. \quad (3.33)$$

如果训练集 \mathcal{D} 线性可分，感知器学习算法 I 的权重更新次数不超过 $\frac{R^2}{\gamma^2}$ 。

证明：

感知器的权重向量的更新方式为：

$$\mathbf{w}_k = \mathbf{w}_{k-1} + y^{(k)} \mathbf{x}^{(k)}, \quad (3.34)$$

其中 $\mathbf{x}^{(k)}, y^{(k)}$ 表示第 k 个错误分类的样本。因为初始权重向量为 0，在第 K 次更新时感知器的权重向量为：

$$\mathbf{w}_K = \sum_{k=1}^K y^{(k)} \mathbf{x}^{(k)}. \quad (3.35)$$

计算 $\|\mathbf{w}_K\|^2$ 的上下界：

上界为：

$$\|\mathbf{w}_K\|^2 = \|\mathbf{w}_{K-1} + y^{(K)} \mathbf{x}^{(K)}\|^2 \quad (3.36)$$

$$= \|\mathbf{w}_{K-1}\|^2 + \|y^{(K)} \mathbf{x}^{(K)}\|^2 + 2y^{(K)} \mathbf{w}_{K-1}^T \mathbf{x}^{(K)} \quad (3.37)$$

$$\leq \|\mathbf{w}_{K-1}\|^2 + R^2 \quad (3.38)$$

$$\leq \|\mathbf{w}_{K-2}\|^2 + 2R^2 \quad (3.39)$$

$$\leq KR^2 \quad (3.40)$$

下界为：

$$\|\mathbf{w}_K\|^2 = \|\mathbf{w}^*\|^2 \cdot \|\mathbf{w}_K\|^2 \quad (3.41)$$

$$\geq \|\mathbf{w}^{*T} \mathbf{w}_K\|^2 \quad (3.42)$$

$$= \|\mathbf{w}^{*T} \sum_{k=1}^K (y^{(k)} \mathbf{x}^{(k)})\|^2 \quad (3.43)$$

$$= \left\| \sum_{k=1}^K \mathbf{w}^{*T} (y^{(k)} \mathbf{x}^{(k)}) \right\|^2 \quad \because \|\mathbf{w}^*\| = 1 \quad (3.44)$$

$$\geq K^2 \gamma^2 \quad (3.45)$$

由3.40与3.45得到：

$$K^2 \gamma^2 \leq \|\mathbf{w}_K\|^2 \leq KR^2 \quad (3.46)$$

则有 $K^2 \gamma^2 \leq KR^2$, 变形得：

$$K \leq \frac{R^2}{\gamma^2} \quad (3.47)$$

证明在线性可分的条件下，算法2会在 $\frac{R^2}{\gamma^2}$ 步内收敛

虽然感知器在线性可分的数据上可以保证收敛，但其存在以下不足之处：

1. 在数据集线性可分时，感知器虽然可以找到一个超平面把两类数据分开，但并不能保证其泛化能力。
2. 感知器对样本顺序比较敏感。每次迭代的顺序不一致时，找到的分割超平面也往往不一致。
3. Freund 和 Schapire 于 1999 年证明 [2]，如果训练集不是线性可分的，就永远不会收敛⁵

3.3.3 参数平均感知器

如果训练数据是线性可分的，那么感知器是可以找到一个判别函数来分割不同类的数据，而且间隔 γ 越大，收敛越快。但是不能保证找到的判别函数就是最好的，有可能会导致过拟合。

上一节末提到，感知器的一个缺点就是对样本顺序很敏感，具体表现在迭代次序上排在后面的错误样本，比前面的错误样本对最终的权重向量影响更大。比如有 1,000 个训练样本，在学习 100 个样本后，感知器已经学习到一个很好的权重向量。在接下来的 899 个样本上都预测正确，也没有更新权重向量。但是在最后第 1,000 个样本时预测错误，权重发生了更新。那么这次更新可能会使本来预测正确率大的权重向量往差的方向偏转或移动。

Freund and Schapire 在 1999 年提出可以使用“参数平均”的策略来改善这种情况，从而提高感知器的性能，称为投票感知器（Voted Perceptron）[2]。

投票感知器记录第 k 次更新后得到的权重 \mathbf{w}_k 在之后的训练过程中正确分类样本的次数 c_k 。这样最后的分类器形式为：

$$\hat{y} = \operatorname{sgn}\left(\sum_{k=1}^K c_k \operatorname{sgn}(\mathbf{w}_k^T \mathbf{x})\right) \quad (3.48)$$

$\operatorname{sgn}(\cdot)$ 为符号函数。

投票感知器虽然提高了感知器的泛化能力，但是需要保存 K 个权重向量及他们所对应的正确分类的次数。这样会造成很大的内存占用问题，因此，Collins 在 2002 年提出了可以使用一个简化的版本，也叫做平均感知器（Averaged Perceptron）

⁵ 其证明发表于《Large Margin Classification Using the Perceptron Algorithm》
<https://link.springer.com/content/pdf/10.1023%2FA%3A1007662407062.pdf>

[1]⁶

$$\hat{y} = \operatorname{sgn}\left(\sum_{k=1}^K c_k (\mathbf{w}_k^T \mathbf{x})\right) \quad (3.49)$$

$$= \operatorname{sgn}\left(\left(\sum_{k=1}^K c_k \mathbf{w}_k\right)^T \mathbf{x}\right) \quad (3.50)$$

$$= \operatorname{sgn}(\bar{\mathbf{w}}^T \mathbf{x}) \quad (3.51)$$

其中 $\bar{\mathbf{w}}$ 为平均的权重向量。

假设 $\mathbf{w}_{t,i}$ 是在第 t 轮更新到第 i 个样本时权重向量的值，平均的权重向量 $\bar{\mathbf{w}}$ 也可以写为

$$\bar{\mathbf{w}} = \frac{\sum_{t=1}^T \sum_{i=1}^n \mathbf{w}_{t,i}}{iT} \quad (3.52)$$

这个方法非常简单，只需要在算法 3.1 中增加一个 $\bar{\mathbf{w}}$ ，并且在处理每一个样本后，更新 $\bar{\mathbf{w}}$:

$$\bar{\mathbf{w}} \leftarrow \bar{\mathbf{w}} + \mathbf{w}_{t,i} \quad (3.53)$$

但这个方法需要在处理每一个样本时都要对 $\bar{\mathbf{w}}$ 进行更新，而且 $\bar{\mathbf{w}}$ 和 $\mathbf{w}_{t,i}$ 都是稠密向量，因此更新会比较费时。为了提高迭代速度，提出了很多改进的方法，让这个更新只需要在错误预测发生时才进行更新。Daumé III 提出了一个改进的平均感知器算法的训练过程。如算法3所示

Q3: 那如果像推荐选择专业方向这种不是只有两个类别的怎么办呢？这个问题是可以用下一节的多分类方法解决的。

3.4 多分类

多类分类 (Multi-class Classification) 问题是指分类的类别数 C 大于 2。如给小明推荐专业方向，就有 5 个类。多类分类一般需要多个线性判别函数，但设计这些判别函数有很多种方式。

假设一个多类分类问题的类别为 $\{1, 2, \dots, C\}$ ，常用的方式有以下三种：

1. "一对其余" 方式 (one VS rest): 把多类分类问题转换为 C 个 "一对其余" 的两类分类问题。这种方式共需要 C 个判别函数，其中第 c 个判别函数 f_c 是将类 c 的样本和不属于类 c 的样本分开。
2. "一对一" 方式：把多类分类问题转换为 $C(C - 1)/2$ 个 "一对一" 的两类分类问题。这种方式共需要 $C(C - 1)/2$ 个判别函数，其中第 (i, j) 个判别函数是把类 i 和类 j 的样本分开。

⁶其算法发表于《New Ranking Algorithms for Parsing and Tagging: Kernels over Discrete Structures, and the Voted Perceptron》
<https://pdfs.semanticscholar.org/fe63/8b5610475d4524684fb2c2b7b08c119c8700.pdf>

Algorithm 3: 平均感知器算法

Input: 训练集 $\{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, 最大迭代次数 T

- 1 初始化: $\mathbf{w} \leftarrow 0, \mathbf{u} \leftarrow 0, c \leftarrow 0$;
- 2 **for** $t = 1 \rightarrow T$ **do**
- 3 随机对训练样本进行随机排序;
- 4 **for** $i = 1 \rightarrow N$ **do**
- 5 选取一个样本 $(\mathbf{x}^{(i)}, y^{(i)})$;
- 6 计算预测类别 \hat{y}_t ;
- 7 **if** $\hat{y}_t \neq y_t$ **then**
- 8 $\mathbf{w} \leftarrow \mathbf{w} + y^{(i)}\mathbf{x}^{(i)}$;
- 9 $\mathbf{u} \leftarrow \mathbf{u} + cy^{(i)}\mathbf{x}^{(i)}$;
- 10 **end**
- 11 $c \leftarrow c + 1$;
- 12 **end**
- 13 **end**
- 14 $\bar{\mathbf{w}} = \mathbf{w}_T - \frac{1}{c}\mathbf{u}$;

Output: $\bar{\mathbf{w}}$

3. "argmax" 方式: 这是一种改进的“一对其余”方式, 共需要 C 个判别函数:

$$f_c(\mathbf{x}, \mathbf{w}_c) = \mathbf{w}_c^T \mathbf{x} + b_c, c = [1, \dots, C] \quad (3.54)$$

如果存在类别 c , 对于所有的其他类别 $\bar{c} (\bar{c} \neq c)$ 都满足 $f_c(\mathbf{x}, \mathbf{w}_c) > f_{\bar{c}}(\mathbf{x}, \mathbf{w}_{\bar{c}})$, 那么 \mathbf{x} 属于类别 c 。即

$$y = \operatorname{argmax}_{c=1}^C f_c(\mathbf{x}, \mathbf{w}_c) \quad (3.55)$$

“一对其余”方式需要训练的模型相对“一对一”要少很多, 只需要训练 C 个, 但是其存在负类样本数量远远多于正类, 导致训练样本不平衡, 同时没加入一个新的类, 就需要对所有的模型重新训练一次。

而“一对一”方式最突出的问题是需要训练的模型数量太多, 代价太大。

在“argmax”方式中, 相邻两类 i 和 j 的决策边界实际上是由 $f_i(\mathbf{x}, \mathbf{w}_i) - f_j(\mathbf{x}, \mathbf{w}_j) = 0$ 决定, 其法向量为 $\mathbf{w}_i - \mathbf{w}_j$ 。

定义 2 多类线性可分 对于训练集 $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^N$, 如果存在 C 个权重向量 $\mathbf{w}_c, 1 \leq c \leq C$, 对所有第 c 类的样本都满足 $f_c(\mathbf{x}, \mathbf{w}_c) > f_{\bar{c}}(\mathbf{x}, \mathbf{w}_{\bar{c}}), \forall \bar{c} \neq c$, 那么训练集 \mathcal{D} 是线性可分的。

从上面定义2可以看出, 如果数据集可以多类线性可分的, 那么一定存在一个“argmax”方式的线性分类器可以将它们正确分开。

3.4.1 感知器用于多分类

之前介绍的分类模型中，分类函数都是在输入 \mathbf{x} 的特征空间上。为了使得感知器可以处理更复杂的输出，我们引入一个构建输入输出联合空间上的特征函数 $\phi(\mathbf{x}, \mathbf{y})$ ，将样本 (\mathbf{x}, \mathbf{y}) 对映射到一个特征向量空间。

在联合特征空间中，我们可以建立一个多分类的感知器模型：

$$\hat{y} = \operatorname{argmax}_{y \in Gen(\mathbf{x})} \mathbf{w}^T \phi(\mathbf{x}, \mathbf{y}) \quad (3.56)$$

其中 \mathbf{w} 为权重向量， $Gen(\mathbf{x})$ 表示输入 \mathbf{x} 所有的输出目标集合，如智能软件，数字媒体，嵌入式系统，移动开发，数据挖掘。当处理 C 类分类问题时， $Gen(\mathbf{x}) = \{1, \dots, C\}$ 。

在 C 类分类中，一种常用的特征函数 $\phi(\mathbf{x}, \mathbf{y})$ 是 \mathbf{y} 和 \mathbf{x} 的外积，其中 \mathbf{y} 为类别的 one-hot 向量表示。

$$\phi(\mathbf{x}, \mathbf{y}) = \operatorname{vec}(\mathbf{y}\mathbf{x}^T) \in \mathbb{R}^{(d \times C)} \quad (3.57)$$

vec 为向量化算子。给定样本 (\mathbf{x}, \mathbf{y}) ，若 $\mathbf{x} \in \mathbb{R}^d$ ， \mathbf{y} 为第 c 维为 1 的 one-hot 向量，则

$$\phi(\mathbf{x}, \mathbf{y}) = \begin{bmatrix} \vdots \\ 0 \\ x_1 \\ \vdots \\ x_d \\ 0 \\ \vdots \end{bmatrix}$$

其中 $[x_1, x_2, \dots, x_d]$ 所在的行是第 $(c - 1) \times d + 1 \rightarrow (c - 1) \times d + d$ 行。

多分类感知器算法训练过程如算法4所示：

3.4.2 多分类感知器的收敛性

多分类感知器在满足多类线性可分条件时，也能够保证在有限步骤内收敛。多类线性可分条件如下：

定义 3 多类线性可分 对于训练集 $\mathcal{D} = (\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^N$ ，如果存在一个正的常数 $\gamma (\gamma > 0)$ 和权重向量 \mathbf{w} ，并且 $\|\mathbf{w}\| = 1$ ，对所有 i 都满足 $\mathbf{w}, \phi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \mathbf{w}, \phi(\mathbf{x}^{(i)}, \mathbf{y}) \geq \gamma, \mathbf{y} \neq \mathbf{y}^{(i)}$ ($\phi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) \in \mathbb{R}^d$ 为样本 $\mathbf{x}^{(i)}, \mathbf{y}^{(i)}$ 的联合特征向量)，那么训练集 \mathcal{D} 在联合特征向量空间中是多类线性可分的。

多分类感知器的收敛性定理如下：

Algorithm 4: 多分类感知器参数学习算法

Input: 训练集: $\{(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})\}_{i=1}^N$, 最大迭代次数 T

- 1 初始化: $\mathbf{w}_0 \leftarrow 0, k \leftarrow 0;$
- 2 **for** $t = 1 \rightarrow T$ **do**
- 3 随机对训练样本进行随机排序;
- 4 **for** $i = 1 \rightarrow N$ **do**
- 5 选取一个样本 $(\mathbf{x}^{(i)}, \mathbf{y}^{(i)})$;
- 6 用公式3.56计算预测类别 $\hat{\mathbf{y}}^{(i)}$;
- 7 **if** $\hat{\mathbf{y}}^{(i)} \neq \mathbf{y}^{(i)}$ **then**
- 8 $\mathbf{w}_{k+1} \leftarrow \mathbf{w}_k + (\phi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \phi(\mathbf{x}^{(i)}, \hat{\mathbf{y}}^{(i)}))$;
- 9 $k = k + 1$;
- 10 **end**
- 11 **end**
- 12 **end**

Output: \mathbf{w}_k

定理 2 多分类感知器收敛性 对于满足多类线性可分的训练集 $\mathcal{D} = (\mathbf{x}^{(i)}, \mathbf{y}^{(i)})_{i=1}^N$, 假设 R 是所有样本中真实标签和错误标签在特征空间 $\phi(\mathbf{x}, \mathbf{y})$ 最远的距离。

$$R = \max_i \max_{z \neq \mathbf{y}^{(i)}} \|\phi(\mathbf{x}^{(i)}, \mathbf{y}^{(i)}) - \phi(\mathbf{x}^{(i)}, z)\|.$$

那么多分类感知器参数学习算法4的权重更新次数不超过 $\frac{R^2}{\gamma^2}$ 。

3.5 更多

· 核函数:

《Kernel Methods for Pattern Analysis》 Shawe-Taylor, J.; Cristianini, N. (2004). Cambridge University Press.

《Comparing support vector machines with Gaussian kernels to radial basis function classifiers》 Bernhard Schölkopf, Kah Kay Sung, Christopher J. C. Burges, Federico Girosi, Partha Niyogi, Tomaso A. Poggio, Vladimir Vapnik(1997)

《Kernel methods in machine learning》 Thomas Hofmann, Bernhard Scholkopf, Alexander J. Smola (2008)

3.6 章节总结

通过本章的学习，我们主要学习了感知器和支持向量机用于线性分类，我们希望你掌握以下内容以便顺利解决小明的烦恼：

1. 线性分类模型包含两个主要部分：线性判别函数 $f(\mathbf{x}, \mathbf{w})$ 和用于映射到离散空间的非线性决策函数 $g(\cdot)$ ，在二分类中可以是符号函数；
2. 对于二类线性分类问题，我们希望可以找到一个权重向量 \mathbf{w}^* ，使得所有样本都满足 $yf(\mathbf{x}, \mathbf{w}) > 0$
3. 对于多分类线性问题，我们希望可以找到 C 个权重向量 $\mathbf{w}_c, 1 \leq c \leq C$ ，使得所有第 c 类的样本都满足 $f_c(\mathbf{x}, \mathbf{w}_c) > f_{\bar{c}}(\mathbf{x}, \mathbf{w}_{\bar{c}}), \forall \bar{c} \neq c$ ；
4. 多分类问题一般有“一对其余”，“一对一”，“argmax”三种模型设计方式。凡是多类线性可分的都可以应用“argmax”方法；
5. 数据集中所有满足 $y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) = 1$ 的样本点，也就是落在两个分类的样本点平行于切割面 $\mathbf{w}^T \mathbf{x} + b = 0$ 的边缘平面上的边界点都称为支持向量（SupportVector）；
6. 为了解决噪声点造成的非严格线性可分问题，我们引入了软间隔，给以模型适当的容错率，同时使用了 hinge loss；
7. 为了解决当前特征空间的线性不可分问题，支持向量机可以应用核函数以映射到适合的特征空间使其称为线性可分问题。
8. 感知器是最简单的神经网络，只有一个神经元，它只针对分错的样本进行权重更新；
9. 为了避免后面的错误样本对权重影响过大的问题，提出了“参数平均”的策略，也叫投票感知器；之后为了解决保存 K 个权重向量带来的额外开销问题，使用了简化版本，即平均感知器；
10. 对于线性可分问题，无论是二分类，还是多分类，感知器的算法收敛步数都是有限的，且不超过 $\frac{R^2}{\gamma}$ 。

参考文献

- [1] Michael Collins and Nigel Duffy. New ranking algorithms for parsing and tagging: Kernels over discrete structures, and the voted perceptron. In *ACL*, 2002.
- [2] Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296, 1998.
- [3] Anthony Widjaja Lin. Learning with kernels: Support vector machines, regularization, optimization, and beyond. *IEEE Transactions on Neural Networks*, 16:781–781, 2003.
- [4] Ryan T. McDonald, Keith B. Hall, and Gideon Mann. Distributed training strategies for the structured perceptron. In *HLT-NAACL*, 2010.
- [5] Albert B Novikoff. On convergence proofs for perceptrons. 1963.
- [6] John C. Platt. Fast training of support vector machines using sequential minimal optimization. 1999.
- [7] Frank F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.
- [8] Fred W. Smith. Pattern classifier design by linear programming. *IEEE Transactions on Computers*, C-17:367–372, 1968.

- [9] Johan A. K. Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural Processing Letters*, 9:293–300, 1999.
 - [10] Vladimir Vapnik and Alexei Ya. Chervonenkis. A note on one class of perceptrons. 1964.
 - [11] 邱锡鹏. 神经网络与深度学习, 2019.
-

第4章 逻辑回归

监督学习分为“回归问题”和“分类问题”两大类。具体而言，回归模型用来预测连续的值，比如给定过去的天气属性，预测明天的温度，降水量等；而分类问题用来预测离散的值，比如说给定一张图片，预测它的类别，是猫还是狗等。最常遇到的分类问题为线性可分问题和线性不可分问题，如图 4.1

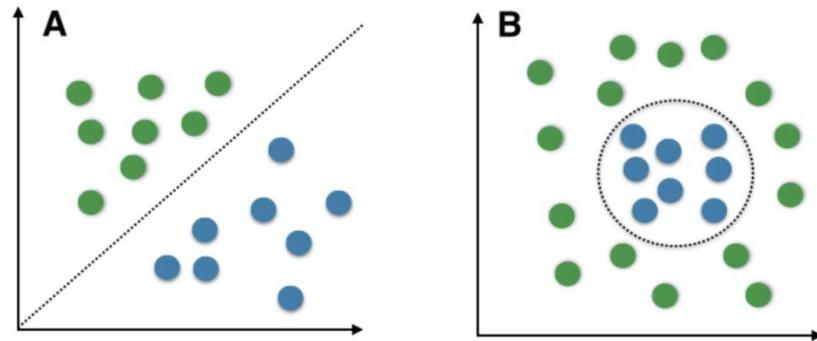


图 4.1：左图为在二维平面下（即输入向量有两个特征值），正类和负类样本间找得到一个线性边界，使得正负类样本完全区分，这个边界也叫做线性决策边界。右图为一个线性不可分的情况，这种情况下决策边界更为复杂。

正如在此书前被“机器学习的奥秘”吓得瑟瑟发抖的你一样，小明同样经历着这样的一个思维斗争过程：究竟要不要选机器学习，毕竟挂科率很高！正如大多数搞机器学习的人一样，他喜欢以大多数人过往的行为以及结果，来估计自己的结果。

如果像他这样成绩平平的师兄师姐，通过了机器学习的考核，那他也有信心通过。反之，如果大多数像他这样的人挂了，那他就不选。当然这只是一个定性的想法，作为双一流高校的理科生，看问题当然要用量化数据，于是他立即向上一届师兄师姐要了一些成绩和机器学习考试结果。他选了两个认为与机器学习考试结果有直接关系的课程成绩作为输入 $\vec{x} = \{x_1, x_2\}$ ，数学成绩和编程成绩，每一个学生的结果，也就是输出标记为 $y = \{1 \text{ 如果通过} 0 \text{ 如果没有通过}\}$ 。他总共收集了 50 个以往学生的成绩，即 $X = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_{50}\}$ ，以及他们的结果 $Y = \{y_1, y_2, \dots, y_{50}\}$ 。下面就让我们来看看如何根据这些信息帮助他做选择。

4.1 简介

4.1.0.0.1 逻辑回归 (Logistic Regression) 是一种线性模型，它既没有逻辑，也不是用作回归。上一章节中我们学过线性回归，理论上来说我们用线性模型来回

归到分类变量 0 和 1 也不是不可以的。究竟是数学成绩 x_1 还是编程水平 x_2 对机器学习通过与否更重要呢，我们可以像线性回归一样分别赋予这两个成绩一个权重 w_1, w_2 以及一个偏置 b ，用我们的数据输入来 X 来学习确定这些参数的大小，得到我们的模型。也就是：

$$y = w_1x_1 + w_2x_2 + b \quad (4.1)$$

写成矩阵形式：

$$y = w^T \vec{x} + b \quad (4.2)$$

其中, $x = \{x_1; x_2\}$, $w = \{w_1; w_2\}$

然而问题的关键在于这个分类问题难以优化。机器学习很多时候都可以归纳为优化问题，优化问题是在训练样本使得总体误差小，而机器学习则要考虑对未来样本的准确率，从而加入损失优化的正则项¹。

在线性回归的章节中，我们定义了损失函数，也就是一个刻画预测出来值和真实值的差距的表达式，如最小化均方差（欧几里得距离），我们当然希望这个差距越小越好。总体训练样本差距小，说明我们的模型比较准确，也就是说对于新的样本预测的值要准确。对于分类问题，我们用什么刻画预测与实际的误差，从而能够让我们最小化误差来更新参数呢？

4.2 逻辑函数

4.2.0.0.1 处理二分类问题时，当然也可以还按照之前线性回归的算法根据给定的样本 x 来预测 y ，但这会忽略掉 y 是一个散列值，容易遇到性能问题，运行效率低且效果差。而逻辑回归与线性回归模型基本相同，也是计算各个特征向量的权重和，不同的是，线性回归直接返回预测结果，逻辑回归则返回结果的概率。与此同时，我们已经确定了 $y \in \{0, 1\}$ ，也就是 y 必须是 0 或者 1。此时可以改变一下假说函数的形式来解决这个问题：

$$h_w(x) = g(\sum_{i=1}^m w_i x_i) = g(w^T x) \quad (4.3)$$

4.2.0.0.2 在上面提到的例子中 x_i 分别代表病人的年龄，性别，血糖。 w_i 是我们需要处理的参数，也可以看作某因素的重要性，他们乘积的和将是病人是否会患上心脏病的重要判断依据， $h_w(x)$ 就是我们通过预测得到的结果。

¹任何损害优化的过程都可以之为正则化

4.2.0.0.3 假设函数中的 $g(z) = \frac{1}{1+e^{-z}}$ 叫做逻辑函数，或者也叫双弯曲 S 型函数 (sigmoid function)，模型的正负概率分别为

$$p(y=1|x; w) = \frac{e^{w^\top x + b}}{1 + e^{w^\top x + b}} \quad (4.4)$$

$$p(y=0|x; w) = \frac{1}{1 + e^{w^\top x + b}} \quad (4.5)$$

4.2.0.0.4 下图是 sigmoid 函数图像

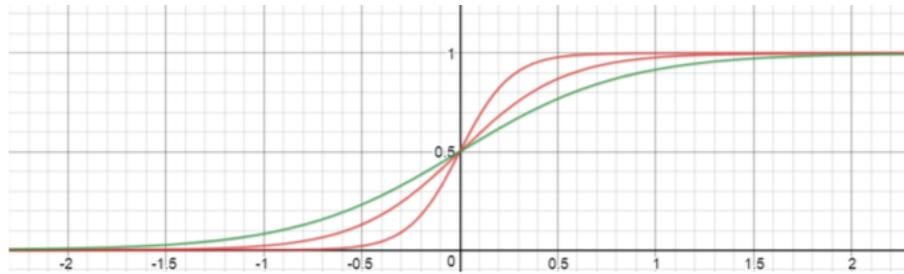


图 4.2: Sigmoid Function

4.2.0.0.5 由图可看出当 $z \rightarrow +\infty$ 时， $g(z) \rightarrow 1$ ，而当 $z \rightarrow -\infty$ 时， $g(z) \rightarrow 0$ 。此外，这个函数是连续的且一直在 0 和 1 之间波动，将 $z = w^\top x + b$ 代入 sigmoid 函数可得回归模型的参数形式。考虑到二分类任务，我们需将预测值 z 值转换为 0 或者 1，分别代表不会得心脏病和会得心脏病，最理想的是“单位阶跃函数”(unit-step function)

$$y = \begin{cases} 0, & z < 0 \\ 0.5, & z = 0 \\ 1, & z > 0 \end{cases} \quad (4.6)$$

4.2.0.0.6 即若预测值 z 大于零就判为正例（会得心脏病），小于零则判为反例（不会得心脏病），预测值为临界值零则可任意判别。若将 y 视为样本作为正例的可能性，那么 $1 - y$ 就是其作为反例的可能性，两者的比值

$$\frac{y}{1 - y} \quad (4.7)$$

4.2.0.0.7 称为“几率”(odds)反映了样本作为正例的相对可能性，对几率取对数则得到“对数几率”，此时代入 y 可得

$$\ln \frac{y}{1 - y} = w^\top x + b \quad (4.8)$$

4.3 代价函数

4.3.0.0.1 那么参数 w_i 又该如何选择呢？这就用到我们的代价函数来监督 w_i 。总的来说，任何能够测量模型预测值与真实值之间的差异的函数都能叫做代价函数 (Cost Function)。当我们确定了模型 h ，后一步要做的就是训练模型的参数 w ，那么模型的训练什么时候结束，就需要用到代价函数。参数训练的过程就是不断改变 w 从而获得更小的代价 $J(w)$ ，也就是使二分类的准确性更高。

4.3.0.0.2 代价函数中较简单的函数是均方误差

$$E_{in}(h) = \frac{1}{n} \sum_{i=1}^n \left(h_w(x_i) - \frac{1}{2} (1 + y_i) \right)^2 \quad (4.9)$$

4.3.0.0.3 n 代表训练样本个数， $h_w(x_i)$ 代表预测出的第 i 个样本的 y 值， y_i 代表原训练样本中第 i 个样本的实际值。比如我们样本中有五十位病人的数据，每位病人实际上是否有心脏病和被预测出是否有心脏病不一定是一致的，这个方法就是帮助我们判断预测的准确性，均方误差非常直观也便于理解，但却不容易最小化。因此，对于逻辑回归，我们更常使用的代价函数是交叉熵 (Cross Entropy)。

4.3.0.0.4 交叉熵是对“出乎意料” (surprise) 的度量。我们的目标是去计算函数 $xy = y(x)$ 。但是我们却让它计算函数 $xh_w = h_w(x)$ 。假设我们把 h_w 当作 y 等于 1 的概率， $1-h_w$ 是 y 等于 0 的概率。那么，交叉熵衡量的是我们在知道 y 的真实值时的平均“出乎意料”程度。当输出是我们期望的值，我们的“出乎意料”程度比较低；当输出不是我们期望的，我们的“出乎意料”程度就比较高。

$$E_{in}(w) = \frac{1}{n} \sum_{i=1}^n \log \left(1 + e^{-y_i \cdot w^\top x_i} \right) \quad (4.10)$$

4.3.0.0.5 交叉熵是基于 h 的直观概率的解释，非常方便且易于最小化，常规化代价函数 $J(w)$

$$J(w) = \frac{1}{n} \sum_{i=1}^n \log \left(1 + e^{-y_i \cdot w^\top x_i} \right) + \frac{\lambda}{2} \|w\|_2^2 \quad (4.11)$$

4.3.0.0.6 常规化参数 λ 需要在很好地拟合训练集和保持模型相对简单之间进行权衡，这是一个不断尝试的过程。至于找出最佳权重 w ，我们可以用梯度下降

使 $E_{in}(\mathbf{w})$ 最小化，先计算参数 w 的梯度损失 $\frac{\partial J(\mathbf{w})}{\partial w}$ ，再以速度 η 更新参数

$$w' \rightarrow w - \eta \frac{\partial J(w)}{\partial w} = (1 - \eta \lambda)w + \eta \frac{1}{n} \sum_{i=1}^n \frac{y_i x_i}{1 + e^{y_i \cdot w^\top x_i}} \quad (4.12)$$

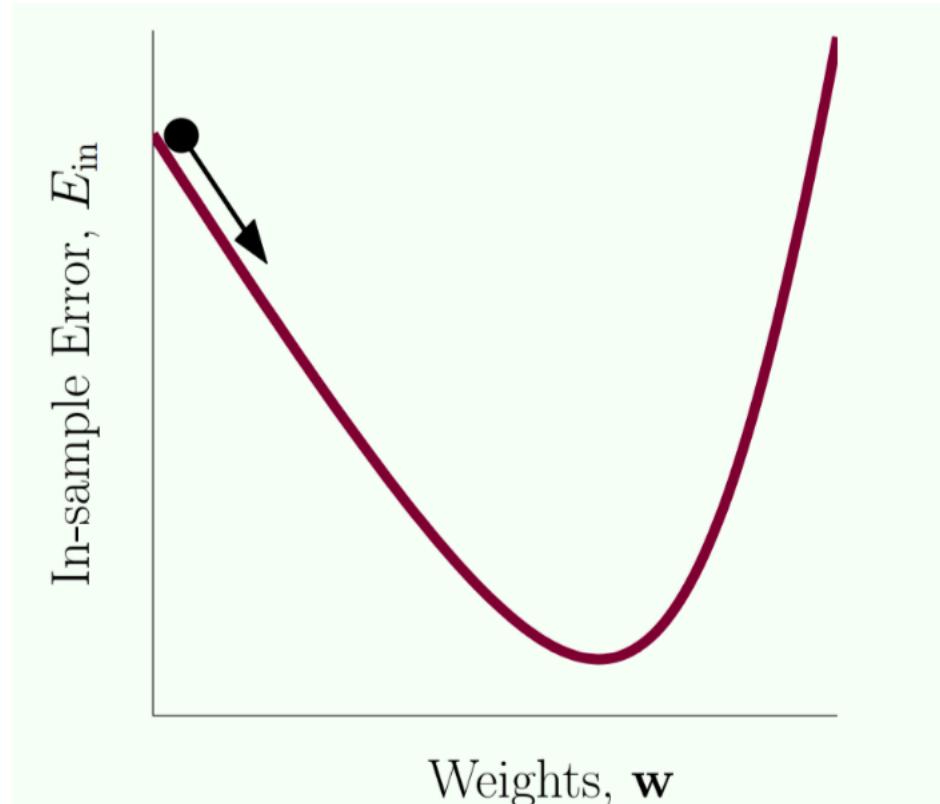


图 4.3: 权重与代价函数的关系

4.3.0.0.7 对于一个样本来说损失函数的梯度为

$$\begin{aligned} \frac{\partial J(w)}{\partial w} &= -\frac{1}{\partial w} \cdot \partial [y \cdot \log h_w(x) + (1 - y) \log (1 - h_w(x))] \\ &= -y \cdot \frac{1}{h_w(x)} \cdot \frac{\partial h_w(x)}{\partial w} + (1 - y) \cdot \frac{1}{1 - h_w(x)} \frac{\partial h_w(x)}{\partial w} \end{aligned}$$

4.3.0.0.8 其中

$$g(z) = \frac{1}{1 + e^{-z}}, g'(z) = \frac{e^{-z}}{(1 + e^{-z})^2} = g(z)[1 - g(z)] \quad (4.13)$$

4.3.0.9 进一步化简损失函数

$$\begin{aligned}
 \frac{\partial J(w)}{\partial w} &= -y \cdot \frac{1}{h_w(x)} \cdot \frac{\partial h_w(x)}{\partial w} + (1-y) \cdot \frac{1}{1-h_w(x)} \frac{\partial h_w(x)}{\partial w} \\
 &= -y \cdot \frac{1}{h_w(x)} \cdot \frac{\partial g(w^\top x)}{\partial w} + (1-y) \cdot \frac{1}{1-h_w(x)} \frac{\partial g(w^\top x)}{\partial w} \\
 &= \left(-\frac{xy}{h_w(x)} + \frac{x(1-y)}{1-h_w(x)} \right) \cdot g(w^\top x) \cdot [1-g(w^\top x)] \\
 &= (h_w(x) - y) x
 \end{aligned} \tag{4.14}$$

4.3.0.10 化简后推导 w , 仍然观察一个样本的梯度下降

$$\begin{aligned}
 \frac{\partial J(w)}{\partial w} &= (h_w(x) - y) x \\
 w := w - \alpha (h_w(x) - y) x
 \end{aligned} \tag{4.15}$$

4.3.0.11 此时可以推导出对于所有样本的梯度下降

$$\begin{aligned}
 \frac{\partial J(w)}{\partial w} &= \frac{1}{n} \sum_{i=1}^n (h_w(x_i) - y_i) x_i \\
 w := w - \frac{1}{n} \sum_{i=1}^n \alpha (h_w(x_i) - y_i) x_i
 \end{aligned} \tag{4.16}$$

4.4 Softmax

4.5 简介

4.5.0.1 Softmax 回归 (Softmax Regression), 也称为多项或多类的逻辑回归, 是逻辑回归在多分类问题上的推广。对于多类 (Multi-class) 问题, 类别标签 $y \in \{1, 2, \dots, k\}$ 可以有 k 个取值, 给定一个样本 x , softmax 回归预测的类别 j 的条件概率为, 从而进行分类, 图 2 为 Softmax 层示意图。

4.6 多分类

4.6.0.1 设想这样一个三分类问题, 给学生学习成果分类, 其中响应变量学生的学习成果的取值可以分为三类, 优秀、普通和落后, 三个取值的概率由学生的各科成绩决定。与线性回归不同的是, 对于每个学生将根据所有科目成绩成绩及科目的权重得到三个输出 (z_1, z_2, z_3) 。

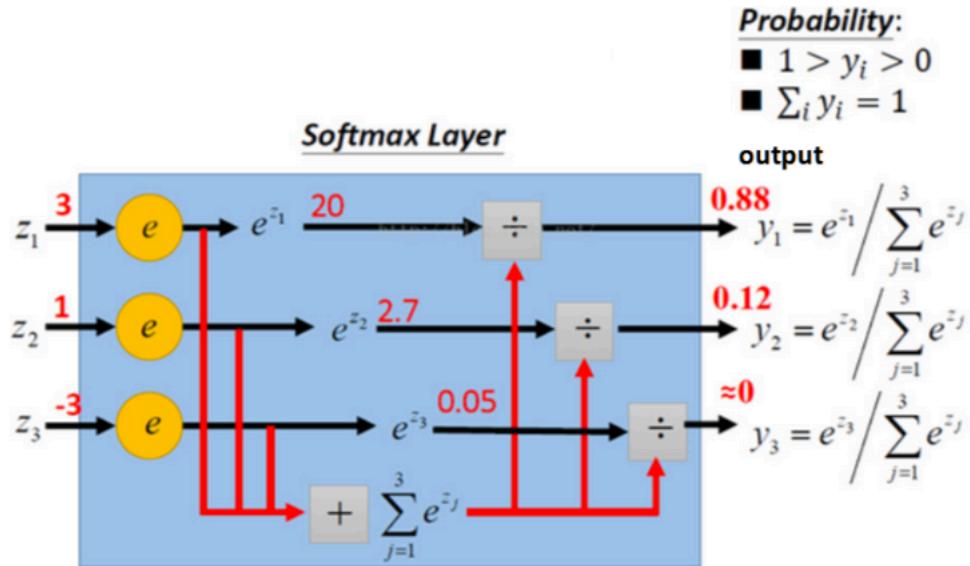


图 4.4: Softmax Layer

4.6.0.0.2 既然分类问题要的到离散的预测输出，一个简单的办法是将三个输出值 z_i 当作是预测类别 i 的置信度，并将值最大的输出所对应的类作为预测输出，如图中所示的 z_1, z_2, z_3 分别为 3, 1, -3，由于 z_1 最大那么预测类别为 1，即该学生的学习成果为优秀。

4.6.0.0.3 然而，直接使用输出层的输出存在一定问题，由于输出层的输出值的输出范围不确定，难以直观判断这些值的意义。另一方面，由于真实标签是离散值，这些离散值与范围无法确定的输出值之间的误差难以衡量。

4.6.0.0.4 Softmax 回归很好的解决了这个问题，Softmax 层将输出值转换成正值且所有值的和为 1 的概率分布：

$$p(y_i = j | x_i; w) = \frac{e^{w_j^\top x_i}}{\sum_{j=1}^k e^{w_j^\top x_i}} \quad (4.17)$$

4.6.0.0.5 $p(y = j | x)$ 代表某类别标签发生的可能性，其中

4.6.0.0.6

$$y_1 = \frac{e^{z_1}}{\sum_{j=1}^3 e^{z_j}} = 0.88, \quad y_2 = \frac{e^{z_2}}{\sum_{j=1}^3 e^{z_j}} = 0.12, \quad y_3 = \frac{e^{z_3}}{\sum_{j=1}^3 e^{z_j}} \approx 0$$

4.6.0.0.7 不难看出 $y_1 + y_2 + y_3 = 1$ ，这是因为 $\frac{1}{\sum_{j=1}^k e^{w_j^\top x(i)}}$ 使分布规范化，此时 $y_1 = 0.88$ ，所以不管 y_2 、 y_3 如何取值，都能得出该学生的学习成果为优秀。

4.7 Softmax 与逻辑回归

4.7.0.0.1 我们习惯性认为逻辑回归用于二分类问题，Softmax 用于多类问题，其实不然，对于 Softmax 回归，当分类数目为两类时，Softmax 与逻辑回归两者作用等价，下面我们从理论上来分析二分类场景。

4.7.0.0.2 Softmax 回归的 Hypothesis 函数为：

$$h_w(x) = \begin{bmatrix} p(y_i = 1|x_i; w) \\ p(y_i = 2|x_i; w) \\ \vdots \\ p(y_i = k|x_i; w) \end{bmatrix} = \frac{1}{\sum_{j=1}^k e^{w_j^\top x_i}} \begin{bmatrix} e^{w_1^\top x_i} \\ e^{w_2^\top x_i} \\ \vdots \\ e^{w_k^\top x_i} \end{bmatrix} \quad (4.18)$$

4.7.0.0.3 当处理二分类问题时：

$$\begin{aligned} h_w(x) &= \begin{bmatrix} p(y = 0|x; w) \\ p(y = 1|x; w) \end{bmatrix} \\ &= \frac{1}{e^{w_0^\top x} + e^{w_1^\top x}} \begin{bmatrix} e^{w_0^\top x} \\ e^{w_1^\top x} \end{bmatrix} \\ &= \frac{1}{e^{(w_0 - w_1)^\top x} + e^{(w_1 - w_0)^\top x}} \begin{bmatrix} e^{(w_0 - w_1)^\top x} \\ e^{(w_1 - w_0)^\top x} \end{bmatrix} \\ &= \frac{1}{e^{(w_0 - w_1)^\top x} + e^{(\mathbf{0})^\top x}} \begin{bmatrix} e^{(w_0 - w_1)^\top x} \\ e^{(\mathbf{0})^\top x} \end{bmatrix} \end{aligned} \quad (4.19)$$

4.7.0.0.4 不妨令 $-\mathbf{w} = \mathbf{w}_0 - \mathbf{w}_1$ ，可得：

$$\begin{aligned} h_w(x) &= \frac{1}{1 + e^{-w^\top x}} \begin{bmatrix} e^{-w^\top x} \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 - \frac{1}{1 + e^{-w^\top x}} \\ \frac{1}{1 + e^{-w^\top x}} \end{bmatrix} \end{aligned} \quad (4.20)$$

4.7.0.0.5 不难看出 Softmax 回归就是一般化的逻辑回归

第 5 章 梯度下降 (Gradient Descent)

5.1 梯度下降 (Gradient Descent)

从第二章可以得知，线性模型中可以通过闭式解的方法求得模型参数的值。这就意味着通过对时间复杂度的分析可知，其时间复杂度达到 $O(n^3)$ ，所以通过闭式解求解模型参数的方法将耗费大量的计算量。同时我们也可以看到，闭式解中存在着矩阵不可逆的风险。为了解决这个问题，我们可以采用梯度下降的方法。在本节中，我们将引入梯度下降的概念，同时介绍随机梯度下降，批量梯度下降以及其他梯度下降方法。

梯度下降算法就像是一个在山间寻找最低点的人，他看不到真正的最低点在哪，甚至他不知道离他最近的局部最低点在哪。他唯一知道的是自己周围的地势，他唯一能够依靠的也只有自己周围的地势。所以为了能够找到一个低的位置，他必须作出判断。一个显而易见的方法便是朝着所在位置更低的方向走去。

我们将上图所展示的山脉图看作是解空间，为了找到一个最低点，旅行者要根据自己所处的环境不断调节自己的方向，使自己向着更低点前进。真正的梯度下降算法是一个批处理算法，缓慢但结果是确定的。梯度下降算法的表达式为：

$$w' = w - \eta \frac{1}{n} \sum_{i=1}^m \frac{\partial}{\partial w} (\mathcal{L}(w)) \quad (5.1)$$

$$\frac{\partial}{\partial w} (\mathcal{L}(w)) = \left(\frac{\partial}{\partial w_1} (\mathcal{L}(w)), \frac{\partial}{\partial w_2} (\mathcal{L}(w)), \dots, \frac{\partial}{\partial w_m} (\mathcal{L}(w)) \right)^T \quad (5.2)$$

其中， $\mathcal{L}(w)$ 为模型的损失函数。此式对损失函数关于 w 求导，之后将导数一一相加求平均值，最后以 η 的学习率获得最终结果。学习率代表了梯度下降的速度。下图为梯度下降的伪代码实现和使用梯度下降的逻辑回归算法实验结果。

学习率是指我们该如何通过损失函数的梯度调整网络权重的超参数。学习率越低，损失函数的变化速度就越慢。虽然使用低学习率可以确保我们不会错过任何局部极小值，但也意味着我们将花费更长的时间来进行收敛。

Algorithm 5: GD

- 1 Initialize parameter w and learning rate η ;
 - 2 **while** an approximate minimum is not obtained **do**
 - 3 | $w' = w - \eta \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial w} (\mathcal{L}(w))$;
 - 4 **end**
-

算法的第一行对 w 和 η 进行初始化，一般来说 w 均初始化为 1， η 则根据需要取值。算法的第 2、3 行说明了算法的迭代公式和一个循环条件，我们可以指定一个精确度来使 w 更加接近结果，当我们预计的情况迟迟不出现，那么也可以根据迭代次数来结束迭代。

梯度下降算法解决了用闭式解计算结果的时间复杂度问题和矩阵不可逆的困境，但是我们也发现梯度下降存在着一些固有的特点：

- 1) 梯度下降对每一个样本求导，使得它所利用的信息是冗余的。
- 2) 现今信息产生的速度极快，我们不可能等到所有的数据都准备好了再去处理。

5.2 梯度下降的理论基础

首先我们要明确的是我们的优化函数是：

$$\arg \min_w \mathcal{L}(w). \quad (5.3)$$

在梯度下降算法里面，我们使用了 $d = -\frac{\partial}{\partial w}(\mathcal{L}(w))$ 作为式子的优化方向。所以我们应该证明：

$$\mathcal{L}(w') = \mathcal{L}(w + \eta d) \leq \mathcal{L}(w) \quad (5.4)$$

其中， $\eta > 0, \eta \rightarrow 0$ 。

为了证明这个式子，我们利用泰勒公式展开有：

$$\begin{aligned} \mathcal{L}(w + \eta d) &= \mathcal{L}(w) + \left(\frac{\partial}{\partial w}(\mathcal{L}(w)) \right)^T \eta d + o(\eta d) \\ &= \mathcal{L}(w) + \left(\frac{\partial}{\partial w}(\mathcal{L}(w)) \right)^T \eta d \end{aligned} \quad (5.5)$$

又注意到：

$$\left(\frac{\partial}{\partial w}(\mathcal{L}(w)) \right)^T \eta d = -\eta d^T d \leq 0 \quad (5.6)$$

所以前式得证。这也就说明了梯度下降算法在每一次迭代中都将使得损失函数减小，使我们得到结果。

5.3 随机梯度下降 (Stochastic Gradient Descent)

有了前一小结的介绍，我们大致知道了随即梯度算法的实现过程和它的缺陷。为了克服这些缺陷，自然而然的我们可以想到如果不是对所有的样本进行求导而是只在已存在的样本集合中选取一个样本进行迭代的方法，所以随机梯度下

降算法被用来加快计算的速度。不同于梯度下降需要对每一个样本求导，随机梯度下降只对所有样本中的一个样本求导，并将其作为迭代的依据。

随机梯度下降的迭代公式为：

$$w' = w - \eta \frac{\partial}{\partial w} (\mathcal{L}_i(w)) \quad (5.7)$$

公式中 $\frac{\partial}{\partial w} (\mathcal{L}_i(w))$ 为以某一个样本为依据的损失函数对 w 的导数。下面是 SGD 算法的结果图和算法伪代码。

Algorithm 6: SGD

```

1 Initialize parameter  $w$  and learning rate  $\eta$ ;
2 while an approximate minimum is not obtained do
3   Randomly select an example  $i$  in the training set;
4    $w' = w - \eta \frac{\partial}{\partial w} (\mathcal{L}_i(w));$ 
5 end
```

像梯度下降算法一样，算法的第一行对 w 和 η 进行初始化，算法的第 2-4 行说明了算法的迭代公式和循环条件。第三行为随机梯度下降算法和梯度下降算法不同的一点，随机梯度下降算法需要每一次迭代随机选取一个样本作为本次迭代的求导对象。

由于只使用了一个样本点，随机梯度下降算法大大加快了训练的速度。当然，这也是随机梯度下降的缺点所在，可以看到，由于训练数据的噪声点比较多，每一次随机选取并不能一定保证选到有利于结果的样本，所以我们在结果图中看到了剧烈的波动。但是又由于迭代多轮，整体来说结果还是倾向于越来越好，朝着极小值更新。

在随机梯度下降算法中，还应该注意到的是学习率对最后结果和学习效率的影响极大。学习率过大过小都将影响训练的难度：

- 1) 当学习率太大，训练的过程会大幅震荡，结果发散，且精度不高。
- 2) 当学习率太小，极易陷入局部最优解且训练速度过慢。

所以我们可以采用动态的学习率来进行训练。所谓动态即是在一开始采用大的学习率，之后每一次迭代都将学习率略微减小。这样在一开始算法可以更快的收敛，在算法的最后结果又可以有一定的精度。下面是一个动态调整学习率的例子：

$$\eta^{t+1} = \frac{\eta^t}{t+1} \quad (5.8)$$

随机梯度下降算法在运行时间上优化了梯度下降算法，它有着一定的优点，也有着一定的缺点：

- 1) 更适合大型数据集，而且通常收敛速度更快。
- 2) 不容易出现局部极小值。



- 3) 适用于非平稳环境以及在线设置
- 4) 即使是最好的经典方法也不能处理随机逼近。
- 5) 难以达到高准确率。
- 6) 收敛性的理论定义不太明确。
- 7) SGD 的另外一个问题是训练速度。可以想象若学习率过大，我们无法获得理想的结果，而若学习率过小，训练可能会非常耗时。

5.4 批量随机梯度下降 (Minibatch SGD)

前面介绍了梯度下降算法和随机梯度下降算法，这两者由于每次迭代选取的样本数目不同导致了各自不同的优点和缺点。梯度下降算法的运行时间和随机梯度下降算法的波动都不是我们想要的。为此，批量随机梯度下降算法平衡了这两种算法，采取了使用 $n(m > n > 0)$ 个样本数进行一次迭代。这使得批量随机梯度下降算法具有梯度下降的稳健和随机梯度下降对运行时间的优化能力。

批量随机梯度下降算法的迭代公式为：

$$w' = w - \eta \frac{1}{|S_k|} \sum_{i \in S_k} \frac{\partial}{\partial w} (\mathcal{L}(w)) \quad (5.9)$$

在这里， S_k 为随机选取的样本组成的集合。下面是 MSGD 算法的结果图和算法伪代码。

Algorithm 7: MSGD

```

1 Initialize parameter  $w$  and learning rate  $\eta$ ;
2 while an approximate minimum is not obtained do
3   | Randomly select  $|S_k|$  examples in the training set;
4   |  $w' = w - \eta \frac{1}{|S_k|} \sum_{i \in S_k} \frac{\partial}{\partial w} (\mathcal{L}(w))$ ;
5 end
```

像之前一样，算法的第一行对 w 和 η 进行初始化，算法的第 2-4 行说明了算法的迭代公式和循环条件。第三行为批量随机梯度下降算法和其他算法不同的一点，批量随机梯度下降算法需要每一次迭代随机选取 $|S_k|$ 样本作为本次迭代的求导对象。

批量随机梯度下降算法综合了梯度下降算法和随机梯度下降算法，但是仍然存在一些问题：

- 1) 选择一个合适的学习率非常重要，如果学习率设置地非常小的话，收敛率非常慢。但是，学习率过大的话，算法不容易收敛，导致目标函数在局部最小点周围跳转而不收敛到最小点。
- 2) 也可以在迭代的过程中动态地改变学习率，但是需要预先定义一些规则。但是这种规则不能适用于不同的数据。

- 3) 当目标函数是高度非凸的时候, 比如神经网络, 我们希望避免参数陷在局部最优点。

5.5 例子

我们假设一个优化函数:

$$\frac{\|w\|^2}{2} + C \sum_{i=1}^N \max(0, 1 - y_i(w^T x_i + b)) \quad (5.10)$$

由于要分类讨论, 在此我们假设:

$$g_w(x_i) = \frac{\partial(\max(0, 1 - y_i(w^T x_i + b)))}{\partial w} \quad (5.11)$$

$$g_b(x_i) = \frac{\partial(\max(0, 1 - y_i(w^T x_i + b)))}{\partial b} \quad (5.12)$$

分类讨论有, 当 $1 - y_i(w^T x_i + b) \geq 0$

$$\begin{aligned} g_w(x_i) &= \frac{\partial(\max(0, 1 - y_i(w^T x_i + b)))}{\partial w} \\ &= -\frac{\partial y_i w^T x_i}{\partial w} \\ &= -y_i x_i \end{aligned} \quad (5.13)$$

$$\begin{aligned} g_b(x_i) &= \frac{\partial(\max(0, 1 - y_i(w^T x_i + b)))}{\partial b} \\ &= -\frac{\partial y_i b}{\partial b} \\ &= -y_i \end{aligned} \quad (5.14)$$

当 $1 - y_i(w^T x_i + b) < 0$

$$\begin{aligned} g_w(x_i) &= \frac{\partial(\max(0, 1 - y_i(w^T x_i + b)))}{\partial w} \\ &= 0 \end{aligned} \quad (5.15)$$

$$\begin{aligned} g_b(x_i) &= \frac{\partial(\max(0, 1 - y_i(w^T x_i + b)))}{\partial b} \\ &= 0 \end{aligned} \quad (5.16)$$

又由于:

$$\frac{\partial \|w\|^2}{\partial w} = 2w \quad (5.17)$$

所以，综上我们得到最终的导数为：

$$\frac{\partial f(w, b)}{\partial w} = w + C \sum_{i=1}^N g_w(x_i) \quad (5.18)$$

$$\frac{\partial f(w, b)}{\partial b} = C \sum_{i=1}^N g_b(x_i) \quad (5.19)$$

所以根据之前介绍的梯度下降算法，我们有迭代公式：

$$w' = w - \eta \frac{\partial f(w, b)}{\partial w} \quad (5.20)$$

$$b' = b - \eta \frac{\partial f(w, b)}{\partial b} \quad (5.21)$$

5.6 其他方法

5.6.1 Momentum

在上面所描述的方法中，我们可以直观的感受到当我们迭代到一个局部最小值时，我们存在极大的可能将陷在那里，又或者当我们遇到一个鞍点，使得导数为0，那么迭代也变得没有了意义。这时候如果存在一种能够冲出这一点的方法就变得极有意义了。

设想我们在每次迭代所处的位置决定了我们下一步怎么走，这就意味着每一步我们都是没有初速度的。那么如果我们在迭代过程中加入一个初速度使得迭代方向既受到当前位置的影响，也收到之前状态的影响，那么对于迭代冲出局部最优点和鞍点极有意义。Momentum 方法于是应运而生：

Algorithm 8: Momentum

```

1 Initialize parameter  $w$  and learning rate  $\eta$ ;
2  $v=0$ ;
3 while an approximate minimum is not obtained do
4    $d = \frac{1}{n} \sum_{i=1}^n \frac{\partial}{\partial w} (\mathcal{L}(w))$ ;
5    $v_{t+1} = rho * v_t * v_t + d$ ;
6    $w' = w - \eta v_{t+1}$ ;
7 end

```

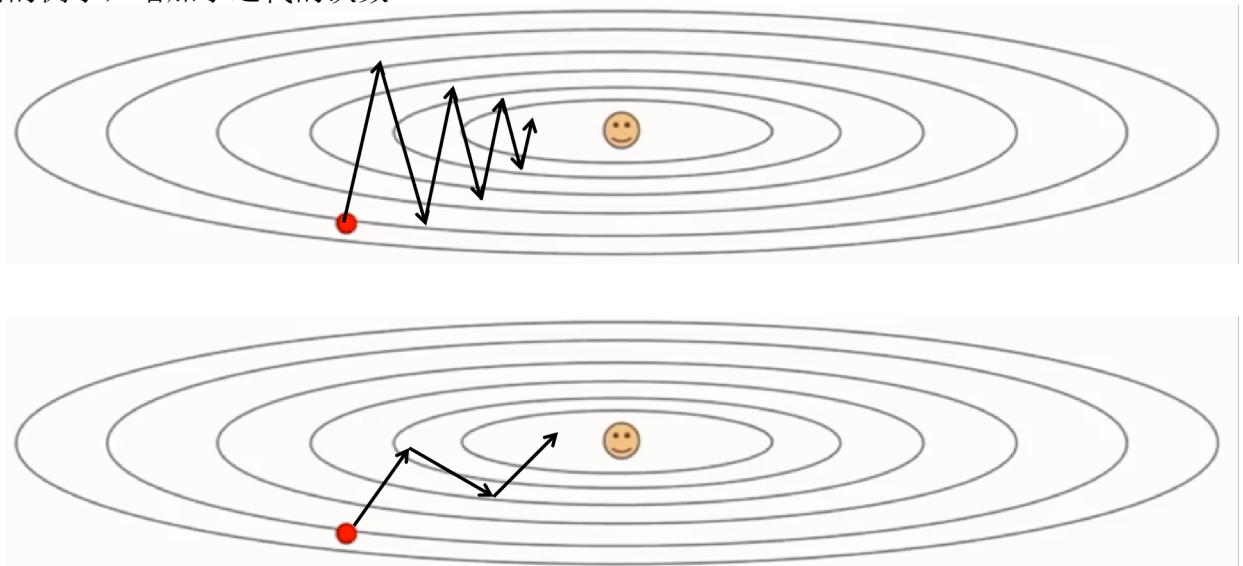
在 Momentum 方法中，我们用速度 v 代替了梯度来对 w 进行迭代。从伪代码的第五行来看，变量 v 收到之前的速度和当前的梯度影响。其中 rho 是摩擦系数，一般来说取 0.9 或 0.99，来使得之前的速度对之后的影响越来越小。在 Momentum

方法中，由于速度 v 并不是一个超参数，所以将其初始化为 0 即可。

5.6.2 Adagrad

之前在 SGD 算法中我们探讨了学习率对于最后结果的影响，但是如果我只是盲目的去渐渐减小学习率实在是下策。Adagrad 算法是一个非常有效的学习率更新算法，该算法的提出不仅仅是由于全局学习率大小对结果的影响，更是由于不同维度梯度大小的不一致导致的运行时间和曲折的学习路线问题。

下图是两种由于在不同方向上梯度大小问题所导致的结果。第一幅图由于在一个方向上梯度过大导致摇摆幅度过大，使得最后综合来看不如各个方向梯度都均衡的例子，增加了迭代的次数。



为了解决这个问题，Adagrad 算法便被提了出来：

Algorithm 9: Adagrad

- 1 Initialize parameter w and learning rate η ;
- 2 Initialize the small parameter δ ;
- 3 $r=0$;
- 4 **while** an approximate minimum is not obtained **do**
- 5 Randomly select $|S_k|$ examples in the training set;
- 6 $d = \frac{1}{|S_k|} \sum_{i \in S_k} \frac{\partial}{\partial w} (\mathcal{L}(w))$;
- 7 $r = r + d * d$;
- 8 $\theta' = \theta - \frac{\eta}{\delta + \sqrt{r}}$;
- 9 $w' = w - \theta' d$;
- 10 **end**

简单来讲，设置全局学习率之后，每次通过全局学习率逐参数的除以历史梯度平方和的平方根，使得每个参数的学习率不同。起到的效果是在参数空间更为

平缓的方向，会取得更大的进步（因为平缓，所以历史梯度平方和较小，对应学习下降的幅度较小），并且能够使得陡峭的方向变得平缓，从而加快训练速度。其中 δ 是一个很小的量，用于使除法运算分母不为 0。

5.6.3 RMSProp

Adagrad 算法的创新之处在于它将梯度逐渐减小并解决了不同维度下迭代速度不一致的问题。RMSProp 加了一个衰减系数来控制历史信息的获取多少，实践表明在非凸条件下 RMSProp 算法表现的比 Adagrad 算法更好。

该算法如下所示：

Algorithm 10: RMSProp

```

1 Initialize parameter  $w$  and learning rate  $\eta$ ;
2 Initialize the small parameter  $\delta$ ;
3 Initialize the attenuation coefficient  $\rho$ ;
4  $r=0$ ;
5 while an approximate minimum is not obtained do
6   Randomly select  $|S_k|$  examples in the training set;
7    $d = \frac{1}{|S_k|} \sum_{i \in S_k} \frac{\partial}{\partial w} (\mathcal{L}(w))$ ;
8    $r = \rho r + d * d$ ;
9    $\theta' = \theta - \frac{\eta}{\delta + \sqrt{r}}$ ;
10   $w' = w - \theta' d$ ;
11 end
```

我们可以从伪代码中看到，RMSProp 和 Adagrad 算法的主要区别只有在对 r 的更新中加入了一个衰减系数 ρ 。Adagrad 会累加之前所有的梯度平方，而 RMSprop 仅仅是计算对应的平均值，因此可缓解 Adagrad 算法学习率下降较快的问题。

当然，无论是 Momentum 算法、Adagrad 算法还是 RMSprop 算法，之后要谈到的 Adam 算法综合了上述算法的优点，于是 Adam 算法也成了一个比其他算法更通用的算法。

5.6.4 Adam

Adam 算法的提出者描述其为两种随机梯度下降扩展式的优点集合，即：适应性梯度算法（AdaGrad）为每一个参数保留一个学习率以提升在稀疏梯度（即自然语言和计算机视觉问题）上的性能。均方根传播（RMSProp）基于权重梯度最近量级的均值为每一个参数适应性地保留学习率。这意味着算法在非稳态和在线问题上有很有优秀的性能。Adam 算法同时获得了 AdaGrad 和 RMSProp 算法

的优点。Adam 不仅如 RMSProp 算法那样基于一阶矩均值计算适应性参数学习率，它同时还充分利用了梯度的二阶矩均值（即有偏方差）。

Algorithm 11: Adam

```

1 Initialize parameter  $w$  and learning rate  $\eta$ ;
2 Initialize the small parameter  $\delta$ ;
3 Initialize the attenuation coefficient  $\rho_1$  and  $\rho_2$ ;
4  $r=0$ ;
5  $v=0$ ;
6 while an approximate minimum is not obtained do
7   Randomly select  $|S_k|$  examples in the training set;
8    $d = \frac{1}{|S_k|} \sum_{i \in S_k} \frac{\partial}{\partial w} (\mathcal{L}(w))$ ;
9    $r = \rho_1 r + (1 - \rho_1) d * d$ ;
10   $v = \rho_2 v + (1 - \rho_2) d$ ;
11   $\hat{r} = r / (1 - \rho_1)$ ;
12   $\hat{v} = v / (1 - \rho_2)$ ;
13   $\theta' = \theta * \frac{\hat{v}}{\delta + \sqrt{\hat{r}}}$ ;
14   $w' = w - \theta' d$ ;
15 end

```

其中， ρ_1 和 ρ_2 可以设置为一个接近 1 的数值，例如 0.9 或 0.99。 δ 是一个极小的量，可以取 10^{-7} ，目的是为预防除法运算分母为 0 的情况。



第 6 章 过拟合、正则化和验证

6.1 过拟合 – Overfitting

6.1.1 训练误差和泛化误差

我们继续讨论机器学习课程成绩的例子。如果，我要用高等数学、C 语言、线性代数、毛概这几门课程预测机器学习课程成绩，我可以建立一个简单的线性回归模型去拟合它，也可以为了适应更复杂的数据而选择复杂的高阶多项式模型。然而，你可能会发现，当高阶多项式模型在训练数据集上更准确时，它在测试数据集上却不一定更准确。这是为什么呢？

我们先来看这样一个例子，一个多项式函数拟合的过程：

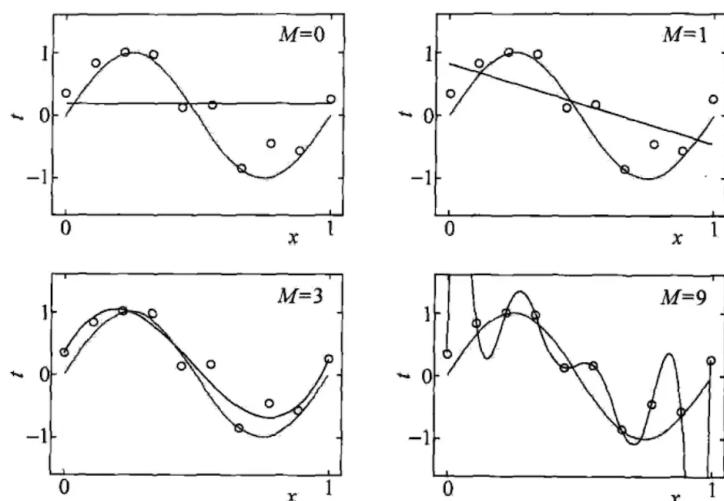


图 1.2 M 次多项式函数拟合问题的例子

可以发现，随着多项式函数拟合次数的增多，对于训练数据的拟合越来越好，拟合的线能够穿过所有的数据点，但是对新数据的预测效果往往很差。

为了解释上述现象，我们需要区分训练误差（Training Error）和泛化误差（Generalization Error）。通俗来讲，前者指模型在训练数据集上表现出的误差，后者指模型在任意一个测试数据样本上表现出的误差的期望，并常常通过测试数据集上的误差来近似。

我们以考试成绩为例来直观地解释训练误差和泛化误差这两个概念。训练误差可以认为是做课本上课后题的错误率，泛化误差则可以通过考试题目的错误率来近似。假设课后题和考试题都随机采样于一个未知的依照相同考纲的巨大试题库。如果一位同学完全不学习，那么他在课后题和考试题上的错误率都很高，也

就是训练误差和泛化误差都很大。另一位同学如果对课本上的知识点掌握得很好，课后题和考试题的错误率都很低，那么训练误差和泛化误差都很小。第三位同学没有很好的掌握学习方法，只是对课后题死记硬背，变通能力不够好，那么训练误差很小，泛化误差却很大。

6.1.2 过拟合与欠拟合

我们继续来看上图的例子， $M=0$ 或 $M=1$ 时，拟合的是一条直线，模型无法得到较低的训练误差，我们把这种现象叫做欠拟合 (Underfitting)； $M=9$ 时，拟合的曲线过度复杂，在测试数据集上无法得到良好的判断，误差远大于在训练数据集上的误差，我们把这种现象叫做过拟合 (Overfitting)。随着模型复杂度的增加，泛化误差和训练误差的变化趋势如下：

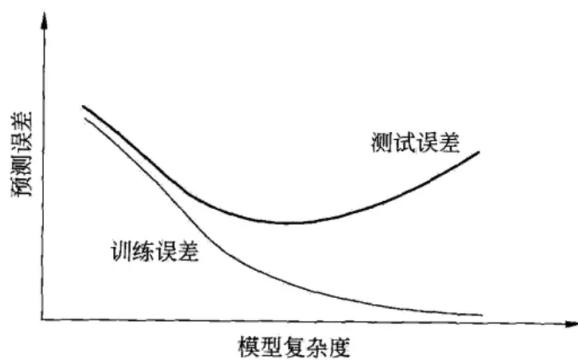
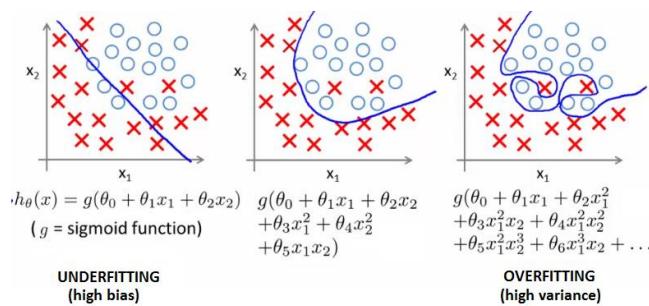


图 1.3 训练误差和测试误差与模型复杂度的关系

可以看出来，欠拟合和过拟合都会使泛化误差较大，不能取得较好的效果。下面用一张图来更加直观的表示欠拟合和过拟合的区别：



6.1.3 如何减少过拟合

如何提高神经网络的泛化能力反而成为影响模型能力的最关键因素，那么怎么减少过拟合呢？

1. 获取更多数据

这是解决过拟合最有效的方法，只要给足够多的数据，让模型「看见」尽可能多的例外情况，它就会不断修正自己，从而得到更好的结果。

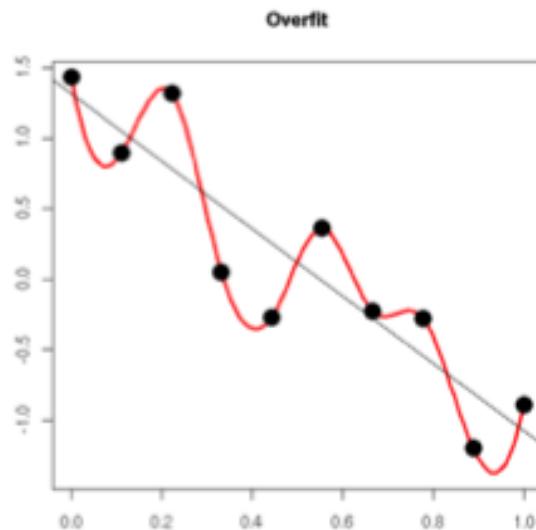
如何获取跟多的数据，可以有以下几个方法：

- (a). 从数据源头获取更多数据：这个是最容易想到的，比如预测机器学习成绩，我再多找几位同学的数据就好了；但是，在很多情况下，大幅增加数据本身就不容易；另外，我们不清楚获取多少数据才算够。
 - (b). 根据当前数据集估计数据分布函数，使用该分布产生更多数据：这个一般不用，因为估计分布参数的过程也会代入抽样误差。
 - (c). 数据增强（Data Augmentation）：通过一定规则扩充数据。如在物体分类问题里，物体在图像中的位置、姿态、尺度、整体图片明暗度等都不会影响分类结果。我们就可以通过图像平移、翻转、缩放、切割等手段将数据库成倍扩充。
2. 使用合适的模型
- 过拟合主要是由两个原因造成的：数据太少 + 模型太复杂。所以，我们可以通过使用合适复杂度的模型来防止过拟合问题，让其足够拟合真正的规则，同时又不至于拟合太多抽样误差。
- (a). 网络结构：这个很好理解，减少网络的层数、神经元个数等均可以限制网络的拟合能力。
 - (b). 正则化（Regularization）：正则化是一种比较好的防止过拟合的方法，我们会在下一节详细讨论。
3. 结合多种模型
- 简而言之，训练多个模型，以每个模型的平均输出作为结果，比如：Bagging, Boosting, Dropout。

6.2 正则化 – Regularization

6.2.1 什么是正则化

在传统的机器学习中，提高泛化能力的方法主要是限制模型复杂度。如果参数对应一个较小的值，那么会得到形式更加简单的拟合。然而，越复杂的模型，越是会尝试拟合所有的训练数据，包括一些异常样本，这就容易造成在较小的区间内预测值产生较大的波动，这种大的波动反映了在某些小的区间里导数值很大。而只有较大的参数值才能产生较大的导数。因此复杂的模型，其参数值会比较大：



正则化（Regularization）是一类通过限制模型复杂度，从而避免过拟合，提高泛化能力的方法，包括引入一些约束规则，增加风险，提前停止等。

惩罚高阶参数，使它们趋近于 0，这样就会得到较为简单的假设，也就是得到简单的函数，这样就不易发生过拟合。但是在实际问题中，并不知道哪些是高阶多项式的项，所以在代价函数中增加一个正则化项，将代价函数中所有参数值都最小化，收缩每一个参数。

对于我们的机器学习课程成绩预测来说，如果采用非常高的高阶多项式来拟合，我们将会得到一个非常弯曲和复杂的曲线函数，现在我们只需要用正则化的方法，就可以得到一个更加合适的曲线，但这个曲线不是一个真正的二次函数，而是更加流畅和简单的一个曲线。这样就得到了对于这个数据更好的假设。

6.2.2 L1 和 L2 正则化

正则化参数要做的就是控制两个目标之间的平衡关系，在最小化训练误差的同时，正则化参数使模型简单。最小化误差是为了更好的拟合训练数据，正则化参数是为了防止模型过分拟合训练数据。L1 和 L2 正则化是机器学习中最常用的正则化方法，通过约束参数的 L1 和 L2 范数来减小模型在训练数据集上的过拟合现象。

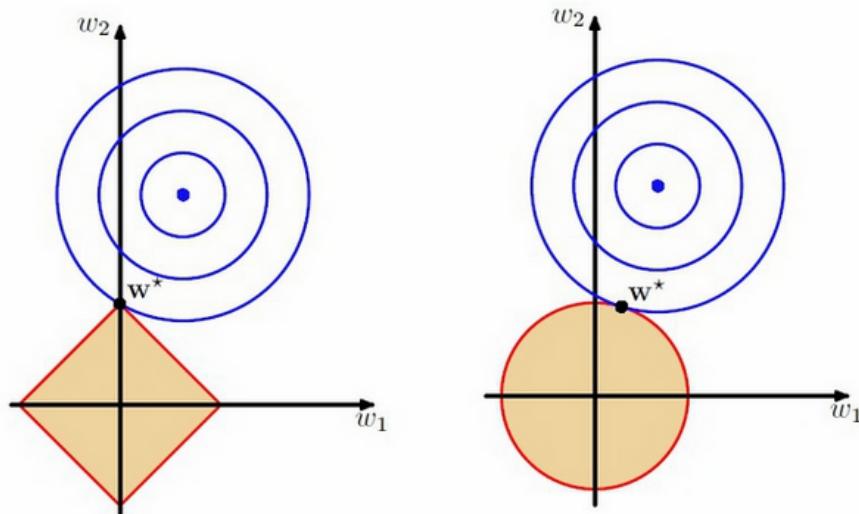
L1 正则：

$$L(w) = \frac{1}{N} \sum_{i=1}^N (f(x_i; w) - y_i)^2 + \lambda ||Q|| \quad (6.1)$$

L2 正则：

$$L(w) = \frac{1}{N} \sum_{i=1}^N (f(x_i; w) - y_i)^2 + \frac{\lambda ||Q||^2}{2} \quad (6.2)$$

L1 与 L2 正则的比较：假设我们有一个二维模型，在 (w_1, w_2) 平面上可以画出损失函数的等高线，圆心就是样本值，半径就是误差，而约束条件则就是红色边界。等高线与约束条件相交的地方就是最优解。



左图为 L1 正则的约束项，右图为 L2 正则的约束项。通过可视化可以发现，使用 L1 正则化在取得最优解的时候 w_1 的值为 0，相当于去掉了特征，而使用 L2 正则化在取得最优解的时候特征参数都有大于 0 的值。

所以我们可以得出结论：L1 会趋向于产生少量的特征，而其他的特征都为 0，而 L2 会选择更多的特征，特征值都趋近于 0。L1 正则在选择特征时非常有用，而 L2 在实际训练中更有用。所以，在所有特征中如果只有少数特征起重要作用，选择 L1 进行特征选择；而如果所有特征中大部分都能起作用，而且作用很平均，那么使用 L2 会更合适。

6.3 验证 – Validation

6.3.1 保留交叉验证

在机器学习中，通常需要评估若干候选模型的表现并从中选择模型，这一过程称为模型选择（Model Selection）。可供选择的候选模型可以是有着不同超参数的同类模型。然而，从严格意义上讲，测试集只能在所有超参数和模型参数选定后使用一次。不可以使用测试数据选择模型，如调参。由于无法从训练误差估计泛化误差，因此也不应只依赖训练数据选择模型。鉴于此，我们可以预留一部分在训练数据集和测试数据集以外的数据来进行模型选择。这部分数据被称为验证数据集，简称验证集（Validation Set）。例如，我们可以从给定的训练集中随机选取一小部分作为验证集，而将剩余部分作为真正的训练集。下面这个方法叫保留交叉验证（Hold-out Cross Validation），也叫简单交叉验证（Simple Cross Validation）：

Algorithm 12: 保留交叉验证

Input: 训练集 \mathcal{T}

- 1 随机拆分训练集 \mathcal{T} 成 \mathcal{T}_{train} (例如, 可以选择整体数据中的 70% 用于训练) 和 \mathcal{T}_{cv} (训练集中剩余的 30% 用于验证)。这里的 \mathcal{T}_{cv} 就叫做保留交叉验证集 (Hold-out Cross Validation Set) ;
- 2 对集合 \mathcal{T}_{train} 中的每一个模型进行训练, 然后得到假设类 (Hypothesis) \mathcal{F}_i ;
- 3 筛选并输出对保留交叉验证集有最小误差 $\mathcal{L}_{S_{cv}}(\mathcal{F}_i)$ 的假设 \mathcal{F}_i ;

Output: \mathcal{F}_i

这样通过在一部分未进行训练的样本集合 \mathcal{T}_{cv} 上进行测试, 我们对每个假设 \mathcal{F}_i 的真实泛化误差就能得到一个比较好的估计, 然后就能选择出来一个有最小估计泛化误差 (Smallest Estimated Generalization Error) 的假设了。

6.3.2 K 折交叉验证

使用保留交叉验证集的一个弊端就是“浪费”了训练样本数据集的 30% 左右。甚至即便我们使用了备选的那个针对整个训练集使用模型进行重新训练的步骤, 也还不成, 因为这无非是相当于我们只尝试在一个 $0.7n$ 规模的训练样本集上试图寻找一个好的模型来解决一个机器学习问题, 而并不是使用了全部的 n 个训练样本, 进行训练而得到的模型。当然了, 如果数据非常充足, 或者是很廉价的话, 也可以用这种方法, 而如果训练样本数据本身就很稀缺的话 (比如说只有 20 个样本), 那就最好用其他方法了。

下面就是一种这样的方法, 名字叫 k-折交叉验证 (K-fold Cross Validation), 这样每次的用于验证的保留数据规模都更小:

通常这里进行折叠的次数 (Number of Folds) k 一般是 10, 即 $k = 10$ 。这样每次进行保留用于验证的数据就只有 $1/k$, 这就比之前的 30% 要小多了, 当然这样一来这个过程也要比简单的保留交叉验证方法消耗更多算力成本, 因为现在需要对每个模型都进行 k 次训练。



Algorithm 13: k-折交叉验证

Input: 训练集 \mathcal{T}

- 1 随机将训练集 \mathcal{T} 切分成 k 个不想交的子集。其中每一个子集的规模为 n/k 个训练样本。这些子集为 $\mathcal{T}_1, \dots, \mathcal{T}_k$;
- 2 对每个模型 M_i , 我们都按照下面的步骤进行评估 (Evaluate): 对 $j = 1, \dots, k$ 在 $\mathcal{T}_1 \cup \dots \cup \mathcal{T}_{j-1} \cup \mathcal{T}_{j+1} \cup \dots \cup \mathcal{T}_k$ 上 (也就是除了 \mathcal{T}_j 之外的其他数据), 对模型 M_i 得到假设 \mathcal{F}_{ij} 。接下来针对 \mathcal{T}_j 使用假设 \mathcal{F}_{ij} 进行测试, 得到经验误差 $\mathcal{L}_{S_{cv}}(\mathcal{F}_{ij})$;
- 3 对每个 $\mathcal{L}_{S_{cv}}(\mathcal{F}_{ij})$ 取平均值, 计算得到的值就当作是模型 M_i 的估计泛化误差 (Estimated Generalization Error) ;
- 4 选择具有最小估计泛化误差 (Lowest Estimated Generalization Error) 的模型 M_i , 然后再整个训练样本集 \mathcal{T} 上重新训练该模型。这样得到的假设 (Hypothesis) 就可以输出作为最终结果。;

Output: 模型 M_i

第 7 章 决策树和随机森林

7.1 决策树

7.1.1 决策树的概念

决策树是机器学习中十分经典的监督学习算法，被广泛应用于分类及回归问题：分类树被用于解决分类问题，它的输出是样本的类别；回归树被用于解决回归问题，它的输出是一个实数，例如学生成绩、房屋价格等。在这里，我们主要讨论的决策树是分类树，回归树将在后面章节讨论。决策树的原理十分简单，类似于不断嵌套的 if-else 语句，判断当前的条件（即样本的属性）进入不同的分支，最终在叶子结点做出一个分类决策。

例如，我们需要挑选出符合条件的学生参加“数学建模比赛”，可能构建出如下的决策树（图7.1）：

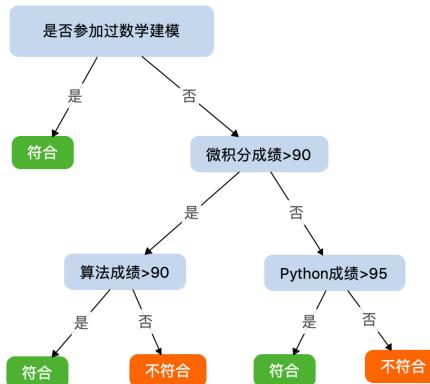


图 7.1：选择符合条件的学生参加“数学建模比赛”

对于给定的带标签的数据集，我们可以根据数据集的属性来构建一棵决策树，从而根据特征来对样本进行分类。

7.1.2 决策树的生成

现在我们知道对于给定的带标签的数据集，我们可以通过构建决策树来对样本进行分类，从而实现预测的目的，那么我们应该如何构建决策树呢？

Algorithm 14: 决策树生成算法

输入： 训练集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)\}$
 属性集 $A = \{a_1, a_2, \dots, a_d\}$

输出： 以 node 为根结点的决策树

```

1: 函数 TreeGenerate(D,A)
2: 生成结点 node;
3: if D 中所有样本都属于类别 C then
4:   将 node 标记为 C 类叶子结点;
5:   return
6: end if
7: if A ≠ ∅ or D 中样本在 A 上取值相同 then
8:   将 node 标记为叶节点, 其类别标记为 D 中样本数最多的类;
9:   return
10: end if
11: 从 A 中选择最优划分属性  $a_*$ ;
12: for  $a_*$  的每一个值  $a_*^v$  do
13:   为 node 生成一个分支; 令  $D_v$  表示 D 中在  $a_*$  上取值为  $a_*^v$  的样本子集;
14:   if  $D_v$  为空 then
15:     将分支结点标记为叶结点, 其类别标记为 D 中样本最多的类;
16:   return
17: else
18:   以 TreeGenerate( $D_v, A \setminus \{a_*\}$ ) 为分支结点;
19: end if
20: end for

```

7.1.2.1 生成算法

使用决策树进行分类是一个自顶向下的过程，在构建决策树时，我们也同样采用自顶向下的过程：给定训练集 D ，我们先生成一个结点 N ，然后判断该训练集是否都为同一类 C ，若是则将该结点标记为 C 类并返回结点 N ；否则，我们选择 D 中的其中一个属性作为结点 N 的分类依据，将训练集 D 划分成两个子集 D_1 和 D_2 ，再以训练集 D_1 和 D_2 分别递归上述过程，直至返回结点，并将返回结点作为当前结点 N 的子结点后返回即可。算法14给出了决策树生成算法的伪代码。

当然，这只是简单介绍了一下核心的构建过程，在实际编写代码还有一些细节需要处理，比如没有可以用来划分的属性了，或训练集的子集 D_1 或 D_2 存在空集等。

思考：算法14并没有介绍应该如何选择最优的划分属性，那么在实践中我们应该如何选择属性呢？



7.1.3 选择属性

选择属性的算法有很多，其中我们可以使用信息增益、增益率或基尼指数来度量属性。

决策树的构建，依赖于数据的属性，因为在决策树的每一个结点，我们都需要根据数据的属性来做一次决策，选择下一个分支。而为了构建一棵好的决策树，我们需要选择出尽可能好的特征，这样我们的决策树也会尽可能地小，效率也会更高。

好的属性应该具备这样的特点：在该属性上取值相同的样本，都为同一类别，或尽可能多地为同一类别；相反，若某个属性对样本的区分度不大，即在该属性上取值相同的样本中，正负类的样本数量差不多，这样的属性则不太好。

例如，我们的任务是“判断这个人的性别”，那么我们可以先选择“头发长度”这一属性进行判断，因为往往大多数男生的头发比较短，女生的头发则比较长。在判断完“头发长度”这一属性以后，我们就已经能够判断出大部分的男生和女生了，此时，我们可以再通过诸如衣着、身高、体重等其他属性进一步确认这个人的性别。如果我们最开始就选择“体重”进行判断，我们则很难将大部分的男生和女生分开，这也意味着我们选择的属性对我们的分类任务并没有太多的帮助，这只会加深我们的决策树。因此，在构建决策树的时候，就应该避免选择这样的属性，而应该选择像“头发长度”这样具有强区分度的属性。

7.1.3.1 信息熵

在了解信息增益和增益率之前，我们有必要先来了解一下信息熵。

信息熵是一个用来度量数据集纯度的指标。给定带有正类和负类的数据集 D ，那么 D 的信息熵定义如下（公式7.1）：

$$\text{Entropy}(D) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_- \quad (7.1)$$

其中， p_+ 是正类占总样本的比例， p_- 是负类占总样本的比例。

例如，训练集 D 中有 10 个正类样本和 5 个负类样本，那么 $p_+ = \frac{2}{3}$ ， $p_- = \frac{1}{3}$ ，因此训练集 D 的信息熵是

$$\text{Entropy}(D) = -\frac{2}{3} \log_2 \frac{2}{3} - \frac{1}{3} \log_2 \frac{1}{3} = 0.918$$

$\text{Entropy}(D)$ 的函数图像如图7.2。

从图中可以看出， $\text{Entropy}(D)$ 的值越小，代表 D 的纯度越高。当样本中仅有正类或仅有负类时，即 $p_+ = 1$ 或 $p_+ = 0$ 时，样本纯度最高，此时 $\text{Entropy}(D) = 0$ ；当正类与负类各占一半时，即 $p_+ = 0.5$ 时，样本纯度最低，此时 $\text{Entropy}(D) = 1$ 。

对于多分类问题，我们也有对应的信息熵。

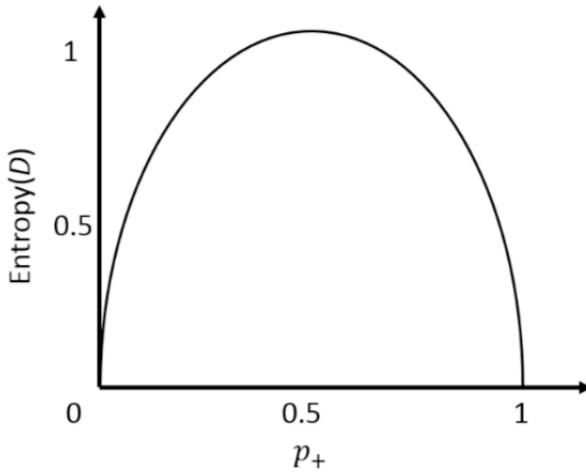


图 7.2: \$\text{Entropy}(D)\$ 的函数图像

如果给定的训练集 D 有 c 个类，那么 D 的信息熵为：

$$\text{Entropy}(D) \equiv \sum_{n=1}^c -p_i \log_2 p_i \quad (7.2)$$

其中， p_i 为第 i 类占总样本的比例。

7.1.3.2 信息增益

信息增益反映的是通过属性 A 划分训练集 D 能够带来的纯度提升量，它能够直观的反映属性 A 的划分效果—— $\text{Gain}(D, A)$ 越大，划分后的纯度提升越大，属性 A 的划分效果越好。ID3 算法则是使用了信息增益来选择属性。

信息增益的定义为

$$\text{Gain}(D, A) = \text{Entropy}(D) - \sum_{j=1}^v \frac{|D_j|}{|D|} \text{Entropy}(D_j) \quad (7.3)$$

假设我们以属性 A 来对训练集 D 进行划分，其中属性 A 共有 v 个值，这样我们可以将 D 划分成 v 个子集 D_j ，其中 $j \in \{1, 2, \dots, v\}$ 。此时，我们可以根据上述公式分别计算出 v 个子集的信息熵 $\text{Entropy}(D_j)$ 。我们希望挑选出具有尽可能好的划分效果的属性 A ，换句话说，我们希望通过属性 A 划分后的子集应尽可能地纯净，即 $\text{Entropy}(D_j)$ 应越小越好。而为了衡量所有子集的纯净度，我们可以通过给每个子集的信息熵乘上一个权重 $\frac{|D_j|}{|D|}$ 后求和，得到按属性 A 划分后的 D 的信息熵的数学期望：

$$\text{Entropy}_A(D) = \sum_{j=1}^v \frac{|D_j|}{|D|} \text{Entropy}(D_j) \quad (7.4)$$

通过这个数学期望，便能够反映所有子集加权平均后的纯净度， $Entropy_A(D)$ 越小则说明通过属性 A 划分后的子集越纯净，划分效果则越好。但是，仅通过 $Entropy_A(D)$ 来判断划分效果是不够的。试想，倘若训练集 D 在通过属性 A 划分之前就已经十分纯净，比如，当 D 中的所有样本均为同类时，那么通过属性 A 划分后的子集仍然为同类，依然十分纯净，此时 $Entropy(D_j)$ 均为 0，那么 $Entropy_A(D) = 0$ 。可以看到，虽然 $Entropy_A(D)$ 非常小，但是这并不是属性 A 的功劳，而是因为本来样本就已经很纯净了。

为了解决这个问题，我们还需要考虑到划分前训练集的情况，即考虑到 $Entropy(D)$ 。而我们其实只需要计算 $Entropy(D) - Entropy_A(D_j)$ 即可，这就是信息增益了（公式7.3）。

通过计算属性的信息增益，我们就可以挑选出具有最优划分效果的属性了。

现在让我们看一个例子。由于编译技术的挂科率非常高，学校现在希望找出正在修这门课的有可能挂科的同学，并给他们发出提醒。图7.3是上一届学生的成绩，我们需要根据现有的数据集，训练出一棵决策树来预测编译技术有可能挂科的同学。

	操作系统	C++	计算机网络	数据结构与算法	数学分析	编译技术
0	54	83	62	49	65	35
1	51	61	63	24	27	77
2	65	57	61	74	65	56
3	79	72	76	79	91	88
4	92	69	87	95	100	90
5	83	84	82	94	89	82
6	63	75	64	83	62	60
7	61	69	88	51	45	50
8	76	78	91	87	79	72
9	60	58	65	78	74	56
10	73	56	72	76	76	29
11	87	87	89	79	90	75

图 7.3：上一届学生的成绩

为了简化起见，我们先将数据以 60 为界限进行二值化，小于 60 则认为是挂科，处理后得到图7.4。

我们首先计算一下所有属性的信息增益。

$$Gain(D, \text{操作系统}) = 0.98 - 0.975 = 0.05$$

$$Gain(D, C++) = 0.98 - 0.573 = 0.407$$

$$Gain(D, \text{计算机网络}) = 0.98 - 0.98 = 0$$

	操作系统		C++	计算机网络	数据结构与算法	数学分析	编译技术
0	True	False		False		True	False
1	True	False		False		True	True
2	False	True		False		False	True
3	False	False		False		False	False
4	False	False		False		False	False
5	False	False		False		False	False
6	False	False		False		False	False
7	False	False		False		True	True
8	False	False		False		False	False
9	False	True		False		False	False
10	False	True		False		False	True
11	False	False		False		False	False

图 7.4: 二值化后的学生的成绩

$$Gain(D, \text{数据结构与算法}) = 0.98 - 0.918 = 0.062$$

$$Gain(D, \text{数学分析}) = 0.98 - 0.975 = 0.05$$

$$\operatorname{argmax}_A \{0.05, 0.407, 0, 0.062, 0.05\} = C++$$

其中，我们可以看到《计算机网络》的信息增益最小，而《C++》的信息增益最大。显然，所有的同学都通过了《计算机网络》，因此《计算机网络》的成绩并没有区分能力，所以其信息增益也最小。而我们发现，《C++》挂科的同学其《编译技术》也挂科了，因此至少直观上看起来，《C++》的成绩的区分能力是最强的，所以其信息增益也最大。通过这样计算信息增益，我们就知道了第一个结点应该选用《C++》的成绩来作为划分属性。

于是我们可以得到如图7.5所示的结点。再计算每个子结点的信息熵，我们发现左子结点的信息熵为0，即数据都为同一类，我们只需将该结点标记为该类别即可。我们发现右子结点的信息熵较大，即右子结点的数据混乱程度较高，我们需要将右子结点继续分化。

将右子结点的数据集标记为 D_2 ，再次计算所有剩余属性的信息增益。

$$Gain(D_2, \text{操作系统}) = 0.76 - 0.683 = 0.077$$

$$Gain(D_2, \text{计算机网络}) = 0.76 - 0.76 = 0$$

$$Gain(D_2, \text{数据结构与算法}) = 0.76 - 0.306 = 0.454$$

$$Gain(D_2, \text{数学分析}) = 0.76 - 0.683 = 0.077$$

$$\operatorname{argmax}_A \{0.077, 0, 0.454, 0.077\} = \text{数据结构与算法}$$

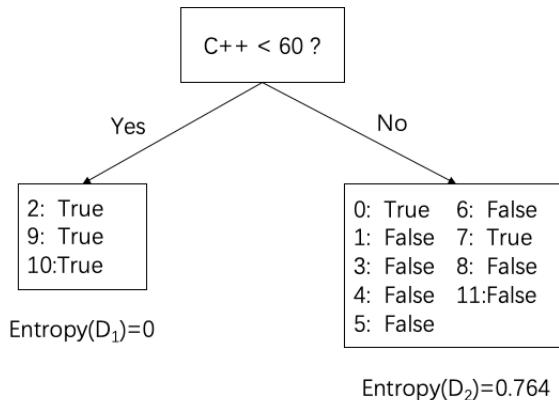


图 7.5：使用《C++》成绩作为第一个结点的划分属性

可以看到，对于数据集 D_2 ，《数据结构与算法》的成绩具有最好的划分效果，因此我们用《数据结构与算法》的成绩进一步划分子结点。划分结果如图7.6。

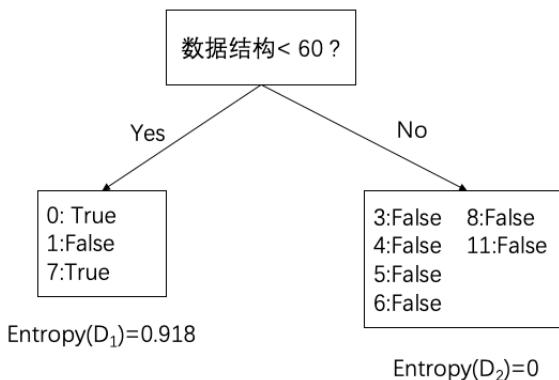


图 7.6：使用《数据结构与算法》成绩划分子结点

此时，右子结点的信息熵为 0，即所有数据都为同一类，因此我们可以将此结点标记为该类。对于左子结点，我们可以依据同样的方法继续划分，直到子结点的满足终止条件，如用完了所有的划分属性或信息熵小于设定的阈值等。这里由于左子结点包含的样本数量太少，再细分可能导致“过拟合”，因此我们停止左子结点的分裂，并标记为数量较多的类别。最终我们可以得到一颗如图7.7的决策树，输出结果的含义是编译技术是否会挂科。

思考：使用信息增益选择属性有没有什么问题？

7.1.3.3 增益率

由于信息增益挑选属性总是趋于选择取值更多的属性，因此，我们可以采用“增益率”来减少信息增益带来的影响。C4.5 算法则使用了增益率来选择属性。增益率的定义为(公式7.5)：

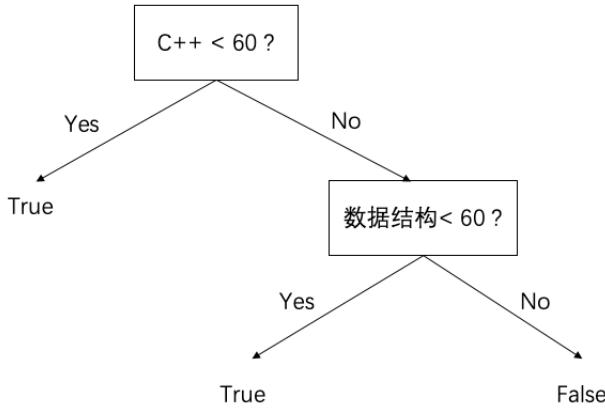


图 7.7: 最终生成的决策树

$$Gain_ratio(D, A) = \frac{Gain(D, A)}{IV(A)} \quad (7.5)$$

其中

$$IV(A) = - \sum_{j=1}^v \frac{|D_j|}{|D|} \log_2 \frac{|D_j|}{|D|} \quad (7.6)$$

$IV(A)$ 被称为属性 A 的“固有值”，反映了属性 A 的可能取值的数量，属性 A 可取值数越多， $IV(A)$ 越大。因此通过用信息增益除以 $IV(A)$ 可以减少属性可取值数量带来的影响。

但需要注意的是，增益率又倾向于选择可取值更少的属性，所以最终我们还需要结合信息增益来处理：我们可以先从候选划分属性中找出信息增益高于平均水平的属性，这样就能筛选出可取值较多且划分效果较好的属性，然后再从中选择出增益率最高的属性即可。

下面我们来解释一下为什么有了信息增益我们还需要使用增益率。

在使用信息增益挑选属性的时候，其实容易出现一个问题。我们可以先看一个例子：假如我们的训练集有 5 个样本，每个样本具有唯一的 ID 属性，且这个训练集的 ID 属性就会具有 5 个值。于是，我们如果使用 ID 属性划分该训练集，将产生 5 个子集，每个子集都仅包含一个样本，此时其纯度达到最大，即 $\sum_{j=1}^v \frac{|D_j|}{|D|} Entropy(D_j) = 0$ ，所以信息增益 $Gain(D, A) = Entropy(D)$ ，也达到了最大值。虽然 ID 属性的信息增益最大，但使用 ID 划分并没有意义，因为这不能让我们对新样本进行预测。

根据上述例子我们可以发现，信息增益挑选属性总是趋于选择取值更多的

属性，这就导致了容易选择一些无意义的属性，反而影响预测效果。为了解决这个问题，我们还需要考虑属性的取值数目，而增益率引入了“固有值”（公式7.6），便减少了可取值数量带来的影响。

7.1.3.4 基尼指数

CART 分类树算法使用了基尼指数来选择属性。

在了解基尼指数之前，我们需要先了解基尼值。

基尼值的定义为：

$$Gini(D) = \sum_{k=1}^K p_k(1 - p_k) = 1 - \sum_{k=1}^K p_k^2 \quad (7.7)$$

其中，数据集 D 有 K 个类， p_k 为第 k 类样本的频率。

$Gini(D)$ 指的是在数据集 D 中随机抽取的两个样本的类别不一致的概率，因此， $Gini(D)$ 越高意味着数据集 D 越混乱。

对于数据集 D ，属性 A 的基尼指数的定义为：

$$Gini_index(D, A) = \sum_{j=1}^v \frac{|D_j|}{|D|} Gini(D_j) \quad (7.8)$$

可见，基尼指数越小，数据集越纯净，划分效果也越好。因此，在实践中我们选择基尼指数最小的属性作为最优划分属性。

7.1.4 剪枝

为了防止决策树训练过程中出现“过拟合”，我们可以通过主动去掉一些分类效果不明显的分枝，从而提高决策树的泛化能力，这一过程称为剪枝。

剪枝分为预剪枝和后剪枝。

7.1.4.1 预剪枝

预剪枝是在生成决策树的过程中，对每一个结点先进行评估，若该结点不能提升泛化能力，则停止分化，生成叶子结点。

我们可以使用验证集来评估结点：对当前结点，选取样本最多的类别作为该结点的类别，在划分前我们可以用验证集可以得到一个精度；划分后，再用验证集计算精度，对比划分前后的精度即可评估结点的泛化能力是否得到提升。

7.1.4.2 后剪枝

后剪枝是先生成一棵完整的决策树，然后自底向上对决策树的每一个结点进行评估，若剪除分支后泛化能力得到提升，则剪除该分支。

常见的后剪枝算法有：错误率降低剪枝（REP）、悲观剪枝（PEP）、代价复杂度剪枝（CCP）、最小错误剪枝（MEP）。此为拓展话题，有兴趣的读者可以查阅相关文献。

7.1.5 小结

在这节里，我们学习了决策树算法。首先，我们需要明确决策树可以用来解决分类和回归问题，不过我们这里主要讨论用于分类的决策树。我们提到了如何根据训练集生成一棵决策树，也给出了相应的伪代码（算法14）。在决策树的实现过程中，我们需要重点关注的问题是如何从属性集里选择出最优的划分属性。由此，我们引入了信息熵的概念（公式7.1），通过信息熵，我们可以计算出每个属性的信息增益（公式7.3）。有了信息增益，我们便可以挑选出较优的划分属性了。但是，信息增益总是倾向于选择可取值更多的属性，为了解决这个问题，我们引入了增益率（公式7.5）。由于增益率总是倾向于选择可取值更少的属性，因此在使用增益率挑选属性时，我们通常不直接选取增益率最大的属性，而是先从候选划分属性中先找出信息增益高于平均水平的属性，然后再从中挑选出增益率最高的属性。

其中，信息增益和增益率是本节中最重要的两个公式，望各位读者能够熟练掌握。

1. 信息增益：

$$Gain(D, A) = Entropy(D) - \sum_{j=1}^v \frac{|D_j|}{|D|} Entropy(D_j)$$

2. 增益率：

$$Gain_ratio(D, A) = \frac{Gain(D, A)}{IV(A)}$$

其中，

$$IV(A) = - \sum_{j=1}^v \frac{|D_j|}{|D|} \log_2 \frac{|D_j|}{|D|}$$

7.2 随机森林

7.2.1 Bagging 算法

Bagging 是并行集成学习的一种方法。Bagging 算法描述如算法15。

Algorithm 15: Bagging

输入: 训练集 $D = (x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$;
输出: $H(x) = \arg \max \sum_{m=1}^M I(h_m(x) = y), y \in Y$

- 1: 基学习算法: L
- 2: 学习轮数: M
- 3: **for** $m = 1, 2, \dots, M$ **do**
- 4: $h_m(x) = L(D, D_{bs})$;
- 5: **end for**

Bagging 采用自助采样法 (Bootstrap sampling) 来产生训练集。Bagging 集成要获得较好的泛化效果，各基学习器的独立性要尽可能的大，一种可能是将训练集分为若干个互不相同的子集，每个基学习器在对应子集上训练，从而保证较高的差异。但由于训练样本分为多个子集，每个基学习器只用到一小部分训练数据，从而无法保证基学习器的训练效果。因而利用自主采样，每次从数据集中采取的样本将被放回，通过采样，我们得到的训练集中含有重复的数据，也有部分数据从未被采样，由下式可知，

$$\lim_{m \rightarrow +\infty} \left(1 - \frac{1}{m}\right)^m \rightarrow \frac{1}{e} \approx 0.368$$

初始数据集中约有 63.2% 被选中，另外剩余 36.8% 的样本数据可用作验证集来对泛化性能进行包外估计 (out-of-bag estimate)。为使用包外估计，需记录每个基学习器所使用的训练样本，对未使用 x 样本的基分类器进行预测，即

$$H_{oob}(x) = \arg \max \sum_{m=1}^M I(h_m(x) = y), y \in Y$$

使用包外估计可对决策树进行辅助剪枝，对易受扰动的神经网络减小过拟合风险。

7.2.2 随机森林

俗话说，“三个臭皮匠顶个诸葛亮”，集体的力量总是大于个体的，随机森林就是这样一个利用了集体的力量的算法。

随机森林是 Bagging 算法的变体，它与 Bagging 算法的差别在于：对于随机森林，在每个基决策树的结点生成时，不再是选择划分效果最优的属性，而是先



随机选取 m 个属性，然后在这 m 个属性中选择划分效果最优的属性。因此，假设训练集有 d 个属性，当 $m = d$ 时，基决策树的生成与传统决策树的生成等效；当 $m = 1$ 时，就是随机选择一个属性作为结点的划分属性。

构建随机森林一般有以下几个步骤：

1. 确定基决策树的数量 N 。我们可以通过调整树的规模来调整随机森林的预测效果。
 2. 随机采样。利用自助采样法（bootstrap sampling）我们可以将样本数据分成训练集和袋外数据（Out Of Bag）两部分。
 3. 随机选择特征生成决策树。我们从属性集中随机选择 m 个特征，从 m 个特征中选择出划分效果最优的属性，生成子树。重复这一过程直到满足我们设定的停止条件，得到一棵基决策树。
 4. 估计袋外误差（OOB Error）。我们可以利用袋外数据对生成的决策树计算出袋外误差，得到这棵树的无偏估计。
 5. 重复过程 2 到 4，直到得到有 N 棵基决策树的随机森林。

图7.8是随机森林的构建流程图。

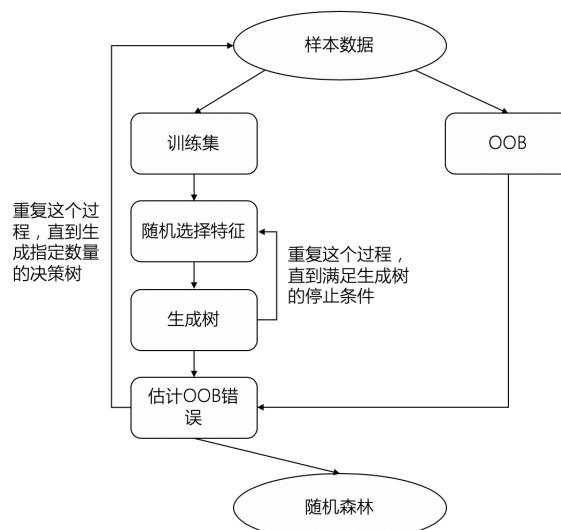


图 7.8: 随机森林的构建流程

由于随机森林结合了多棵决策树，因此它能够一定程度地解决决策树出现过拟合的情况，且具有一定的抗噪声的能力。因为随机森林可以选择大规模的决策树，甚至可以比样本数还多，因此它可以获取更多的数据以减少样本的估计偏差。在性能方面，随机森林要优于 Bagging，因为在构建基决策树时，Bagging 需要考虑所有属性的划分效果，而随机森林仅需要考虑随机选取的 m 个属性。

沿用决策树章节的学生成绩数据，我们可以根据图7.8的流程构建随机森林。在这个例子中，我们构建一个 $N = 3$ 的随机森林。首先随机采样，将样本分成训练集和 OOB 部分，如图7.9

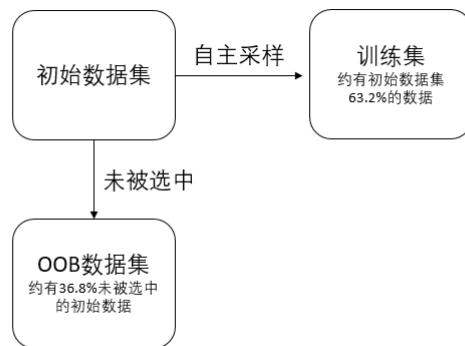


图 7.9: 构建的第一棵决策树

然后随机选择属性生成决策树，在这里我们随机选取了《C++》的成绩作为划分属性。按照决策树的生成策略，我们可以生成一棵如图7.10的决策树。

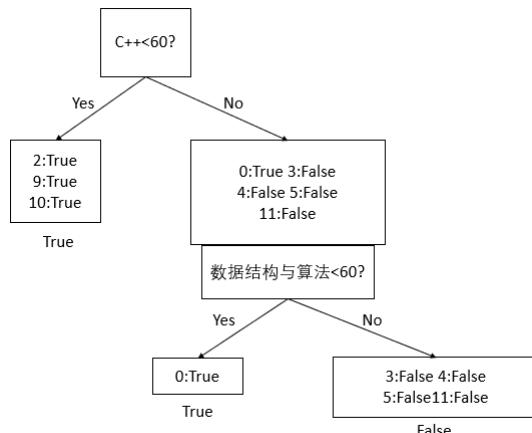


图 7.10: 构建的第一棵决策树

接着我们对这棵树进行 OOB 估计。

重复以上过程，我们最终生成了 3 棵决策树，这就构成了随机森林。

在进行预测的时候，我们会将样本分别放入这 3 棵决策树，得到 3 个输出结果，将出现最多的结果作为最终结果输出。

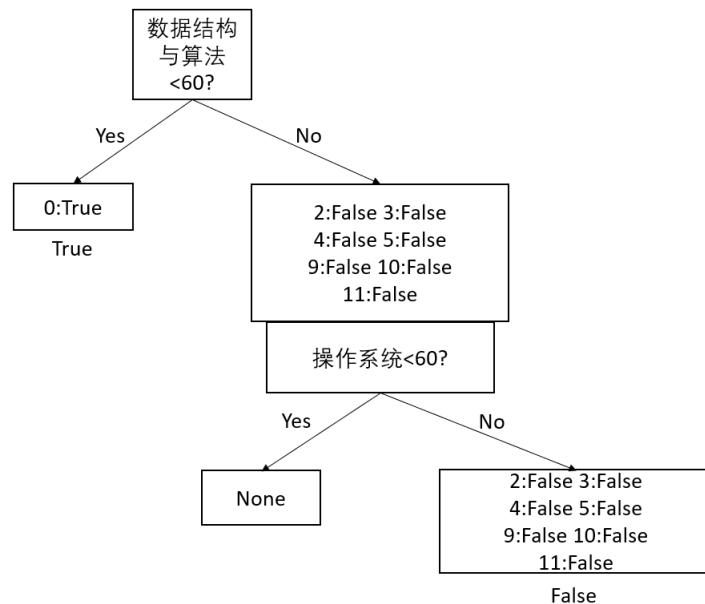


图 7.11: 第二棵决策树

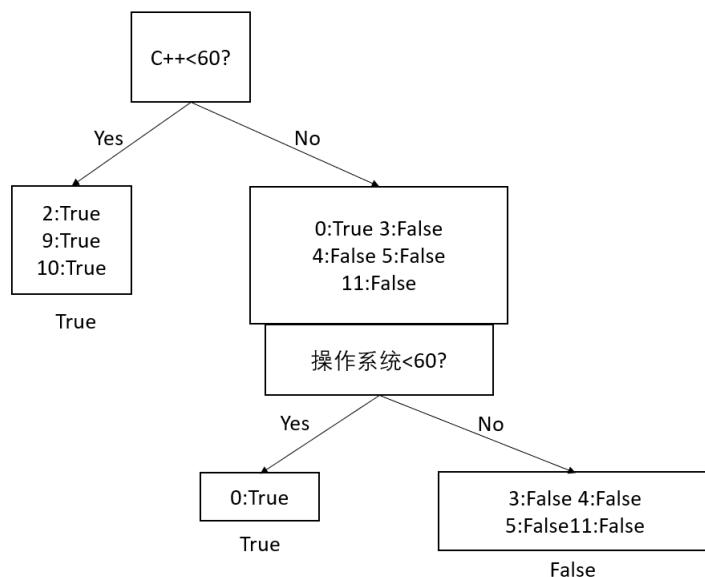


图 7.12: 第三棵决策树

第 8 章 AdaBoost、梯度提升决策树和 XGBoost

8.1 AdaBoost

AdaBoost(Adaptive Boosting) 是集成学习中 Boosting 算法中最为著名的代表之一. Boosting 算法的特征是个体学习器间存在强依赖关系，个体学习器以串行化方式生成.

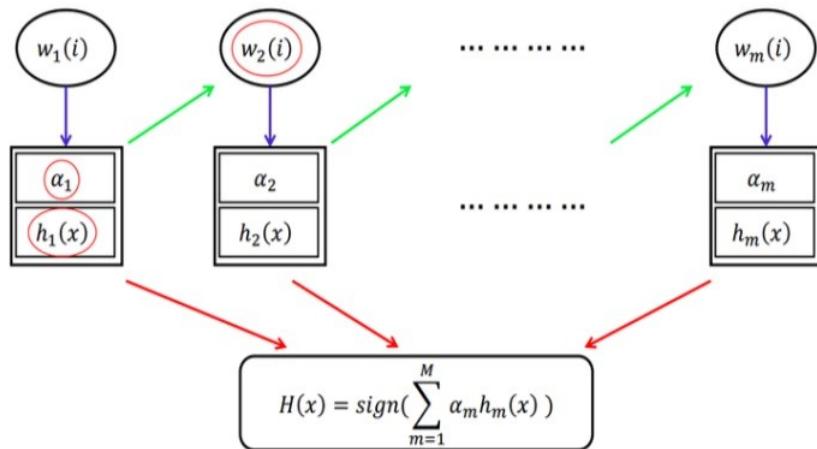


图 8.1: AdaBoost 的串行化生成

Boosting 算法首先依据初始训练集训练出一个基学习器，再根据基学习器的表现对样本分布进行调整，对基学习器做错的样本给予更高的关注和权重，即残差逼近的思想. 以减小偏差.

之后根据调整后的训练样本训练下一个基学习器，如此反复，直到达到预先指定的基学习器数目，再依据基学习器的表现进行结合，从而形成一个具有较好表现的强学习器.

Adaboost 算法的线性加权模型为:

$$H(x) = \sum_{m=1}^M \alpha_m h_m(x)$$

式中 $h_m(x)$ 为基学习器， α_m 为基学习器的权重， $H(x)$ 则为基学习器的线形组合.

AdaBoost 算法的基本流程如下：

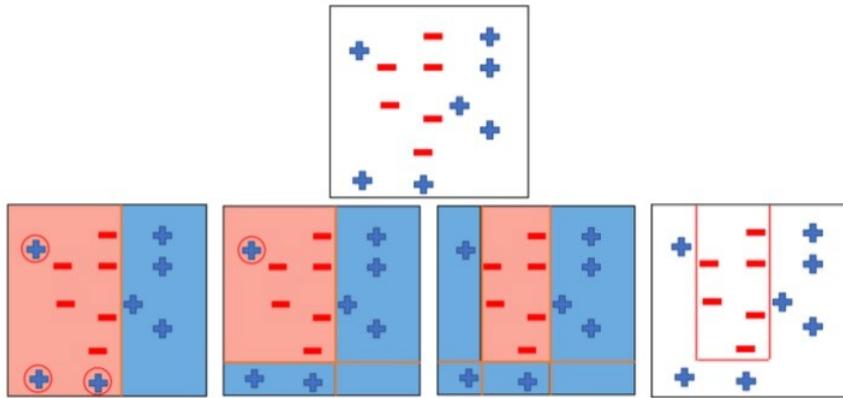


图 8.2: 分错样本给予更高权重

Algorithm 16: AdaBoost

```

1 输入: 训练集  $D = (x_1, y_1), (x_2, y_2), \dots, (x_m, y_m)$ , 其中  $x_i \in X, y_i \in \{-1, 1\}$ ;
2 初始化: 样本分布  $\omega_1(i) = \frac{1}{n}$ 
3 基分类器:  $\mathcal{L}$ 
4 学习轮数:  $M$ 
1: for  $m = 1, 2, \dots, M$  do
2:    $h_m(x) = \mathcal{L}(D, \omega_m)$ 
3:    $\epsilon_m = \sum_{i=1}^n \omega_m(i) \mathbb{I}(h_m(x_i) \neq y_i)$ 
4:   if  $\epsilon > 0.5$  then
5:     break
6:   end if
7:   end
8:    $\alpha_m = \frac{1}{2} \log \frac{1-\epsilon_m}{\epsilon_m}$ 
9:    $\omega_{m+1}(i) = \frac{\omega_m(i)}{\sum_{i=1}^n \omega_m(i) e^{-\alpha_m y_i h_m(x_i)}} e^{-\alpha_m y_i h_m(x_i)}$ 
10: end for
11: end
输出:  $H(x) = \sum_{m=1}^M \alpha_m h_m(x)$ 

```

若 $H(x)$ 可用来最小化指数损失函数

$$l(H|D) = E_{x \sim D}[e^{-f(x)H(x)}]$$

则对 H 求偏导有

$$\frac{\partial l(H|D)}{\partial H(x)} = -e^{-H(x)} P(f(x) = 1|x) + e^{H(x)} P(f(x) = -1|x)$$

令上式为零, 可解得

$$H(x) = \frac{1}{2} \ln \frac{P(f(x) = 1|x)}{P(f(x) = -1|x)}$$

于是有

$$\begin{aligned} sgn(H(x)) &= sgn\left(\frac{1}{2} \ln \frac{P(f(x) = 1|x)}{P(f(x) = -1|x)}\right) \\ &= \begin{cases} 1, & P(f(x) = 1|x) > P(f(x) = -1|x) \\ -1, & P(f(x) = 1|x) < P(f(x) = -1|x) \end{cases} \\ &= \arg \max P(f(x) = y|x), y \in \{-1, 1\} \end{aligned}$$

从上式可以看出，若该损失函数最小化，则分类错误率也将最小化。因此， $H(x)$ 可以最小化损失函数，且指数损失函数是 0/1 原始损失函数的一致的替代损失函数。由于指数损失函数具有更好的数学性质，我们将其作为 0/1 原始损失函数的替代优化目标。AdaBoost 算法中，第一个基分类器是由原始分布产生的，依据第一个基分类器的表现对原始分布进行调整，进而迭代地产生下一个分类器以及其所占权重。下一个分类器的产生也应使得损失函数最小化，因此有

$$\begin{aligned} l(\alpha_m h_m | D_m) &= E_{x \sim D_m} \left[e^{-f(x)\alpha_m h_m(x)} \right] \\ &= E_{x \sim D_m} [e^{-\alpha_m} \mathbb{I}(f(x) = h_m(x)) + e^{\alpha_m} \mathbb{I}(f(x) \neq h_m(x))] \\ &= e^{-\alpha_m} P_{x \sim D_m}(f(x) = h_m(x)) + e^{\alpha_m} P_{x \sim D_m}(f(x) \neq h_m(x)) \\ &= e^{-\alpha_m}(1 - \epsilon_m) + e^{\alpha_m} \epsilon_m \\ \frac{\partial l(\alpha_m h_m | D_m)}{\partial \alpha_m} &= -e^{-\alpha_m}(1 - \epsilon_m) + e^{\alpha_m} \epsilon_m \end{aligned}$$

令偏导为 0，我们得到了新产生的基分类器所应占据的权重，即分类器权重更新公式：

$$\alpha_m = \frac{1}{2} \ln \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$$

AdaBoost 算法在获得 H_{m-1} 之后样本分布将进行调整，使下一轮的基学习器 h_m 能纠正 H_{m-1} 的一些错误，理想的 h_m 能纠正 H_{m-1} 的全部错误，即最小化式

$$\begin{aligned} l(H_{m-1} + h_m | D) &= E_{x \sim D} \left[e^{-f(x)(H_{m-1}(x) + h_m(x))} \right] \\ &= E_{x \sim D} \left[e^{-f(x)H_{m-1}(x)} e^{-f(x)h_m(x)} \right] \end{aligned}$$

利用 $e^{-f(x)h_m(x)}$ 的泰勒展开式，可近似为

$$\begin{aligned} l(H_{m-1} + h_m | D) &\simeq E_{x \sim D} \left[e^{-f(x)H_{m-1}(x)} \left(1 - f(x)h_m(x) + \frac{f^2(x)h_m^2(x)}{2} \right) \right] \\ &= E_{x \sim D} \left[e^{-f(x)H_{m-1}(x)} \left(1 - f(x)h_m(x) + \frac{1}{2} \right) \right] \end{aligned}$$

因此有

$$\begin{aligned}
 h_m(x) &= \arg \min l(H_{m-1} + h|D) \\
 &= \arg \min E_{x \sim D} \left[e^{-f(x)H_{m-1}(x)} \left(1 - f(x)h(x) + \frac{1}{2} \right) \right] \\
 &= \arg \max E_{x \sim D} \left[e^{-f(x)H_{m-1}(x)} f(x)h(x) \right] \\
 &= \arg \max E_{x \sim D} \left[\frac{e^{-f(x)H_{m-1}(x)}}{\sum_{x \sim D} [e^{-f(x)H_{m-1}(x)}]} f(x)h(x) \right]
 \end{aligned}$$

令 D_m 表示一个分布，即

$$D_m(x) = \frac{D(x)e^{-f(x)H_{m-1}(x)}}{\sum_{x \sim D} [e^{-f(x)H_{m-1}(x)}]}$$

于是有

$$\begin{aligned}
 h_m(x) &= \arg \max E_{x \sim D} \left[\frac{e^{-f(x)H_{m-1}(x)}}{\sum_{x \sim D} [e^{-f(x)H_{m-1}(x)}]} f(x)h(x) \right] \\
 &= \arg \max E_{x \sim D_m} [f(x)h(x)]
 \end{aligned}$$

又有

$$f(x)h(x) = 1 - 2\mathbb{I}(f(x) \neq h(x))$$

因此理想的基学习器

$$h_m(x) = \arg \min E_{x \sim D_m} [\mathbb{I}(f(x) \neq h(x))]$$

同时我们得到 D_{m+1} 与 D_m 的关系，即样本分布的更新公式

$$\begin{aligned}
 D_{m+1}(x) &= \frac{D(x)e^{f(x)H_m(x)}}{\sum_{x \sim D} [e^{-f(x)H_m(x)}]} \\
 &= \frac{D(x)e^{-f(x)H_{m-1}(x)}e^{-f(x)\alpha_m h_m(x)}}{\sum_{x \sim D} [e^{-f(x)H_m(x)}]} \\
 &= D_m(x)e^{-f(x)\alpha_m h_m(x)} \frac{\sum_{x \sim D} [e^{-f(x)H_{m-1}(x)}]}{\sum_{x \sim D} [e^{-f(x)H_m(x)}]}
 \end{aligned}$$

至此，我们推出了 Adaboost 的核心公式，基学习器样本权重更新公式以及样本分布的更新公式。值得注意的是，一旦产生准确率低于 0.5 的基学习器，当前基学习器即被抛弃，且学习过程停止。这可能使得初始设置的学习轮数远未达到，导致集成学习器由于较少的基学习器而性能不佳。此时可对分布样本进行重新采样，依据新采样分布训练基学习器，从而实现继续迭代，直到到达预设的 M 轮。



8.2 梯度提升决策树

8.2.1 分类与回归树

我们之前已经学习了决策树。决策树常常被用于分类任务，但是，有没有可能将决策树用作回归任务呢？

在上一章的决策树例子中，决策树的每个叶子节点都有一个输出的分类。如果叶子节点的输出值不再是某个分类，而是一个实数值，就可以将决策树用于回归任务。实际上，一棵决策树就是将输入空间划分成了许多区域，每个区域对应一个输出值。如果输出值是一个分类，就可以被用作分类任务；如果是实数值，则可被用作回归任务。用作回归任务时，只要划分出的区域足够小，就可以用区域中样本点的平均输出值近似地代替真实的输出值。

Breiman 等在 1984 年提出的分类与回归树（Classification And Regression Tree, CART）就是一种像上面一样的决策树，分为分类树和回归树两种。CART 的学习算法有生成和剪枝两步。

8.2.1.1 分类树生成

首先引入基尼指数（Gini index）作为选择最优二分的指标。基尼指数反映了数据集中随机抽取的两个样本的分类不一致的概率，其定义如下：

$$Gini(D) = \sum_{k=1}^K p_k(1 - p_k) = 1 - \sum_{k=1}^K p_k^2 \quad (8.1)$$

其中，数据集 D 有 K 个类，第 k 类样本的频率为 p_k 。

特别地，当数据集 D 中只有两类且其中一类的频率为 p 时，

$$Gini(D) = 2p(1 - p) \quad (8.2)$$

若数据集 D 根据特征 A 是否有 $A = a$ 被分成子集 E, F ，则其基尼系数定义为

$$Gini(D, a) = \frac{|E|}{|D|} Gini(E) + \frac{|F|}{|D|} Gini(F) \quad (8.3)$$

其中 $|D|$ 表示集合 D 的元素个数。

8.2.1.2 回归树生成

假设数据集的输入输出变量均为连续值，设数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ 设输入空间被划分为区域 R_1, R_2, \dots, R_m ，每个区域 R_i 有一个固定的输出值 c_i ，

Algorithm 17: CART 分类树生成算法

输入：训练集 D , 停止计算的条件

输出：一棵 CART 分类树

- 1: 新建节点 N , 其左右子节点分别为 L, R
- 2: 对每个特征 S 的可能取值 a , 根据是否有 $A = a$ 将数据集 D 分为两个子集, 并计算分割后的基尼系数 $Gini(D, a)$
- 3: 选择基尼系数最小的分法, 得到子集 E, F 。
- 4: L =CART 分类树生成算法 (E , 停止计算的条件)
- 5: R =CART 分类树生成算法 (F , 停止计算的条件)
- 6: 返回 N

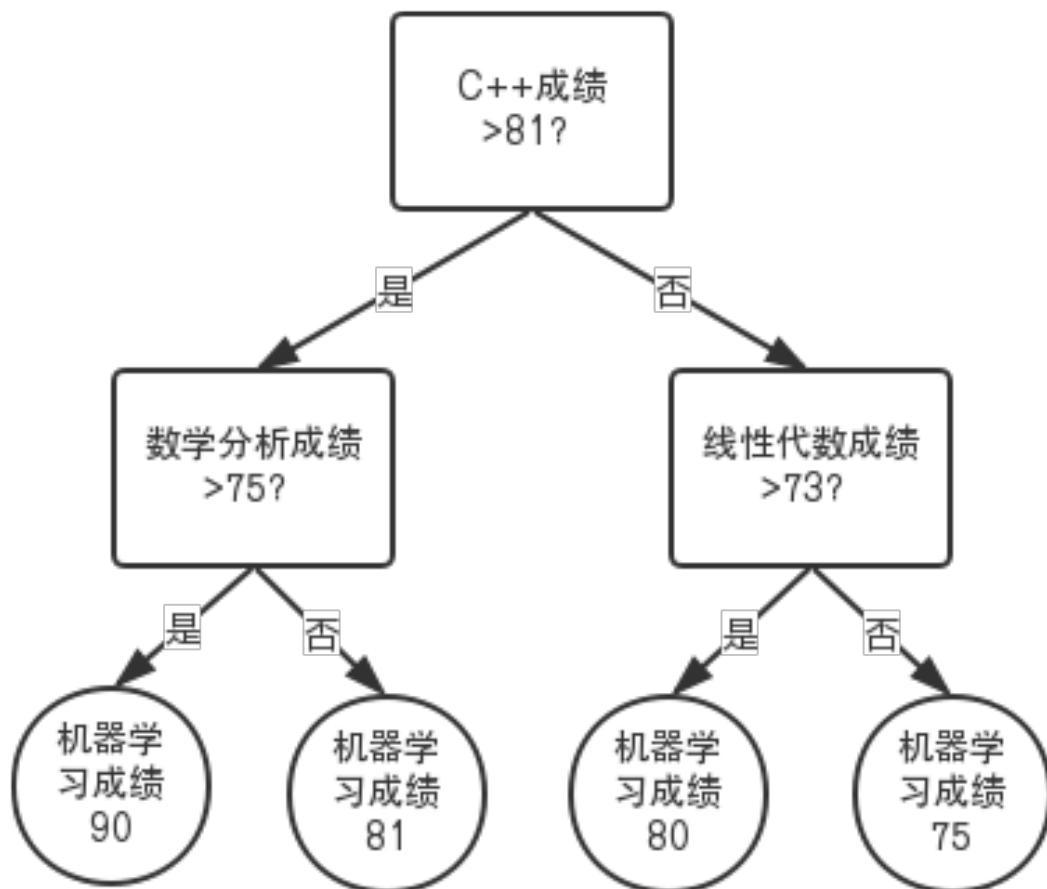


图 8.3: 回归树示例

则回归树可表示为

$$f(x) = \sum_{i=1}^m c_i I(x \in R_i) \quad (8.4)$$

回归树使用平方误差 $\sum_{x_j \in R_i} (y_j - f(x_j))^2$ 作为每个区域 R_i 的预测误差。若要令预测误差最小，易知，在区域 R_i 上， c_i 为 R_i 中所有样本点的平均输出值，即

$$c_i = \text{mean}(y_j | x_j \in R_i) \quad (8.5)$$

显然， R_i 上的最小预测误差为输出值的方差。

在回归树生成的每一步上，为了对输入空间进行划分，将数据集分成两个子集，需要选择第 k 个变量 $x^{(k)}$ 及其取值 s ，作为切分变量（splitting variable）和切分点（splitting point），将输入空间分割为两个区域

$$L(k, s) = \{x | x^{(k)} \leq s\} \quad (8.6)$$

和

$$R(k, s) = \{x | x^{(k)} > s\} \quad (8.7)$$

求解

$$(k_*, s_*) = \arg \min_{(k,s)} (\text{var}(y_i | x_i \in L(k, s)) + \text{var}(y_i | x_i \in R(k, s))) \quad (8.8)$$

可得当前输入空间的最优切分变量和最优切分点。

对每一个叶子节点重复以上步骤，直到满足停止条件为止，即可生成一棵回归树。

Algorithm 18: CART 回归树生成算法

输入： 训练集 D ，停止条件

输出： 一棵 CART 回归树

1: 遍历 $k = 1, 2, \dots, n$ ，求解

$(k_*, s_*) = \arg \min_{(k,s)} (\text{var}(y_i | x_i \in L(k, s)) + \text{var}(y_i | x_i \in R(k, s)))$ ，将训练集 D 分成两个子集 E, F 。

2: 重复以上步骤，直到满足停止条件为止。

8.2.1.3 剪枝

剪枝算法由两步组成：第一步，将 CART 决策树 T_0 从叶子节点开始不断剪枝，直到根节点，每剪一次输出一棵决策树，形成一个决策树序列 $\{T_0, T_1, \dots, T_n\}$ ；然后通过交叉验证法用独立的验证集选出最优的决策树。



给出子树的损失函数

$$C_\alpha(T) = C(T) + \alpha|T| \quad (8.9)$$

其中, T 为任意子树, $C(T)$ 为对训练数据的预测误差 (如基尼指数), $|T|$ 为子树的叶子节点个数, $\alpha \geq 0$ 为参数, 用于权衡拟合程度和模型复杂度。

对于内部节点 t , 剪去以 t 为根节点的子树 T_t (会留下节点) 的临界值 α 可由 $C_\alpha(T_t) = C_\alpha(t)$ 解得。可知

$$\alpha = \frac{C(t) - C(T_t)}{|T_t| - 1} \quad (8.10)$$

对于每个内部节点 t , 设

$$g(t) = \frac{C(t) - C(T_t)}{|T_t| - 1} \quad (8.11)$$

作为 α 的不同取值下剪枝的依据。当 $\alpha \geq g(t)$ 时, 即可对 t 剪枝。

求出每个内部节点的 $g(t)$, 每次取最小的 $g(t)$ 值进行剪枝, 得到剪枝后的决策树序列 $\{T_1, T_2, \dots, T_n\}$ 。

生成决策树序列 $\{T_0, T_1, \dots, T_n\}$ 后, 对每棵树用验证集测试, 选出平方误差或基尼指数最小的作为最优决策树 T_α 。

Algorithm 19: CART 剪枝算法

输入: CART 算法生成的决策树 T_0 , 测试集 D_T

输出: 最优决策树 T_α

- 1: 计算出每个内部节点的 $g(t)$ 值
 - 2: 根据从小到大的 $g(t)$ 值对 T_i 进行剪枝, 得到 T_{i+1} , 直到得到决策树序列 $\{T_0, T_1, \dots, T_n\}$
 - 3: 使用测试集 D_T 选出最优决策树 T_α
-

8.2.2 梯度提升 (Gradient Boosting, GB)

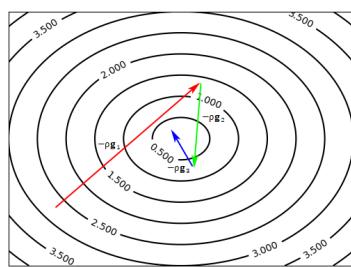


图 8.4: 梯度下降

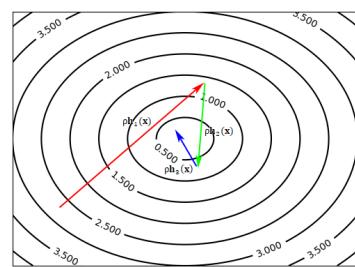


图 8.5: 梯度提升

在梯度下降法中, 通过将变量往损失函数梯度的负方向不断修正, 我们最终

得到了数值解。那么，如果我们将弱学习器得到的预测结果也往损失函数梯度的负方向不断修正，是不是也能得到一个更接近真实值的预测结果呢？

但是，梯度下降法要修正的是一个数值解，其在算法结束后就不会再改变，是一个常量；而梯度提升则需要考虑到不同的输入值，要修正的是一个与输入值有关的变量。所以，这里的梯度是要随着输入值而改变的。如果可以确定修正预测结果的梯度与输入值相关，我们就可以用弱学习器预测不同输入值下的梯度。

对于损失函数 $l(y, \hat{y})$ ，预测值和真实值都是与 x 相关的，所以其梯度也是与 x 相关的。

设有数据集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$ 和单个损失函数 $l(y, \hat{y})$ 。不妨设初始预测函数

$$F_0(x) = 0 \quad (8.12)$$

考虑 $j = 1, 2, \dots, m$ ，对于第 j 步，上一步得到预测函数 $F_{j-1}(x)$ ，使用函数 $f(x) = F_{j-1}(x)$ 预测 y 的损失为

$$L_f = \sum_{i=1}^n l(y_i, f(x_i)) \quad (8.13)$$

求上式的极小值，可以看作 $\hat{u} = \arg \min_u L(u)$ 。其中，

$$u = (f(x_1), f(x_2), \dots, f(x_n)) \quad (8.14)$$

$$L(u) = \sum_{i=1}^n l(y_i, u^{(i)}) \quad (8.15)$$

$u^{(i)}$ 为 u 的第 i 维分量。

求梯度得

$$\frac{\partial L}{\partial u} = (g_1, g_2, \dots, g_n) \quad (8.16)$$

其中，

$$g_i = \left. \frac{\partial l}{\partial \hat{y}} \right|_{\hat{y}=u^{(i)}, y=y_i} \quad (8.17)$$

使用弱学习器 $h_j(x)$ 学习负梯度分量 $-g_i$ 关于 x_i 的变化，得 $-g = h_j(x)$ 。然后，求参数

$$\rho_j = \arg \min_\rho (u + \rho h) \quad (8.18)$$

其中，

$$h = (h_j(x_1), h_j(x_2), \dots, h_j(x_n)) \quad (8.19)$$

得到该步的预测函数

$$F_j(x) = F_{j-1}(x) + \rho_j h_j(x) \quad (8.20)$$

实际上，梯度提升可以看做各个弱学习器的线性组合。

Algorithm 20: 梯度提升算法

输入： 训练集 $D = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
 输出： 梯度提升的一个学习器

- 1: 令 $F_0(x) = 0$
- 2: **for** $j = 1, 2, \dots, m$ **do**
- 3: $u = (F_{j-1}(x_1), F_{j-1}(x_2), \dots, F_{j-1}(x_n))$
- 4: **for** $i=1,2,\dots,n$ **do**
- 5: $g_i = \frac{\partial l}{\partial \hat{y}} \Big|_{\hat{y}=u(i), y=y_i}$
- 6: **end for**
- 7: 用数据集 $\{(x_1, -g_1), (x_2, -g_2), \dots, (x_n, -g_n)\}$ 训练弱学习器 $h_j(x)$
- 8: $h = (h_j(x_1), h_j(x_2), \dots, h_j(x_n))$
- 9: 求 $\rho_j = \arg \min_{\rho} (u + \rho h)$
- 10: 得到 $F_j(x) = F_{j-1}(x) + \rho_j h_j(x)$
- 11: **end for**
- 12: 输出学习器 $F_m(x) = \sum_{j=1}^m \rho_j h_j(x)$

特别地，令 $\rho_j = 1$, $l(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$, 弱学习器为回归树，得到提升树（boosting tree）算法。此时， g_i 为残差。

8.2.3 梯度提升决策树（Gradient Boosting Decision Tree, GBDT）

使用回归树作为梯度提升中的弱学习器，即可得到梯度提升决策树。

具体算法可参照梯度提升算法。

8.3 XGBoost

XGBoost 是一个关于增强树的机器学习系统，旨在实现高效、灵活、可拓展。XGBoost 的实现是基于 GBDT（梯度提升树）的改进版本，可以快速准确的解决许多数据科学问题。在著名机器学习竞赛 Kaggle 中，2015 年度的 29 个获奖算法中，17 个使用了 XGBoost，该算法的威力可见一斑。XGBoost 之所以如此成功，原因在于使用了能够有效提取复杂数据的模型，可以进行并行计算，以及算法的在多种场景的可拓展性。本章第一节旨在介绍理解 XGBoost 所需要的理论知识储备，第二节描述配合例子描述 XGBoost 的基本思想，第三节论证 XGBoost 的算法实现以及数学推导。

8.3.1 理论储备

XGBoost 演化过程 首先，被提出的是决策树，在将决策树与 Bagging 方法结合之后，随机森林便走进了人们的视线，在随机森林方法中加入 boosting 方法，便得到了梯度提升方法 (Gradient Boosting)，最后，基于 Gradient Boosting 进行改进，我们便得到了 XGBoost。

Bagging 算法 对于一个大小为 n 的训练集 \mathcal{T} ，Bagging 算法从中均匀、有放回地选出 m 个大小为 n' 的子集 \mathcal{T}_i ，作为新的训练集。在这 m 个训练集上使用分类、回归等算法，则可得到 m 个模型，再通过取平均值，取多数票等方法，即可得到 Bagging 的结果。

Boosting 算法 提升方法基于这样一种思想：对于一个复杂的问题，将多个专家的判断进行综合考虑进行的判断，往往好于其中任何一个专家的单独判断。提升方法正所谓“三个臭皮匠顶个诸葛亮”。提升方法的详细实现，可以参考 AdaBoost 的实现方式。

决策树以及树的分裂 有一天小明突发奇想，使用刚学习的决策树判断是否应该出门学习，应该怎么做？

首先，小明可能想到，观察天气是否下雨，若没有下雨，直接出门学习即可，若下雨，跳往第二步。第二步，观察天气是否为恶劣天气，若窗外仅仅飘着小雨，那么直接打伞上课。倘若窗外雷声阵阵，一副黑云压城城欲摧的架势，又或者拳头大的冰雹毫不客气地直落九天，为了安全着想，小明选择待在寝室学习。决策过程做图如8.6：

在构建一个决策树的时候，我们需要找到最合适的特征来进行分裂，分裂的算法包括 ID3 算法、C4.5 的生成算法。进一步，为防止过拟合，产生太复杂的决策树，我们引入剪枝，对已生成的决策树进行简化，详细实现请参照决策树章节。

GBDT GBDT 的原理即使用 Boosting 方法，将所有的弱分类器的结果相加得到预测值，同时用下一个弱分类器去拟合误差函数对预测值得残差，GBDT 与 XGBoost 的原理有很大的相似之处，GBDT 的具体实现，详见前节。

CART Xgboost 就是由很多 CART (Classification And Regression Tree) 树集成，CART 树指分类树和回归树的组合，简单的说分类树意味着标签是离散的，回归树意味着标签是连续的。

8.3.2 XGBoost 基本思想

预测方法



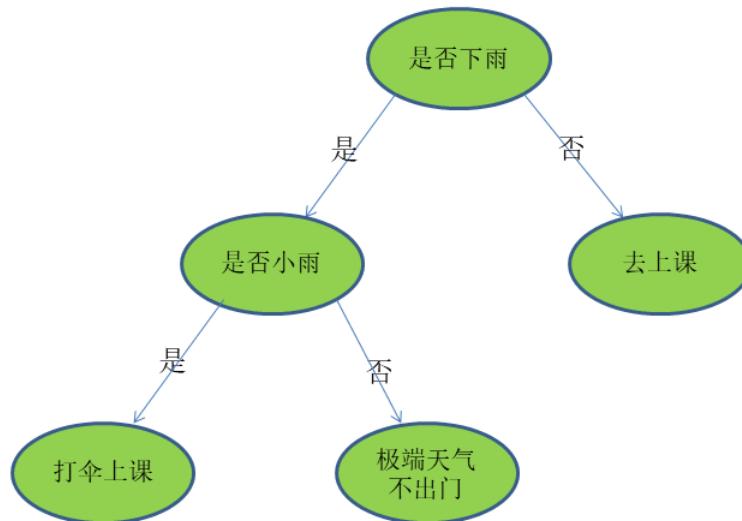


图 8.6: 判断是否出门上学

伴随着炎暑的消散，夏季清凉的泳池、香甜可口的冰淇淋、沁人心脾的西瓜逐渐远去。新的学期即将开始，胸怀壮志的小明和舍友们遇到了一个问题，究竟要不要选择“机器学习”这一门课呢？

根据以往的经验，机器学习这门课知识涉及面广，每章节深刻讲述核心原理，学习难度相对较高，同时，机器学习是当今最火热的研究方向，如果能够扎实掌握本课程知识，无论对是日后深耕学术潜心研究理论，或者投身工业界将产品落地，都能打下良好的基础。犹豫的小明面对者选课系统，紧握着鼠标，迟迟不敢点下“确定选课”按钮。

基于 xgboost 算法，我们为小明宿舍提供了一个解决方案，判断“机器学习”课程对。首先我们构造了两棵 CART 树，在第一棵树中，我们首先判断是否为计算机或数学相关专业的学生，若不是，分值为 -1，若是，再判断是计算机相关专业学生还是数学相关专业学生，若是计算机相关专业，分值为 +2，若是数学相关专业，分值为 +1。在第二棵树中，判断高数、线代、概率论三门课平均成绩是否及格，如果及格，分值为 +0.9，若不及格，分值为 -0.9。最后将对两棵树的判断结果相加，得到最终的判断数值。具体例子详见图8.7.

由图8.7可得知，小明的分值为 2.9，小黑的分值为 -1.9，从而推断出小明相比小黑更适合机器学习课程。

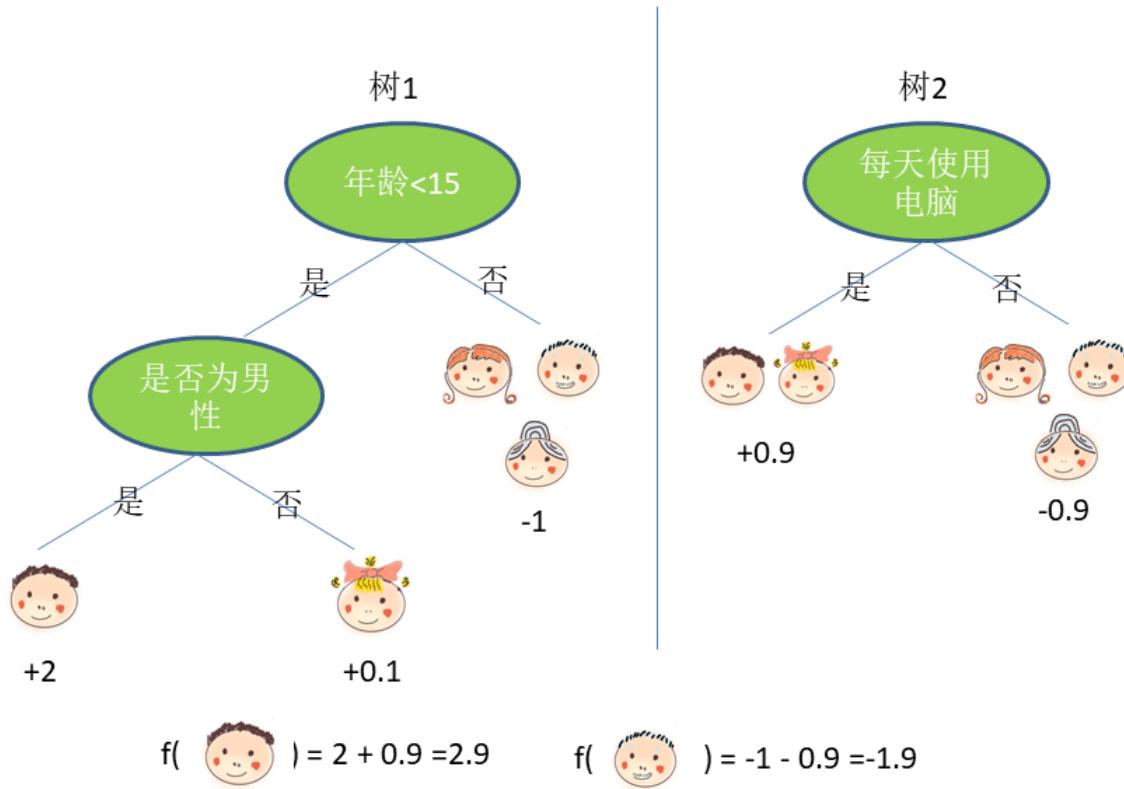


图 8.7: 总预测值为每棵树的预测值之和

训练方式实际上，训练的目的就是希望预测值尽可能接近真实值，我们的思路与 GBDT 相似，针对预测值与真实值之间的损失函数进行训练，获得最佳的 CART 树，以便达到 XGBoost 的最终效果。

详细步骤如下，第一步，通过特征分裂生成一棵新的树，去拟合上一次预测值与实际值之间的残差。第二步，在当前树集合中加入第一步生成树，对树集合的 K 棵树，将样本映射到每一棵树的某个叶节点上，分别获得一个值。第三步，将第二步每个叶节点上的值相加，结果就是该样本的预测值，每一次迭代，都在现有树的基础上，增加一棵树去拟合前面树的预测结果与真实值之间的残差，直到触发停止条件。

分裂方式在预测方法和训练方式中，我们提到 XGBoost 算法需要多棵树共同组合预测与训练，如果知道树的结构，那么便可以获取该结构下最好的分数。那么如何生成这些树呢，更确切一点，如何对每棵树进行分裂呢？

近似算法贪心枚举算法十分强大，该算法能够贪心地枚举所有可能的分裂方式。然而当数据不能被完全载入存储器时，或者在分布集中该算法就不够有效。于

是我们引入近似算法。

在近似算法中，对特征分布提前预设一个百分比，根据该百分比划分特征，再通过划分的特征进行计算，判断最大增益，从而找出最好的分裂特征。

8.3.3 XGBoost 算法实现

基于以上叙述，本章引入算法具体实现以及数学公式推导，详细介绍 XGBoost 算法的实现

目标函数正则化首先，给定一组数据集，其中包含 n 组样本，每组样本中包含 m 个特征： $D = \{(X_i, y_i)\}(|D| = n, x_i \in \mathbb{R}^m, y_i \in \mathbb{R})$ ，一个 XGBoost 模型使用 k 个 CART 数的计算结果组合预测结果：

$$\hat{y}_i = \phi(x_i) = \sum_{i=1}^k f_k(x_i), f_k \in \mathcal{F} \quad (8.21)$$

其中 $\mathcal{F} = \{f(x) = w_{q(x)}\}(q : \mathbb{R}^m \rightarrow T, w \in \mathbb{R}^T)$ ， \mathcal{F} 是所有 CART 树所在的集合， q 代表将一个样本映射到相应的叶索引。 T 是每棵树上叶子结点的总数，每个 f_k 函数对应着一个独立的树结构 q 以及叶权重 w 。在决策树中，每个树叶代表一个离散的值，而在回归树中，每个树叶包含连续的值，我们使用 w_i 来代表第 i 个树叶。Xgboost 的思想就是将特征值 x 向量映射到每一棵 CART 树的叶节点，分别计算出结果后相加，得到最终的预测值。

为了对 f_k 函数进行学习，我们对规范化的目标函数进行优化：

$$Obj^{(t)} = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

where $\Omega(f) = \gamma T + \frac{1}{2} \lambda \|w\|^2$ (8.22)

l 函数衡量着预测值与真实值之间的差距， l 可以为多种凸函数，需要指明的是，正是 l 的多样性，决定了 XGBoost 算法的可拓展性。 λ 代表着模型的复杂度。规范式 $\frac{1}{2} \lambda \|w\|^2$ 的作用是缩小权值防止过拟合。规范式越小则复杂度越低，模型的泛化能力越强。

梯度树强化与其说 XGBoost 是在训练参数，倒不是如说是在训练函数 f_t 。由于公式8.22中存在函数代替参数的情况，因此不可以用譬如计算欧几里得距离等传统方式来优化，我们的模型采用残差拟合的方式来训练。设 $\hat{y}_i^{(t)}$ 为第 t 次迭代后对第 i 组样本的预测值，本算法在上一次模型迭代基础上加上 f_t 来拟合真实值：

$$Obj^{(t)} = \sum_{i=1}^n l(y_i, \hat{y}_i^{(t-1)} + f_t(x_i)) + \Omega(f_t)$$

为了尽快拟合预测值与真实值之间的残差，我们使用泰勒二项展开式的方式分解损失函数 l ，函数展开方法式为 xgboost 算法与 GBDT 的一大区分点，同时也意味着使用 xgboost 算法要求 l 函数二阶可导，展开后公式如下：

$$Obj^{(t)} \simeq \sum_{i=1}^n [l(y_i, \hat{y}_i^{(t-1)}) + g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t)$$

上式中，我们假设 $g_i = \partial_{\hat{y}_i^{(t-1)}} l(y_i, \hat{y}_i^{(t-1)})$ 、 $h_i = \partial_{\hat{y}_i^{(t-1)}}^2 l(y_i, \hat{y}_i^{(t-1)})$ 分别为损失函数的一阶偏导数、二阶偏导数，再将上式中的常数项去掉（常数项不影响损失函数的下降过程），我们得到以下函数：

$$\tilde{Obj}^{(t)} = \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \quad (8.23)$$

假设叶子 j 上的样本集为 $I_j = \{i | q(x_i)\}$ ，将规范化项 Ω 进行拓展，我们可以把公式8.23重写为：

$$\begin{aligned} Obj^{(t)} &\simeq \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^n [g_i f_t(x_i) + \frac{1}{2} h_i f_t^2(x_i)] + \gamma T + \lambda \frac{1}{2} \sum_{j=1}^T w_j^2 \\ &= \sum_{j=1}^T [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \end{aligned} \quad (8.24)$$

假设 $G_j = \sum_{i \in I_j} g_i$, $H_j = \sum_{i \in I_j} h_i$, 可以把上述式子化简为：

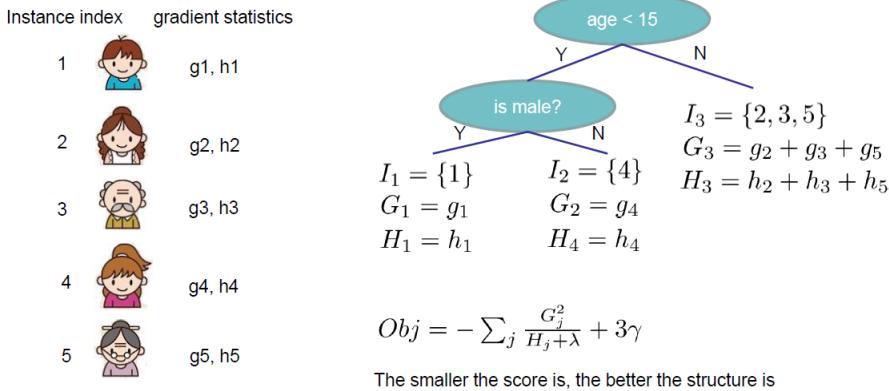
$$\begin{aligned} Obj^{(t)} &= \sum_{j=1}^T [(\sum_{i \in I_j} g_i) w_j + \frac{1}{2} (\sum_{i \in I_j} h_i + \lambda) w_j^2] + \gamma T \\ &= \sum_{j=1}^T [G_j w_j + \frac{1}{2} (H_j + \lambda) w_j^2] + \gamma T \end{aligned} \quad (8.25)$$

通过对 w_j 求导使等式为零求出 w_j 的值，再将 w_j 带回 $\mathcal{L}^{(t)}$ 我们可以得到公式：

$$w_j^* = -\frac{G_j}{H_j + \lambda} \quad Obj^{(t)} = -\frac{1}{2} \sum_{j=1}^T \frac{G_j^2}{H_j + \lambda} + \gamma T \quad (8.26)$$

公式 8.26 可以作为衡量树 q 分类质量的函数，该等式的计算结果越低，证明树结构越好。该公式有点类似判断决策树的不纯净度的公式，都是用于树分裂的衡量标准，不同点在于公式8.26可以用于更多的目标函数。详细实例，请见图8.8。

分裂树算法我们可以采用枚举法对可能的树结构进行打分判定，由于树的结

图 8.8: Obj 函数计算过程

构可能是无穷的，我们采用贪心算法，从树深度 0 开始，每一节点都遍历所有的特征，比如年龄、性别等等，然后对于某个特征，先按照该特征里的值进行排序，然后线性扫描该特征进而确定最好的分割点，最后对所有特征进行分割后，我们选择增益 Gain 最高的那个特征。假设 I_l 和 I_r 是分割过后左右节点的集合，也就是 $I = I_l \cup I_r$ 那么分割后的损失函数可以定义为公式：

$$Gain = \frac{1}{2} \left[\frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{(G_L + G_R)^2}{H_L + H_R + \lambda} \right] - \gamma \quad (8.27)$$

对于已经排好序的样本，从左到右依次扫描，同时由公式 8.27 进行计算，找到 Gain 最大的分裂点，这样足以得到最好的分裂树结构。值得注意的是，树的分裂不是必须的，当分裂带来的增益小于一定的阈值时，我们选择放弃分裂，在公式 8.27 中， γ 代表一个阈值，如果叶子分裂带来的效益小于 γ ，我们便选择不分裂。

贪心枚举树分裂问题是 XGBoost 的一个核心问题. 由于树的结构可能达到无限种，将其全部遍历是不可能的，因此，我们提出贪心枚举算法. 在贪心算法中，一次分裂枚举针对所有特征的所有可能分裂方式计算，需要注意的是，我们会对每一个特征里的值进行排序，然后线性扫描该特征从而找到最好的分割点，对所有特征进行分割以后，我们选择增益最高的那个特征。

举一个例子：小明宿舍五个人 C++ 成绩分别为 $\mathbf{d}_1 = [79, 78, 95, 88, 83]$ ，高数成绩分别为 $\mathbf{d}_2 = [69, 89, 88, 100, 73]$ 首先根据贪心枚举算法，对以上值进行排序后得到 $\mathbf{d}_1^* = [78, 79, 83, 88, 95], \mathbf{d}_2^* = [69, 73, 88, 89, 100]$ ，首先针对 c++ 课程将特征划分为左特征表 $\mathbf{d}_{1l}^* = [78]$ 右特征表 $\mathbf{d}_{1r}^* = [79, 83, 88, 95]$ ，使用衡量划分增益的 gain 函数对划分前后的增益 G_1 进行计算并记录，下一步将特征划分为左特征表 $\mathbf{d}_{1l}^* = [78, 79]$ 右特征表 $\mathbf{d}_{1r}^* = [83, 88, 95]$ ，同样计算增益 G_2 并记录，之后以此类推直到右特征表只剩下一个值，最后在增益值表 $\mathbf{G} = [G_1, G_2, G_3, G_4]$ 选出最大的增益值 G_1^* 作为 c++ 课程的最优划分增益。对于其余特征，比如高数计算增益的步骤同上，在计算完所有特征的最佳增益 G_n^* 后，在总特征增益值表

$\mathbf{G} = [G_1^*, G_2^*, \dots, G_n^*]$ 挑选最高的增益值 G_k^* , 该增益值对应的特征及划分点就是本次划分所选择的特征以及该特征的划分点。

8.3.3.1 时间复杂度分析

对于 XGBoost 算法, 生成 K 棵树, 所有树中, 最大深度为 d , $\|x\|_0$ 表示训练集中的非缺失记录, 对于贪心枚举算法来说, 总的时间复杂度为 $O(Kd\|x\|_0 \log n)$

8.3.4 XGBoost 代码实现

详见 link

8.3.5 advanced topic

稀疏划分算法

现实生活中, 在 XGBoost 的实际运用过程里, 输入数据很有可能时稀疏的。稀疏数据有多种可能性, 1) 数据值缺失。2) 统计中频繁出现的零条目。3) 特征工程中使用的如 one-hot 向量之类的数据。实际运用过程中, 算法必须拥有一整套处理稀疏数据的方案。XGBoost 的设计方案如图8.9, 对于缺失的数据, 在决策树进行判定时, 对应的样本将转入一个默认的方向。

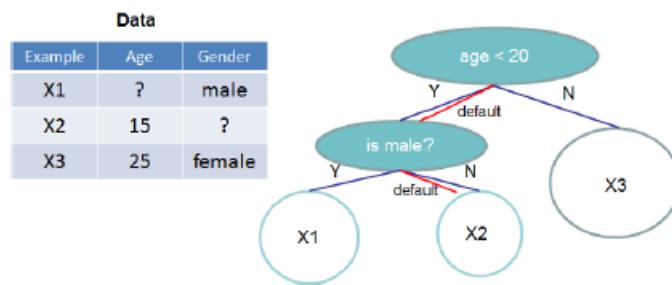


图 8.9: 稀疏数据的处理方法, 带有默认方向的树结构

并行计算经过以上的讲解, 我们可以看出, 其实在 XGBoost 模型训练过程中, 最耗费时间的便是对特征值进行排序。为减少排序时间, 我们将数据存在内存单元中, 我们将该存储单元称之为“块”。每个块中的数据是以列压缩形式存储的, 每一列由其对应的特征值进行排序。这样, 输入的数据在模型训练前只需要被训练一次, 以后便可以被重复使用。于是我们引入并行计算, 对于每一列, 其中存储着相同特征的所有特征值, 针对不同的列, 我们便可以引入多线程进行并行运算排序, 并且在划分时并行选择不同特征的最佳划分点, 并且对最佳划分

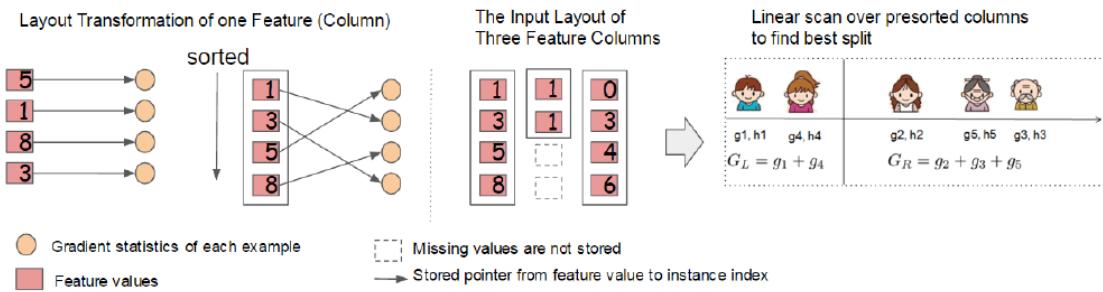


图 8.10: 相同的特征的值放在同一的列，对每列进行并行计算

点进行比较。通过以上并行计算的方法，XGBoost 算法的效率可以大大提高。具体过程如图8.10所示

Cache-aware Access 虽然先前的块结构优化了分裂算法的计算复杂度，但是，新算法要求我们提取每个特征的所有值。这些值的存储空间可能是不连续的，如果读者拥有计算机组成原理这门课程的知识就会知道，不连续空间的数据读取是很低效的，因为要反复跳跃访址。所以在分裂算法中，这种不连续数据读取会显著降低分裂算法的效率。

为解决以上问题，我们提出了 cache-aware Access 算法。具体来说，为每一个线程分配一个内部缓冲器，将数据提取到其中，然后在小批量处理过程中进行数据的整合。实验结果显示，当数据集很庞大时，本算法可以将速度提高一倍左右。

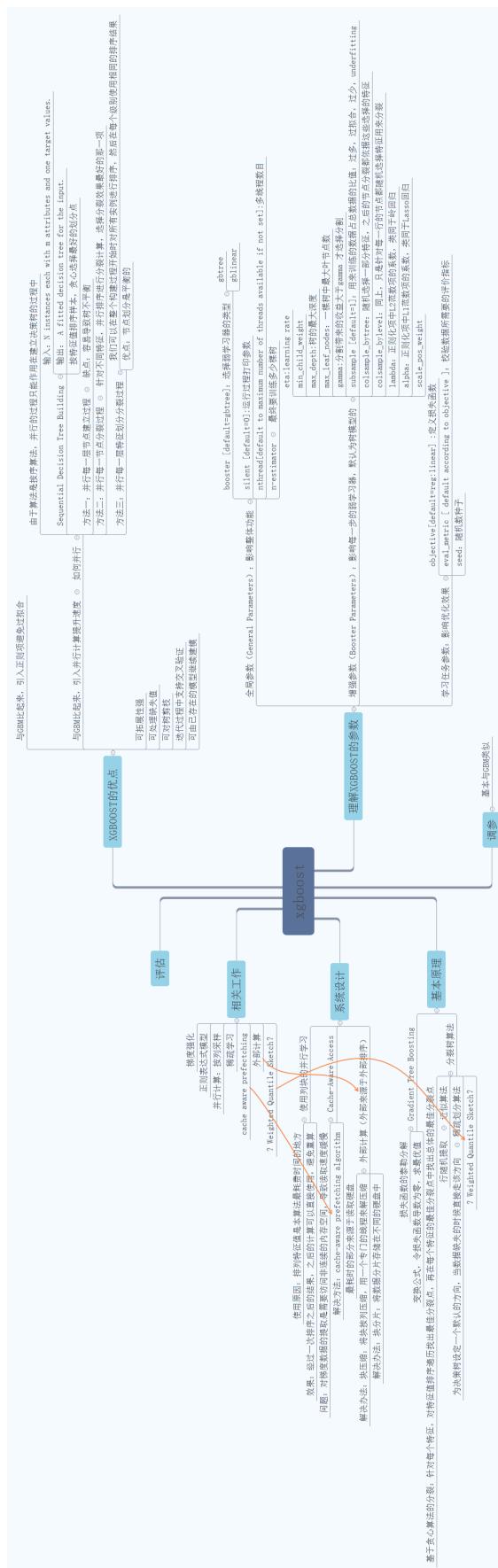


图 8.11: xgboost 思维导图

第9章 深度神经网络

深度学习允许有多层结构的计算模型学习多维的数据表象。近年来，深度学习已经广泛地应用在语音识别，图像识别，物体检测以及其他领域，并且取得了巨大的成功。本章我们将讲述深度神经网络。

在我们进入该章节之前，有必要再来看看之前讨论的线性模型。假设一个模型的输出 y 和输入 x_i 满足以下关系，那么这个模型就是一个线性模型。我们第 x 章所讲的线性回归、逻辑回归都属于这个范畴。

$$y = \sum_i w_i x_i + b$$

其中 $w_i, b \in R$ 为模型的参数。被称之为线性模型是因为当模型的输入只有一个的时候， x 和 y 形成了二维坐标上的一条直线。类似的，当模型有 n 个输入时， x 和 y 形成了 $n+1$ 维空间中的一个超平面。一个线性模型中通过输入得到输出的函数被称之为一个线性变换。因为线性模型的特点是任意线性模型的组合仍然是线性模型，所以线性模型在实际应用中表达能力极其有限，比如它就不能解决异或问题。

维基百科上对深度学习的精确定义为“一类通过多层非线性变换对复杂性数据建模的算法集合”。因为深度神经网络是实现“多层非线性变换”最常用的一种方式，所以在实际中基本上可以认为深度学习是深度神经网络的代名词。从上面的定义可以看出，深度学习两个非常重要的特性——多层和非线性。

这两个特性是非常重要的，在对复杂问题建模时是缺一不可的。

如果没有非线性函数，任意层的全连接神经网络和单层神经网络模型的表达能力没有任何区别（注意，单层神经网络模型也就是我们所说的线性模型），而且他们都是线性模型。而如果只有单层网络结构和非线性函数（没有隐藏层），依然无法解决异或问题，比如我们接下来要讨论的单层感知机。

在本章中我们将在 4.1 节中系统地介绍神经网络的结构，4.2 节中介绍如何实现神经网络的两个重大特性及其意义。4.3 节中我们以一种直观的数学视角来理解神经网络为什么有效。4.4 我们将对神经网络的优化问题进行一点讨论。

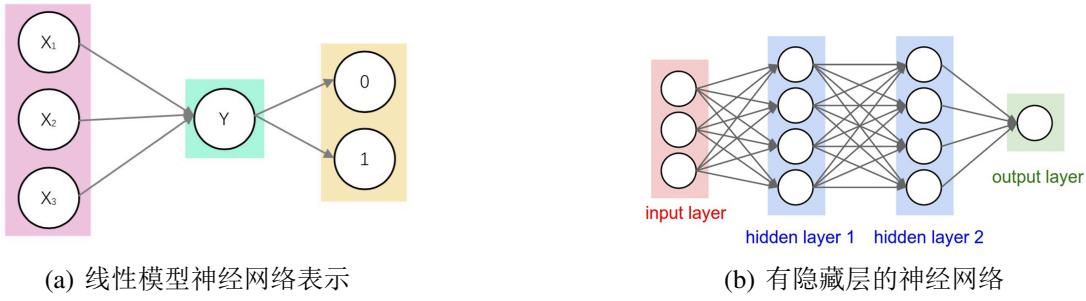


图 9.1

9.1 神经网络的结构

9.1.1 神经元模型

神经网络中最基本的成分是神经元 (neuron) 模型, 该结构受生物学启发。在生物神经网络中, 每个神经元与其他神经元相连, 当它“兴奋”时, 就会向相连的神经元发送化学物质, 从而改变这些神经元内的电位; 如果某神经元的电位超过了一个“阈值”(threshold), 那么它就会被激活, 即“兴奋”起来, 向其他神经元发送化学物质。

1943 年, [McCulloch and Pitts] 将上述描述的情形抽象为如图 9.2 所示的简单模型, 这就是 MP 神经元模型。在这个模型中, 神经元接收到来自 n 个其他模型传递过来的输入信号, 这些输入信号通过带权重的连接进行传递 (connection) 进行传递, 神经元接收到的总输入值将与神经元的阈值进行比较, 然后通过“激活函数 (activation function) *¹”处理以产生神经元的输出。

一个神经元可以用数学语言作如下描述: 我们用 d 个输入 x_1, x_2, \dots, x_d , 即向量 $\mathbf{x} = \{x_1; x_2; \dots; x_d\}$ 来表示神经元的输入, 并用静输入 $z \in R$ 表示一个神经元所获得的输入信号 x 的加权和²,

$$\begin{aligned} z &= \sum_i^d w_i x_i + b \\ &= \mathbf{w} \cdot \mathbf{x} + b \end{aligned} \tag{9.1}$$

其中 $w = \{w_1; w_2; \dots; w_d\} \in R^d$ 是 d 维的权重向量, $b \in R$ 是偏置. 净输入 z 在经过一个非线性函数 $f()$ 后, 得到神经元的激活值 (Activation) a ,

$$a = f(z), \tag{9.2}$$

其中 $f()$ 是非线性函数 (Activation Function), 它将原本是由特征 x 线性变换后的

¹ 激活函数是实现非线性化的重要手段.

² 净输入可以看成是输入特征的线性混合.

净输入 z 转化为激活值 a , 实现模型的非线性。

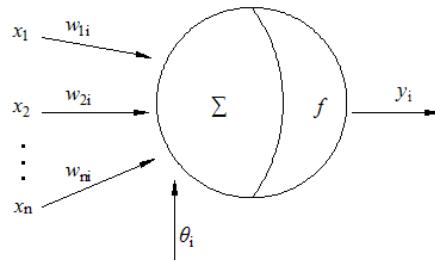


图 9.2: M-P 神经元模型

9.1.2 感知机与多层网络

感知机 (Perceptron) 是由两层神经元组成的网络结构, 如图 () 所示。输入层接受外界输入信号后传递给输出层, 输出层是 M-P 神经元。即,

$$y = f(\sum_i w_i x_i - \theta) \quad (9.3)$$

其中 x 为输入特征, w 为连接的权重, θ 为偏置, $f()$ 是跃迁函数。感知机能够实现逻辑与、或、非运算。

然而在神经网络的发展上, 一个很重要的问题是异或问题。感知机只在输出层神经元进行函数激活处理, 即只拥有一层功能神经元 (function neuron), 其学习能力非常有限。然而, 加入隐藏层 (hidden layer) 后, 神经元有更强的表达能力。(缺实验对比图)。

我们上述讨论的神经元结构, 都是层级结构, 每层神经元与下一层神经元全互连, 神经元之间不存在同层连接, 也不存在跨层连接。这样的神经网络结构通常称为前馈神经网络结构 (feedforwad neural networks)。事实上, 神经网络结构并不局限于这种形式。比如神经元具有记忆功能的反馈神经网络, 以及拥有更灵活的机构的图网络。

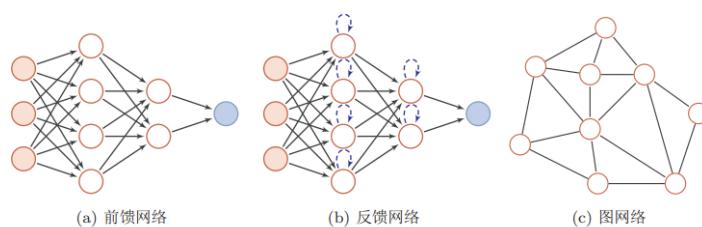


图 9.3: 三种不同类型的网络结构

9.2 神经网络两大特性

9.2.1 激活函数实现去线性化

在开篇介绍中，我们已经了解了激活函数的“神奇”作用。这里星系介绍激活函数是如何起作用的。在 4.1 中介绍神经元结构的输出为所有输入的加权和，这导致整个神经网络是一个线性模型。如果将每一个神经元（也就是神经网络中的节点）的输出通过一个非线性函数，那么整个神经网络的模型也就不再是线性的了。

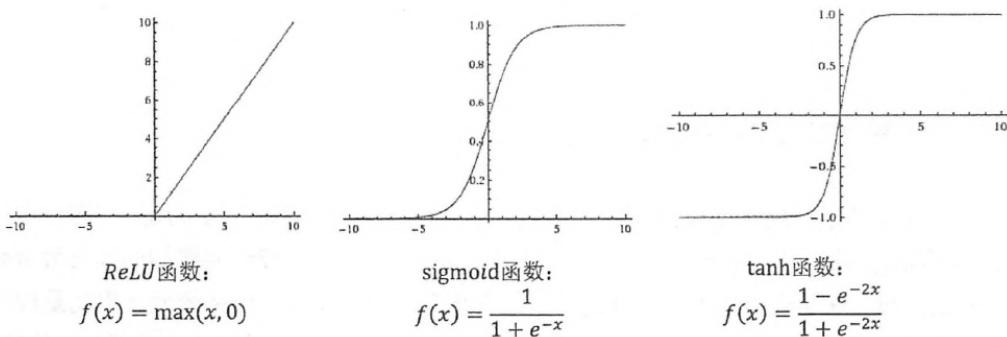


图 9.4: 几种常用的激活函数

9.2.2 多层解决异或问题

深度学习的另一个重要的特性就是多层变换。在神经网络的发展史上，一个重要的问题就是异或问题。神经网络的理论模型由 Warren McCulloch 和 Walter Pitts 在 1943 年首次提出，并在 1958 年由 Frank Rosenblatt 提出感知机模型（perceptron）模型，从而在数学上完成对神经网络的精确建模。感知机可以简单的理解为单层的神经网络，图 9.2 就是感知机的网络结构。这个结构是一个没有隐藏层的神经网络。在上个世纪 60 年代，神经网络作为对人类大脑的模拟算法收到了很多关注，直到 1969 年，Marvin Minsky 和 Seymour Papert 提出感知机是无法解决异或运算的，这里的数学求证过程暂时略去，有兴趣的同学可以动手实验。

9.2.3 从数学的视角上理解神经网络的功能

神经网络是由一层一层构建的，那么每究竟在做什么？首先一层神经网络的数学式子表达如下：

$$\vec{y} = a(W \cdot \vec{x} + b) \quad (9.4)$$

其中 \vec{x} 是输入向量， \vec{y} 是输出向量， b 是偏移向量， W 是权重矩阵， $a()$ 是激活函数。每一层仅仅是把输入 \vec{x} 经过简单的操作得到 \vec{y} 。然而从数学的视角上来看，

这个操作实际上是输入空间到输出空间的变换(矩阵的行空间到列空间)。这种变换通过以下五种操作：

1. 升维/降维 2. 放大/缩小 3. 旋转 4. 平移 5. “弯曲”这5种操作中，1,2,3的操作由 $W \cdot \vec{x}$ 完成，4的操作是由 $+\vec{b}$ 完成，5的操作则是由 $a(\cdot)$ 来实现。在这个网站 Demo³ 显示了空间变换的过程。

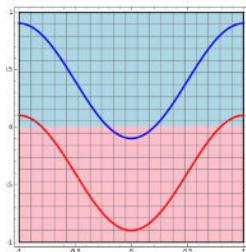


图 9.6：原始空间，无法找到一个分割面

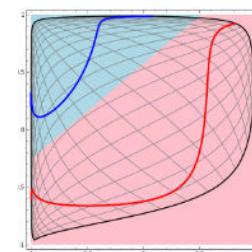


图 9.7：经过空间变换的空间，可以找到一个分割面

具体而言，这种变化对我们进行分类任务有什么作用呢？举一个一维上的例子。假如现在要给我们班里的同学进行性别分类，而我们只有座位号信息。这里我们假设座位号能一定程度上反映性别信息。由于软件学院男女比例比较固定，为 9:1。所以在开学初，大家的学号是按这种比例来排序的 1, 2, …, 9 为男生，10 号为女生，… 依次类推。所以问题就转化成我们如何在一维的空间上用一条直线来分出这两个类别。我们是无法直接找到这样的一个分割面，然而如果把空间换到另一个空间这个问题就变得容易解决了，将 $z = x \% 10$ 则 $z = 0$ 的为女生， $z > 0$ 的为男生。

总而言之，从线性可分视角：神经网络的学习就是学习如何利用矩阵的线性变换加激活函数的非线性变换，将原始输入空间投向线性可分/稀疏的空间去分类/回归。增加节点数：增加维度，也就是增加线性转换能力。增加层数：增加激活函数的次数，即增加非线性转换次数。

9.3 神经网络的优化

正如前面章节所讲述的模型一样，深层卷积神网络也是一种模型， $y = f(x \parallel \theta)$ ，其中输入记为 x ，输出记为 y ，而我们要学习的参数是 θ 。我们通常需要定义一个损失函数，来表示学习错误的损失，从而通过数据来驱动参数的学习。然而，多层网络的学习能力比单层感知机强得多，因此想要训练多层网络，需要更强大的学习算法。由于多层以及非线性激活函数的存在，要求解出 θ (即所有的 W, b) 的闭式解是极其困难的甚至不可能的。而在神经网络当中，最杰出的代表就是反向传播算法，这本节中我们将介绍方向传播算法。

³<https://cs.stanford.edu/people/karpathy/convnetjs//demo/classify2d.html>

9.3.1 反向传播算法

反向传播算法主要由两个阶段组成：激励传播与权重更新。

第一阶段：激活传播每次迭代中的传播环节包含两步：

1. (前向传播阶段) 将训练输入送入网络以获得激活响应。
2. (反向传播阶段) 将激励相应同训练输入对应的目标输出求差，从而获得输出层和隐藏层的响应误差。

第二个阶段：权重更新对于每个突触上的权重，按照以下步骤进行更新：

1. 将输入激励和响应误差相乘，从而获得权重的梯度；
2. 在这个梯度上一个比例后取反加到权重上。

这个比例（百分比）将会影响到训练过程的速度和效果，通常我们称之为“学习率”。梯度的方向知名了误差扩大的方向，因此在权重更新的时候需要对其取反从而减小权重引起的误差。通常情况下，我们反复循环迭代第一和第二个阶段，直到网络对输入的响应达到满意的预定的目标范围为止。

第 10 章 卷积神经网络

最近，小明心中产生了困惑。他住的宿舍楼栋内出现了很多花色、大小都差不多的小猫，小明总是分不清哪只是哪只。他产生了一个大胆的想法。最近刚好学习了神经网络，小明被它强大的学习能力折服了，既然神经网络这么神通广大，那么它是不是可以用来解决我们以前用传统机器学习的方法难以准确处理的数据呢？比如现在非常热门的计算机视觉任务，我们希望计算机在经过训练后可以代替人类来识别图片，但是传统的神经网络真的可以胜任吗？能不能使用神经网络根据小猫的面部照片正确的判断出谁是谁呢？果然，他找到了一种针对图像识别的新算法：卷积神经网络。

深度神经网络是实现“多层非线性变换”最常用的一种方式，其两个重要特性分别是“多层”和“非线性”，通过多层的矩阵线性变换以及激活函数的非线性变换，将原始高复杂度、高纬度的数据空间转换成线性可分的数据空间，并作出正确的分类决策。那么卷积神经网络呢？

小明首先在网络上先大致搜索了解了一些。原来，科学家最早提出卷积神经网络，就是受猫的视觉皮层电生理研究，也就是生物学上感受野机制的启发。所谓感受野（Receptive Field），指的是听觉、视觉等神经系统中一些神经元的特性，它们只接受自己所支配的刺激区域的信号。

在中学的美术课上我们就有学过，一幅画的基本组成是点线面，它们之间的任意组合形成了各种各样的图形特征如曲线，折线，三角形等。人在观察一幅图像时，往往是不太关注太过细节的局部线条特征，只从整体上观察即可得到较好的识别正确率，但是对于计算机，它并没有这种概览全局并总结推断的能力。因此，我们需要赋予它一些利于观察细节、图像特征的工具，让计算机可以通过数学的方式发现某些图像特有的特征及其组合，从而对图像进行识别。在图像识别中，这种工具叫做特征滤波器，当把这种强大的工具与神经网络结合时，其产物就是神奇的卷积神经网络，而卷积指的就是特征滤波器。

卷积神经网络是一种具有局部连接、权重共享等特性的深层前馈神经网络，最早用来处理图像信息。之所以要“卷积”，是因为如果使用全连接前馈神经网络，会存在有参数太多和局部不变性特征的问题，导致图像处理效果差强人意。简单来说，就是神经元数量过多，导致要计算的参数急剧增加，不但导致神经网络的训练效率非常低，而且还容易出现过拟合，再加上有些局部不变的一些图像特征用全连接前馈网络很难提取，因此卷积神经网络就应运而生。与传统神经网络相比，卷积神经网络能够极大的减少要计算的参数量。假设一张小猫照片的像素为 1000×1000 ，如果用传统神经网络来处理，那么输入层的神经元个数即为

$1000 \times 1000 = 10^6$ 个，再假设隐含层有 1000 个神经元，则两层之间所需要计算的参数数量为 $10^6 \times 1000 = 10^9$ 个，是一个十分可怕的数量，而且还仅仅只是两层，可想而知当神经网络不断加深时参数数量会有多么庞大；而如果使用卷积神经网络的话，由于它具有局部连接，参数共享的特征，因此我们不需要考虑隐含层的大小，训练的参数数量即前面提到的特征滤波器（也叫卷积核，后面会有具体介绍）的大小，也许只有 $100 \times 100 = 10^4$ 个，可见卷积神经网络对于处理图片信息十分高效。

10.1 “猫” 脸识别

为了实现辨别小猫图片的功能，需要在神经网络中引入卷积层和池化层等多个新的概念。卷积层用来进行图像处理中的特征提取操作，池化则类似于降维。引入全连接层、Dropout 和 Flatten。全连接层就是在神经网络中经典的神经网络全连接。Dropout 是用来在训练时按照一定的概率随机丢弃一些神经元，以获得更高的训练速度以及防止过拟合。Flatten 用于之前提到的卷积层与全连接层之间，把卷积层输出的多维数据拍扁成一维数据送进全连接层。然后使用随机梯度下降优化算法使损失函数最小化。CNN 则是将这三种特殊的层夹杂在传统的前馈型神经网络模型中，使其可以处理特别的图像模型。本章后面会有更具体的介绍。

10.2 卷积

为了学习掌握卷积神经网络的具体原理与细节，首先需要了解卷积。

卷积（Convolution），是分析数学中一种重要的运算。在信号处理中，经常使用一维卷积；在图像处理中，经常使用二维卷积。

10.2.1 一维卷积

如果一个信号发生器每 t 秒产生一个信号 x_t ，其信息的衰减速率为 w_k ，即在 $k - 1$ 个时长后，信息为原来的 w_k 倍。假设 $w_1 = 1$, $w_2 = 1/2$, $w_3 = 1/4$ ，那么在时刻 t 收到的信号 y_t 为当前时刻产生的信息和以前时刻延迟信息的叠加，

$$\begin{aligned} y_t &= 1 \times x_t + 1/2 \times x_{t-1} + 1/4 \times x_{t-2} \\ &= w_1 \times x_t + w_2 \times x_{t-1} + w_3 \times x_{t-2} \\ &= \sum_{k=1}^3 w_k \cdot x_{t-k+1} \end{aligned} \tag{10.1}$$

其中的 w_1, w_2, \dots 称为滤波器 (Filter) 或者卷积核 (Convolution Kernel)。假设滤波器长 m , 它和一个信号序列 x_1, x_2, \dots 的卷积为

$$y_t = \sum_{k=1}^m w_k \cdot x_{t-k+1} \quad (10.2)$$

信号序列 \mathbf{x} 和滤波器 \mathbf{w} 的卷积定义为

$$\mathbf{y} = \mathbf{w} \otimes \mathbf{x} \quad (10.3)$$

其中 \otimes 表示卷积运算。

10.2.2 二维卷积

给定一个图像 $X \in R^{M \times N}$, 其每个元素均为一个实数表示该点的像素值, 以及滤波器 $W \in R^{m \times n}$, 一般 $m \ll M, n \ll N$, 其卷积为

$$y_{ij} = \sum_{u=1}^m \sum_{v=1}^n w_{uv} \cdot x_{i-u+1, j-v+1} \quad (10.4)$$

图10.1是二维卷积的示例。

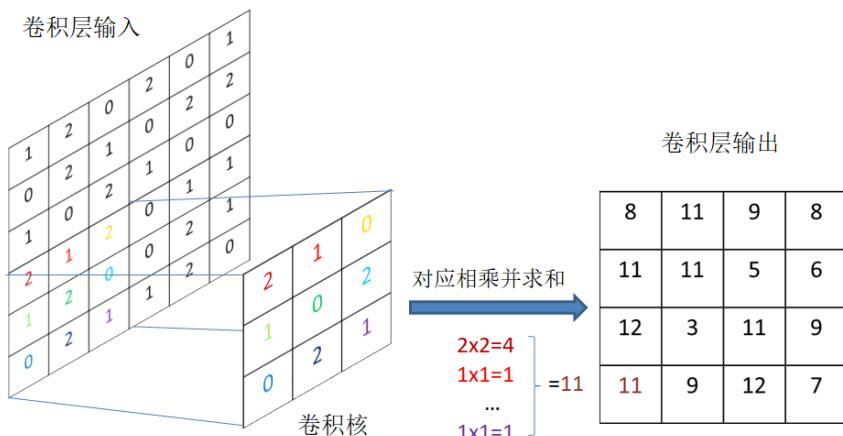


图 10.1: 二维卷积

如图中所示, 左图中右下角的黄色部分的 3×3 矩阵与用于提取特征的矩阵 $\begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{bmatrix}$ 对应位置相乘求和, $1 \times 1 + 1 \times 0 + 1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 0 \times 0 + 0 \times 1 = 4$, 最终得到输出矩阵中右下角的元素 4。其他位置按照相同的方式计算。

下面具体说明滤波器是如何移动的。简单概括来说就是: 从左往右, 从上往下。将滤波器左上角与原始矩阵左上角对齐, 被滤波器覆盖的原始矩阵部分与滤

波器元素对应相乘，然后求和得到的结果就是目标矩阵的第一个元素值。接下来就按照规定的步长开始向右移动滤波器，移动过后重复之前的计算方式得出目标矩阵的其他元素值。向右不能继续移动时，就计算完了目标矩阵的第一行，接下来则按照步长向下移动滤波器，之后再次将滤波器从左往右移动，计算目标矩阵的第二行，不断重复这些操作直到滤波器到达原始矩阵的右下角，则卷积操作完成。如图10.2所示。

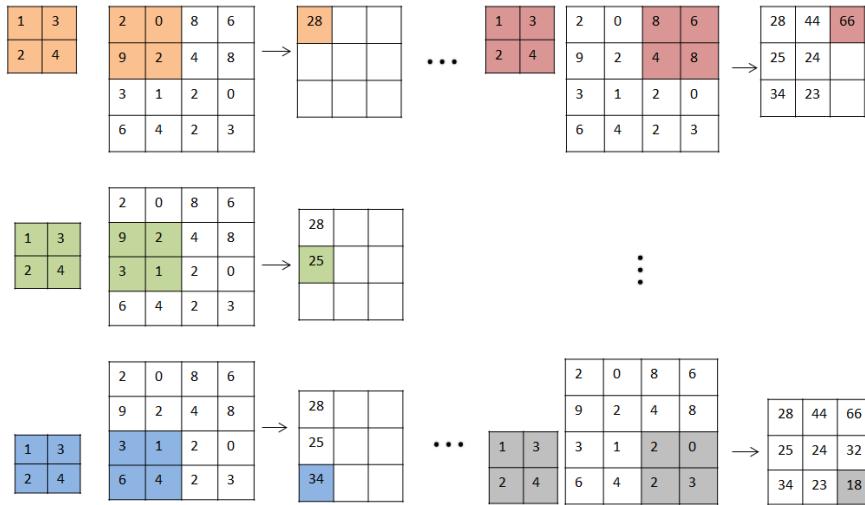


图 10.2: 卷积过程示例

10.2.3 卷积的变种

小明查询后发现，为了更灵活的抽取特征，还可以引入滤波器的滑动步长和零填充来增加卷积的多样性。

步长（Stride）是指滤波器在滑动时的时间间隔。由于一幅图像中同一个物体所处的位置并不会影响它实际上的类别，其包含的图像特征也不会因此改变，故在不同位置上使用同一个滤波器来检测特征是可行方便的方法，这一种性质也称作平移不变性。图10.3给出了步长为 2 的卷积示例。

零填充（Zero Padding）是在输入向量两端进行补零。图10.4给出了两端补零的卷积示例。

假设卷积层输入神经元 n 个，卷积大小 m ，步长 s ，两端各补零 p 个，那么该卷积层的神经元数量为 $(n - m + 2p)/s + 1$ ，为什么是这个公式呢？小明思考后明白了，原来其中 $n - m + 2p$ 表示卷积从输入神经元一端移动到另一端所需要的神经元个数，除以步长即为卷积所需走的步数，而所走的每一步都要站在两个神经元上，就好比“一刀两断”，因此，步数加一即为最终输出神经元的个数。

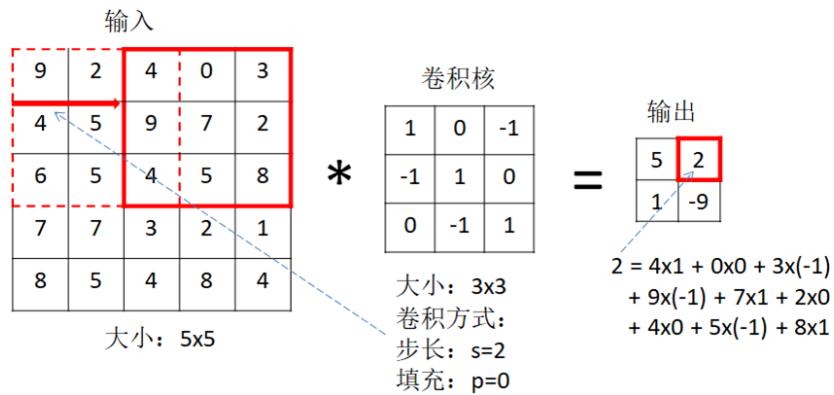


图 10.3: 步长为 2

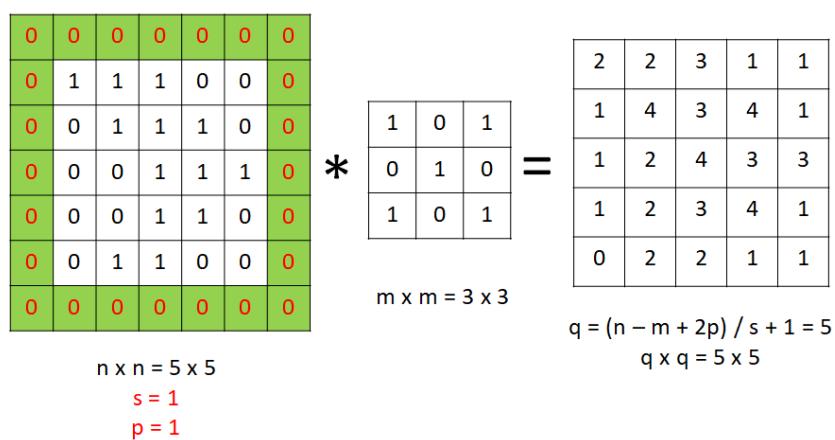


图 10.4: 零填充

上述公式为一般情况，具体而言，卷积有以下三类，这里给出输出计算公式，不再介绍具体细节：

- 窄卷积 (Narrow Convolution): 步长为 1, 不补零, 卷积后输出长度为 $n-m+1$ 。
- 宽卷积 (Wide Convolution): 步长为 1, 两端补零 $p = m - 1$, 卷积后输出长度 $n + m - 1$ 。
- 等宽卷积 (Equal-Width Convolution): 步长为 1, 两端补零 $p = (m - 1)/2$, 卷积后输出长度为 n。

在图像处理中，卷积经常作为特征提取的方法。图像在经过卷积后得到结果被称为特征映射 (Feature Map)。图10.5是特征映射示例。

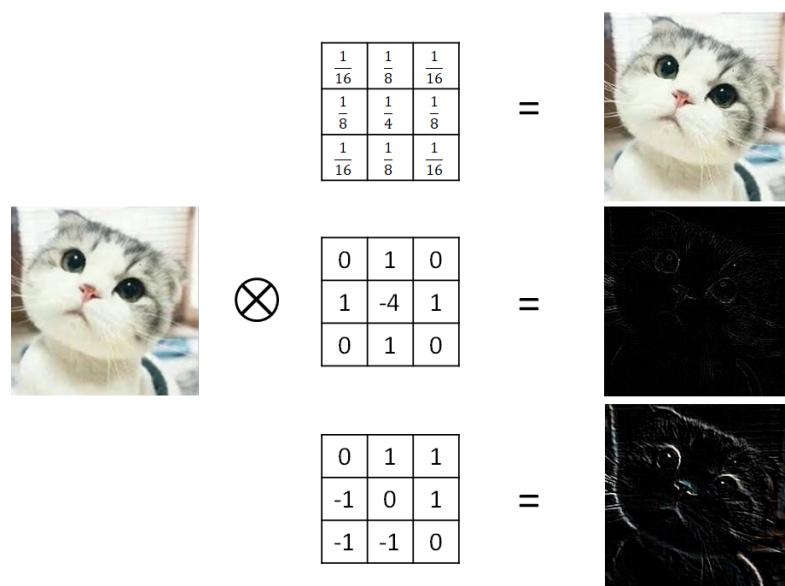


图 10.5: 特征映射

实际上，每个图象都可以表示为像素值矩阵。如图10.7表示的即为图10.6的像素值矩阵。

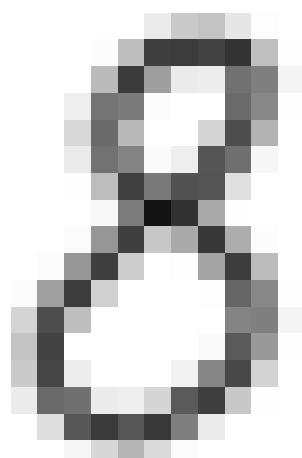


图 10.6: 手写数字 8

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	14	56	29	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	164	240	195	229	198	0	0	0	0	0	0	0	0
0	0	0	0	0	0	196	162	0	0	0	196	70	0	0	0	0	0	0	0
0	0	0	0	0	85	195	0	0	0	222	36	0	0	0	0	0	0	0	0
0	0	0	0	0	137	111	0	0	0	205	0	0	0	0	0	0	0	0	0
0	0	0	0	0	94	185	0	0	46	245	56	0	0	0	0	0	0	0	0
0	0	0	0	0	0	197	168	109	239	99	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	29	236	235	157	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	21	235	114	0	183	197	0	0	0	0	0	0	0	0	0
0	0	0	0	0	26	245	78	0	0	152	197	0	0	0	0	0	0	0	0
0	0	0	0	35	246	65	0	0	0	0	223	40	0	0	0	0	0	0	0
0	0	0	200	81	0	0	0	0	0	0	186	69	0	0	0	0	0	0	0
0	0	0	238	0	0	0	0	0	0	240	6	0	0	0	0	0	0	0	0
0	0	0	224	26	0	0	0	0	0	204	142	0	0	0	0	0	0	0	0
0	0	0	123	197	0	0	0	85	241	142	0	0	0	0	0	0	0	0	0
0	0	0	0	139	241	184	211	209	39	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	4	63	44	0	0	0	0	0	0	0	0	0	0	0	0

图 10.7: 数字 8 的像素值矩阵

10.2.4 图像

现实中的图像有多种图像模式。所谓图像模式就是把色彩分解为部分颜色组件，对颜色组件不同的分类就形成了不同的色彩模式，也就定义了不同的颜色范围。根据把图像分解为一个或多个颜色成分，就定义了该图像的通道数量。通道是用于指代图像的某个分量的常规术语。来自标准数码相机的图像将具有三个通道：红、绿、蓝。

10.2.4.1 灰度图

通道数量为 1 的称为单通道图，也叫灰度图。手写数字 8 的图10.6就是一张灰度图。

10.2.4.2 RGB 图

通道数量为 3 的成为三通道图，也就是通过对红 (R)、绿 (G)、蓝 (B) 三个颜色通道的变化以及他们互相之间的叠加来得到各种各样颜色的 RGB 图片。前面介绍的二维卷积的计算方法，就是在灰度图这种颜色通道为 1 的图像上使用的，那么对于 RGB 这种三通道图，又该如何进行卷积操作呢？

10.2.4.3 多通道图的卷积操作

以 RGB 图为例，假设有一个具有三个通道的 RGB 图像，大小为 $6 \times 6 \times 3$ ，其中 6×6 表示该图像每一个通道的大小，3 表示有三个通道。如图10.8所示。那么，进行卷积操作时，我们就要分别给每一个通道提供一个卷积核，为保证每个通道的卷积后的特征映射相同大小，使用的卷积核也需要大小相同，我们这里使用一个大小为 $3 \times 3 \times 3$ 的卷积核，它同样有 3 个通道，如图10.9。按照步长为 1，不进行零填充的方式进行卷积。最后，在每个通道都完成了卷积操作后，得到了三个特征映射，将其对应位置的数值相加，即得到了将三通道图进行卷积后生成的单通道图，其大小为 4×4 ，如图10.10。

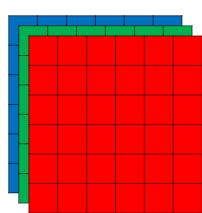


图 10.8: RGB 图通道示例

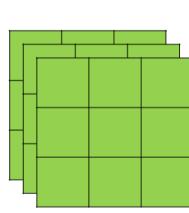


图 10.9: 三通道图的卷积核

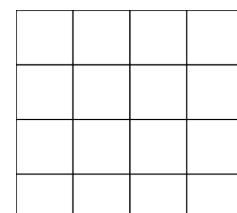


图 10.10: 卷积操作后的单通道特征映射

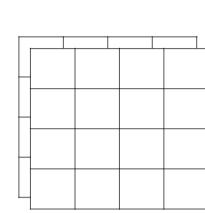


图 10.11: 卷积操作后的双通道特征映射

按照上述操作，我们就可以对多通道图进行卷积操作，生成特征映射后的单通道图。但是在现实应用中，如果直接将输入图像卷积为单通道图，可能会影响

卷积神经网络的训练性能，丢失许多图像的特征。那么我们如何才能在卷积操作后仍然得到多通道的特征映射呢？

从上面的解释中我们知道，用一个三通道的卷积核去处理一个 RGB 图，最后的特征映射为单通道，那么想要生成多通道的特征映射，我们只需要使用多个三通道的卷积核不就行了吗？每一个三通道卷积核分别用来提取图像不同的特征，比如垂直或水平的边缘，还是以上面提到的 RGB 图为例，那么我们用两个 $3 \times 3 \times 3$ 大小的卷积核，就能够得到大小为 $4 \times 4 \times 2$ 的特征映射，如图10.11所示。这样便实现了从多通道图像输入到多通道图像输出的卷积操作。

下面将上述过程一般化。假设有一图像，其大小为 $A \times A \times C$ ，我们想要通过卷积操作，将其映射为具有 X 个通道的图像。那么，我们就可以使用 X 个大小为 $B \times B \times C$ 的卷积核，按照步长为 S ，零填充为 P 进行卷积。注意：卷积核的通道数与输入图像的通道数要相同。每个通道在卷积后的大小为 $((A-B+2P)/S+1) \times ((A-B+2P)/S+1)$ （为何如此请参见“卷积的变种”小节），共有 X 个通道。理解了这一点，也就理解了卷积神经网络中最重要的卷积操作。

10.3 网络结构

卷积神经网络是神经网络的一类，已经证明在图像识别和分类等领域非常有效。卷积神经网络一般由卷积层、汇聚层和全连接层组成。

10.3.1 用卷积代替全连接

采用卷积来代替全连接，能够有效减少权重矩阵的参数。第 l 层的净输入 $z^{(l)}$ 为第 $l-1$ 层活性值 $a^{(l-1)}$ 和滤波器 $w^{(l)} \in R^m$ 的卷积，即

$$z^{(l)} = w^{(l)} \otimes a^{(l-1)} + b^{(l)} \quad (10.5)$$

其中滤波器 $w^{(l)}$ 为可学习的权重向量， $b^{(l)} \in R^{n^{(l)}}$ 为可学习的偏置。

小明从一开始就查到卷积神经网络有着局部连接和权重共享的两个重要特性，现在发现原来它们就是卷积层的性质：

10.3.1.1 局部连接

卷积层中每一个神经元都只和下一层中某个局部区域内的神经元连接（和生物学上的感受野多么的相似）。如图10.12与图10.13所示，层与层之间的连接大大减少。



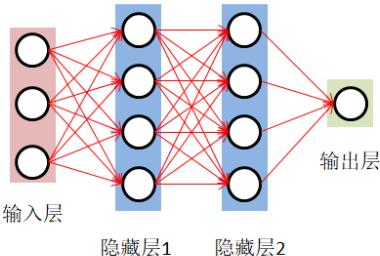


图 10.12: 全连接

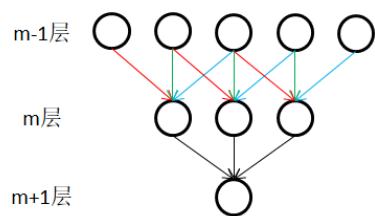


图 10.13: 局部连接

10.3.1.2 权重共享

由于卷积操作是通过移动滤波器实现，因此所经过的位置的权重是共享的。

10.3.2 卷积层

卷积层的作用是提取局部区域的特征，而卷积网络主要用于图像处理上，因此为了更充分地利用图像信息，通常将神经元组织为三维结构的神经层，大小为高度 $M \times$ 宽度 $N \times$ 深度 D ，有 D 个 $M \times N$ 大小的特征映射构成。

特征映射（Feature Map）是一幅图像在经过卷积操作后得到的特征。在输入层，特征映射就是图像本身。如果是灰度图像，就是一个特征映射，深度为 1；如果是彩色图像，分别有 RGB 三个颜色通道的特征映射，输入层深度就为 3。

为了计算输出特征映射 Y^p ，用卷积核 $W^{p,1}, W^{p,2}, \dots, W^{p,D}$ 分别对输入特征映射 X^1, X^2, \dots, X^D 进行卷积，然后再将结果相加，并加上一个标量偏置 b 得到卷积层的净输入 Z^p ，再经过非线性激活函数后就得到了输出特征映射 Y^p 。

$$Z^p = W^p \otimes X + b^p = \sum_{d=1}^D W^{p,d} \otimes X^d + b^p \quad (10.6)$$

$$Y^p = f(Z^p) \quad (10.7)$$

其中 $W^p \in R^{m \times n \times D}$ 为三维卷积核， $f(\cdot)$ 为非线性激活函数，一般用 ReLU 函数。如果希望卷积层输出 P 个特征映射，可以将上述运算重复 P 次。

10.3.3 激活函数

激活函数是作用于卷积层和全连接层之后的非线性计算操作，目的是将非线性引入神经元的输出。因为大多数现实世界的数据都是非线性的，因此我们希望神经元能够学习这些非线性表示。目前主要的激活函数有 Sigmoid, tanh, ReLU 等。

10.3.3.1 Sigmoid 函数

数学形式为 $g(z) = \frac{1}{1+e^{-z}}$ ，将输入值压缩到 [0,1] 之间。实际应用中，Sigmoid 函数在误差的反向传播中会导致梯度消失的问题，在深层的网络中会导致误差不能传播到深层，这也就导致了底层网络的权值不能随之迭代进行更新。如图10.14。

10.3.3.2 tanh(x) 函数

\tanh 数学形式为 $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ 将输入值压缩到 [-1,1] 之间，是一个饱和激活函数，其输出以 0 为中心。如图10.15。

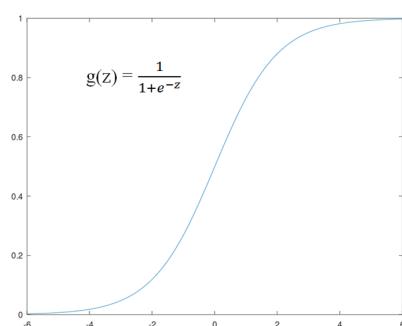


图 10.14: Sigmoid 函数

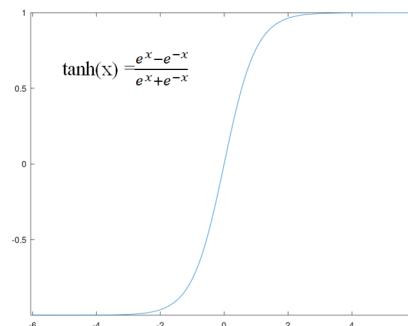


图 10.15: tanh 函数

10.3.3.3 修正型线性单元

修正型线性单元（Rectified Linear Unit, ReLU）数学表现形式为

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases}$$

与前两个激活函数相比，ReLU 是一个非饱和的激活函数，大大加速了权值梯度下降的收敛速度。如图10.16。

在输入为 $X \in R^{M \times N \times D}$ ，输出为 $Y \in R^{M' \times N' \times P}$ 的卷积层中，每一个输入特征映射都需要 D 个滤波器以及一个偏置。假设每个滤波器大小为 $m \times n$ ，那么共需要 $P \times D \times (m \times n) + P$ 个参数。

10.3.4 汇聚层

考虑到卷积神经网络是多用于处理图像的深度神经网络，小明不用上网搜索也猜到了肯定卷积层之后还有别的层，这不，他随便翻一翻便找到了关于汇聚层的一些信息。

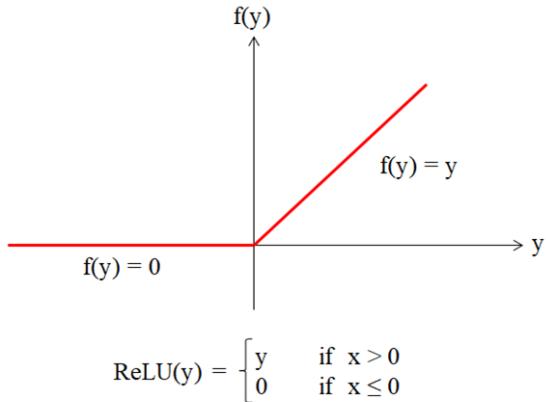


图 10.16: ReLU 函数

汇聚层 (Pooling Layer) 又名池化层，也可以叫子采样层 (Subsampling Layer)，不同于卷积层的特征提取作用，汇聚层则是注重于特征选择的功能。卷积层并不能减少特征映射后的神经元个数，但是汇聚层就能够直接降低特征维数，避免直接在卷积层后连接一个分类器可能会出现的过拟合现象。

具体来看，假设汇聚层的输入特征 $X \in R^{M \times N \times D}$ ，对于其中的每一个特征 X^d ，将其划分为了很多区域 $R_{m,n}^d$ ， $1 \leq m \leq M'$ ， $1 \leq n \leq N'$ ，这些区域可能重叠也可能不重叠。该层的名称汇聚 (Pooling) 就是指对于上述区域进行下采样 (Down Sampling) 得到一个值。

那么这个汇聚操作要怎么实现呢？也就是要是用什么函数呢？常用的汇聚函数有两种：

1. 最大汇聚 (Maximum Pooling): 顾名思义就是取区域内神经元的最大值。

$$Y_{m,n}^d = \max_{i \in R_{m,n}^d} x^i \quad (10.8)$$

上式中的 x^i 为区域 R_k^d 内每个神经元的激活值。

2. 平均汇聚 (Mean Pooling): 即取区域内所有神经元的平均值。

$$Y_{m,n}^d = \frac{1}{|R_{m,n}^d|} \sum_{i \in R_{m,n}^d} x_i \quad (10.9)$$

对每一个输入特征 X^d 的 $M' \times N'$ 个区域进行采样，得到汇聚层的输出特征 $Y^d = \{Y_{m,n}^d\}$ ， $1 \leq m \leq M'$ ， $1 \leq n \leq N'$ 。

图10.17给出了最大汇聚进行子采样操作的示例。小明很明显的看到汇聚层非常有效的减少了神经元的数量，另外也使得网络对局部形态保持不变性。

小明在搜索中还发现，在早期的一些卷积网络中，有时在汇聚层也会使用非

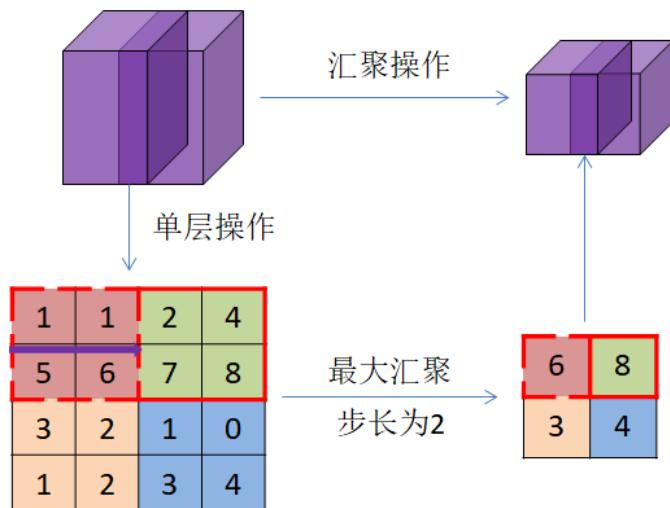


图 10.17: 最大汇聚示例

线性激活函数，比如

$$Y'^d = f(w^d \dot{Y}^d + b^d) \quad (10.10)$$

其中的 Y'^d 为汇聚层的输出， $f(\cdot)$ 为非线性激活函数， w^d 和 b^d 为可学习的变量权重和偏置。

典型的汇聚层是将每个特征映射划分为 2×2 大小的不重叠区域，然后使用最大汇聚进行下采样。汇聚层也可以被看成是一个特殊的卷积层，卷积核大小为 $m \times m$ ，步长为 $s \times s$ ，卷积核为 \max 函数或 mean 函数。过大的采样区域会减少太多的神经元，丢失过多的信息。

10.3.5 典型的卷积网络结构

现在，小明知道了一个典型的卷积神经网络是由卷积层、汇聚层、全连接层组成的。目前常用的卷积网络结构如图10.18所示。一个卷积块由多个卷积层与汇聚层（池化层），然后后面还可以再接多个全连接层。

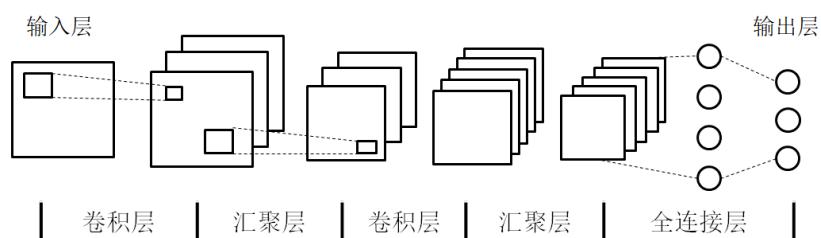


图 10.18: 卷积网络结构

小明发现，整个网络结构越来越趋向于使用更小的卷积核（比如 1×1 和

3×3) 以及更深的结构，而且由于卷积的操作越来越灵活，卷积块中汇聚层的作用变得越来越小，有变成全卷积网络的趋势。

10.4 参数学习

到现在为止，小明已经基本掌握了卷积神经网络每一层的作用以及典型结构，那么离能够使用它也就还差一步：参数学习。在卷积网络中，参数为卷积核中权重以及偏置，小明发现卷积网络也可以用误差反向传播算法进行参数学习，只需要计算卷积层中参数的梯度即可。

一般情况下，对第 1 层为卷积层，第 $l - 1$ 层的输入特征为 $X^{l-1} \in R^{M \times N \times D}$ ，通过卷积计算后可得到第 1 层的特征输入 $Z^{(l)} \in R^{M' \times N' \times P}$ 。第 1 层的第 p ($1 \leq p \leq P$) 个特征输入

$$Z^{(l,p)} = \sum_{d=1}^D W^{(l,p,d)} \otimes X^{(l-1,d)} + b^{(l,p)} \quad (10.11)$$

其中的 $W^{(l,p,d)}$ 和 $b^{(l,p)}$ 为卷积核与偏置。第 1 层共有 $P \times D$ 个卷积核和 P 个偏置，可以分别使用链式法则来计算梯度。

根据上述公式，则损失函数关于第 1 层的卷积核 $W^{(l,p,d)}$ 的偏导数为

$$\begin{aligned} \frac{\partial L(Y, \hat{Y})}{\partial W^{(l,p,d)}} &= \frac{\partial L(Y, \hat{Y})}{\partial Z^{(l,p)}} \otimes X^{(l-1,d)} \\ &= \delta^{(l,p)} \otimes X^{(l-1,d)} \end{aligned} \quad (10.12)$$

其中的 $\delta^{(l,p)} = \frac{\partial L(Y, \hat{Y})}{\partial Z^{(l,p)}}$ 为损失函数关于第 1 层的第 p 个特征输入 $Z^{(l,p)}$ 的偏导数。

同理可得，损失函数关于第 1 层的第 p 个偏置 $b^{(l,p)}$ 的偏导数为

$$\frac{\partial L(Y, \hat{Y})}{\partial b^{(l,p)}} = \sum_{i,j} [\delta^{(l,p)}]_{i,j} \quad (10.13)$$

在查阅中小明了解到，在卷积网络中，每层参数的梯度依赖于其所在层的误差项 $\delta^{(l,p)}$ 。因此下一步，他就打算去学习误差项的计算。

10.4.1 误差项的计算

深入学习后小明发现，对于卷积层和汇聚层，误差项的计算是有所不同的。

10.4.1.1 汇聚层

当第 $l+1$ 层为汇聚层时，由于是下采样， $l+1$ 层的每一个神经元的误差项 δ 实际上对应于第 1 层相应特征的一个区域。1 层的第 p 个特征区域中的每一个神

经元都与第 $l+1$ 层的第 p 个特征区域中的一个神经元相连。根据链式法则可知，第 l 层的一个特征映射的误差项 $\delta^{(l+1,p)}$ ，只需要将 $l+1$ 层对应的特征映射的误差项 $\delta^{(l+1,p)}$ 进行反向上采样，再和 l 层特征映射的激活值偏导数逐元素相乘，就可以得到 $\delta^{(l,p)}$ 。

第 l 层的第 p 个特征映射的误差项 $\delta^{(l,p)}$ 的推导过程如下：

$$\begin{aligned}\delta^{(l,p)} &\triangleq \frac{\partial L(Y, \hat{Y})}{\partial Z^{(l,p)}} \\ &= \frac{\partial X^{(l,p)}}{\partial Z^{(l,p)}} \cdot \frac{\partial Z^{(l+1,p)}}{\partial X^{(l,p)}} \cdot \frac{\partial L(Y, \hat{Y})}{\partial Z^{(l+1,p)}} \\ &= f'_l(Z^{(l,p)}) \odot \mathbf{up}(\delta^{(l+1,p)})\end{aligned}\quad (10.14)$$

其中 $f'_l(\cdot)$ 为第 l 层使用的激活函数一阶导， \mathbf{up} 为上采样函数（upsampling），与汇聚层中使用的下采样操作刚好相反。如果下采样是最大汇聚（max pooling），误差项 $\delta^{(l+1,p)}$ 中的每个值会直接传递到上一层最大值所对应的神经元，而其他神经元的误差项都设为 0。如果是平均汇聚（mean pooling），误差项中每一个值就会平均分配到上一层的所有神经元。

10.4.1.2 卷积层

当 $l+1$ 层为卷积层时，假设特征映射输入 $Z^{(l+1)} \in R^{M' \times N' \times P}$ ，其中第 p ($1 \leq p \leq P$) 个特征映射输入为

$$Z^{(l+1,p)} = \sum_{d=1}^D W^{(l+1,p,d)} \otimes X^{(l,d)} + b^{(l+1,p)} \quad (10.15)$$

其中 $W^{(l+1,p,d)}$ 和 $b^{(l+1,p)}$ 为第 $l+1$ 层的卷积核以及偏置。

第 l 层的第 d 个特征映射的误差项 $\delta^{(l,d)}$ 的推导过程如下：

$$\begin{aligned}\delta^{(l,d)} &= \Delta \frac{\partial L(Y, \hat{Y})}{\partial Z^{(l,d)}} \\ &= \frac{\partial X^{(l,d)}}{\partial Z^{(l,d)}} \cdot \frac{\partial L(Y, \hat{Y})}{\partial X^{(l,d)}} \\ &= f'_l(Z^{(l)}) \odot \sum_{p=1}^P (\mathbf{rot180}(W^{(l+1,p,d)}) \tilde{\otimes} \frac{\partial L(Y, \hat{Y})}{\partial Z^{(l+1,p)}}) \\ &= f'_l(Z^{(l)}) \odot \sum_{p=1}^P (\mathbf{rot180}(W^{(l+1,p,d)}) \tilde{\otimes} \delta^{(l+1,p)})\end{aligned}\quad (10.16)$$

其中 $\tilde{\otimes}$ 为宽卷积。

10.5 提升技巧

卷积神经网络虽然有着局部连接、权重共享的特性，但是训练过程中的参数也是非常多的，这里就提供一些能够提升卷积神经网络性能的一些技巧。

10.5.1 数据增强

如果原始图像数据集不够大，而对于深度网络又需要大量的训练数据才能够取得较好的性能，那么这时我们可以通过数据增强来提高性能，比较流行的有对图像进行水平翻转、随机修建和光照、色彩变换。后面将会提到的 AlexNet 网络中，就使用了 PCA (Principal Component Analysis, 主成分分析) 方式，利用 PCA 改变了训练图像中 RGB 通道的强度。

10.5.2 预处理

在自己获取的图像样本上，在网络的训练之前还需要做预处理操作。第一个简单的预处理操作是将数据以 0 点为中心，然后做规范化操作，这种处理的另一种形式是将每个维度归一化，使得每个维度的最大最小值分别为 1 和 -1。在不同的输入特征具有不同的单位时，应用此预处理是绝对有帮助的。

10.5.3 训练技巧

10.5.3.1 卷积核与池化窗口的大小

在训练期间，输入图像大小更倾向于 2 的倍数。另外，重要的是采用小的卷积核（例如 3×3 ）和小步长（例如 1）进行填充为 0 的卷积计算，这不仅可以减少参数数量，而且提高了整个深度网络的准确率。

10.5.3.2 学习速率

一般来说，学习速率根据批处理的大小来决定。但是当改变批处理的大小时，我们不可能总是再更改学习率。通常，在训练开始时的学习率为 0.1，在训练集上的损失值不再下降时，将学习率除以 2 或者 5，再继续训练，可能会取得不错的效果。

10.5.3.3 预训练模型的微调

由于预先训练的深度模型良好的泛化能力，我们可以直接在其他数据集上直接采用预先训练的模型。

10.5.4 正则化

正则化 (regularization) 是神经网络用来防止过拟合的方法。有以下几种形式：

- L2 正则化 是正则化最常用的形式之一，可以直接在损失函数中加上带有乘性偏置的所有权值 w 的平方和实现。
- L1 正则化 是正则化中相对常见的形式，在原始的代价函数后面加一个 L1 正则化项，即所有权重 w 的绝对值的和，乘以 λ/n 。
- 最大范围限制 正则化的另一种形式是强制每个神经元的权重的绝对上限，并使用预测的梯度下降来强制约束。
- Dropout 是一种很简单的正则化技术，dropout 策略是以一定的概率 (dropout ratio) 将隐层神经元的输入、输出设置为零。选中“dropped out”的隐层神经元在网络中既不参与前向传播，也不参与误差的反向传播，但是它的权重会保留下。所以每次输入样本进行训练，该卷积神经网络就相当于采用了与之前不同的网络结构，但是不同的结构之间共享了权重。
- 批处理正则化 (Batch Normalization) 是在卷积计算和激活函数中间进行规范化计算，逐层尺度归一，首先通过对相应的激活区域做规范化操作，使得输出信号各个维度服从均值为 0，标准差为 1 的正态分布，最后通过缩放与平移 (scale and shift) 操作使批处理规范化计算的输出结果还原为最初的输入特征，从而保证网络的容纳能力。

10.6 几种典型的卷积神经网络

学到这里，小明已经真正了解了完整的卷积神经网络过程，他想学习一些具体的例子，以便能够更加清楚的理解其中的相关概念，为自己以后的运用打好基础。下面便是他找到的一些典型的卷积神经网络。

10.6.1 LeNet-5

LeNet-5[LeCun et al., 1998] 提出的比较早，是一个非常成功的神经网络模型。90 年代美国很多的银行都使用基于 LeNet-5 的手写数字识别系统识别支票上面的手写数字。图10.19所示为 LeNet-5 的网络结构。

不计输入层，LeNet-5 共有 7 层，下面具体介绍每一层的结构：

1. 输入层：输入大小为 32×32 的手写字母图像。
2. C1 层为卷积层，使用 6 个 5×5 的滤波器，得到 6 组大小为 $28 \times 28 = 784$ 的特征映射。因此，C1 层的神经元数量为 $6 \times 784 = 4704$ ，可训练参数数量为 $6 \times 25 + 6 = 156$ ，连接数为 $156 \times 784 = 122304$ （包括偏置在内，下同）。

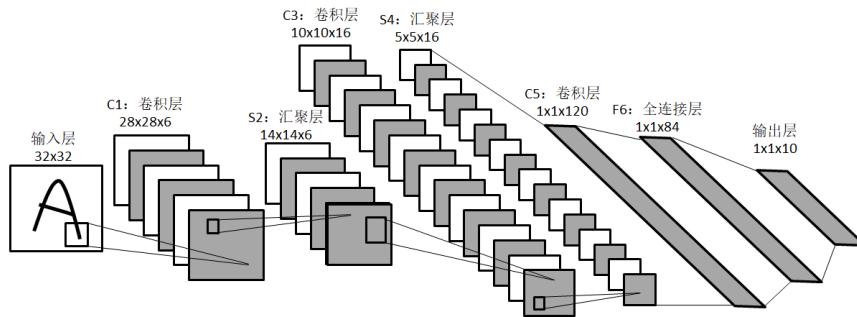


图 10.19: LeNet-5 网络结构

3. S2 层为卷积层。LeNet-5 中用一个连接表（具体介绍见 Advanced Topic）来表示输入与输出之间的依赖关系，共使用了 60 个 5×5 的滤波器，得到 16 组大小为 10×10 的特征映射。神经元数量为 $16 \times 100 = 1600$ ，可训练参数数量为 $(60 \times 25) + 16 = 1516$ ，连接数为 $100 \times 1516 = 151600$ 。
4. S4 层为汇聚层，采样窗口为 2×2 ，得到 16 个 5×5 大小的特征映射，可训练数量为 $16 \times 2 = 32$ ，连接数为 $16 \times 25 \times (4 + 1) = 2000$ 。
5. C5 层为卷积层，使用 $120 \times 16 = 1920$ 个 5×5 的滤波器，得到 120 组大小为 1×1 的特征映射。C5 层的神经元数量为 120，可训练参数数量为 $1920 \times 25 + 120 = 48120$ ，连接数为 $120 \times (16 \times 25 + 1) = 48120$ 。
6. F6 层为全连接层，有 84 个神经元，可训练参数数量为 $84 \times (120 + 1) = 10164$ 。因为是全连接，所以连接数也是 10164。
7. 输出层：由 10 个欧式径向基函数（Radial Basis Function, RBF）函数组成。

10.6.2 AlexNet

AlexNet[Krizhevsky et al., 2012] 是第一个现代深度卷积网络模型，它首次使用了很多技术方法，比如使用 GPU 进行并行训练，采用 ReLU 作为非线性激活函数，使用 Dropout 防止过拟合，使用数据增强来提高准确率等。AlexNet 的结构如图10.20所示，包括有 5 个卷积层，3 个全连接层和一个 softmax 层。因为规模超出了当时单个 GPU 的内存限制，AlexNet 将网络拆分为两半，分别放在两个 GPU 上，GPU 间只在某些层（比如第 3 层）进行通讯。

AlexNet 的具体结构如下：

1. 输入层， $224 \times 224 \times 3$ 的图像。
2. 第一个卷积层，使用两个 $11 \times 11 \times 3 \times 48$ 的卷积核，步长 $s = 4$ ，零填充 $p = 3$ ，得到两个 $55 \times 55 \times 48$ 的特征映射组。
3. 第一个汇聚层，使用大小为 3×3 的最大汇聚操作，步长 $s = 1$ ，零填充 $p = 1$ ，得到两个 $27 \times 27 \times 128$ 的特征映射组。
4. 第二个卷积层，使用两个 $5 \times 5 \times 48 \times 128$ 的卷积核，步长 $s = 1$ ，零填充

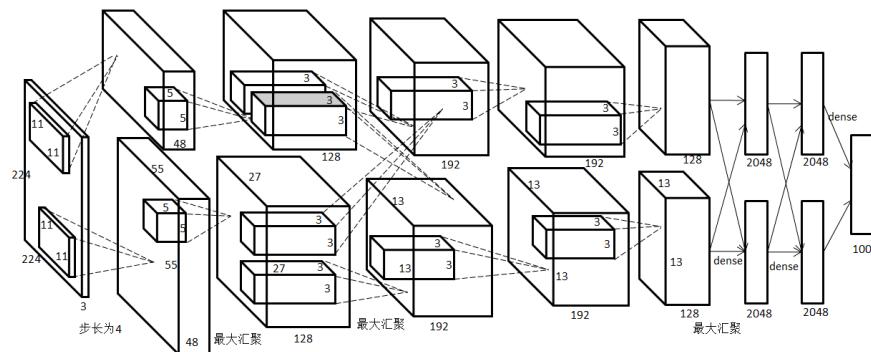


图 10.20: AlexNet 网络结构

$p = 1$, 得到两个 $27 \times 27 \times 128$ 的特征映射组。

5. 第二个汇聚层, 使用大小为 3×3 的最大汇聚操作, 步长 $s = 2$, 得到两个 $13 \times 13 \times 128$ 的特征映射组。
6. 第三个卷积层为两个路径的融合, 使用一个 $3 \times 3 \times 256 \times 384$ 的卷积核, 步长 $s = 1$, 零填充 $p = 1$, 得到两个 $13 \times 13 \times 192$ 的特征映射组。
7. 第四个卷积层, 使用两个 $3 \times 3 \times 192 \times 192$ 的卷积核, 步长 $s = 1$, 零填充 $p = 1$, 得到两个 $13 \times 13 \times 192$ 的特征映射组。
8. 第五个卷积层, 使用两个 $3 \times 3 \times 192 \times 128$ 的卷积核, 步长 $s = 1$, 零填充 $p = 1$, 得到两个 $13 \times 13 \times 128$ 的特征映射组。
9. 汇聚层, 使用大小为 3×3 的最大汇聚操作, 步长 $s = 2$, 得到两个 $6 \times 6 \times 128$ 的特征映射组。
10. 三个全连接层, 神经元数量分别为 4096, 4096, 1000。

10.6.3 Inception 网络

在卷积网络中, 如何设置卷积层的卷积核大小是一个很关键的问题。在 Inception 网络中, 一个卷积层包含有多个不同大小的卷积操作, 成为 Inception 模块。Inception 网络是由多个 inception 模块和少量的汇聚层组成的。Inception 模块同时使用 $1 \times 13 \times 35 \times 5$ 等不同大小的卷积核, 并将得到的特征映射在深度上拼接起来作为输出特征映射。

图10.21给出了v1版本的Inception模块, 采用四组平行的特征抽取方式, 分别为 $1 \times 13 \times 35 \times 5$ 的卷积和 3×3 的最大卷积。同时为了提高效率, Inception 模块在进行 $3 \times 35 \times 5$ 的卷积之前、 3×3 的最大卷积之后, 进行一次 1×1 的卷积来减少特征映射的深度。如果输入特征映射之间有冗余信息, 1×1 的卷积相当于先进行一次特征抽取。

Inception 网络有多个改进版本, 其中比较有代表性的有 Inception v3 网络 [Szegedy et al., 2016]。Inceptionv3 网络用多层的 j 小卷积核替换大的卷积核, 以减

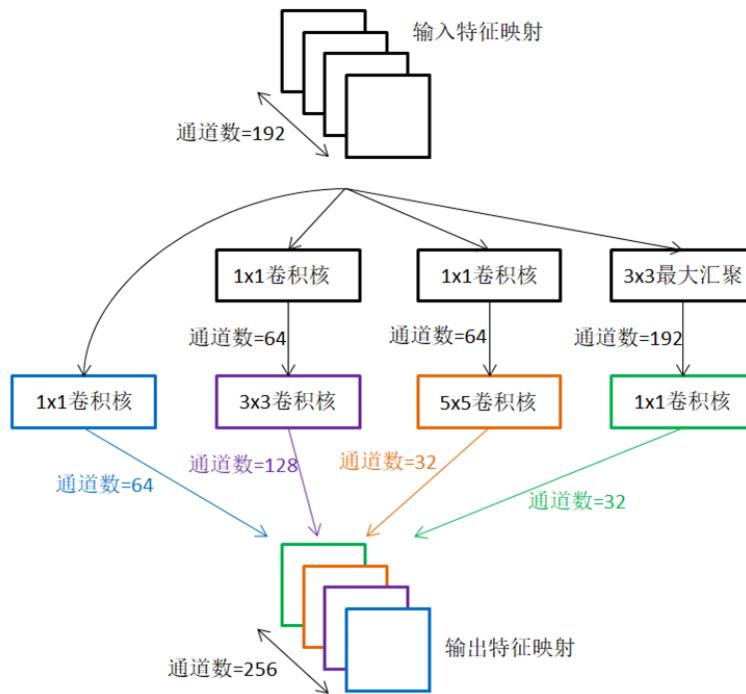


图 10.21: Inception v1 的网络结构

少计算量和参数量，并保持感受野不变。此外，Szegedy et al. [2017] 还提出了结合直连（Shortcut Connect）边的 Inception 模块：Inception-ResNet v2 网络，并在此基础上设计了一个更优化的 Inception v4 模型。

10.6.4 残差网络

残差网络（Residual Network, ResNet）是通过给非线性的卷积层增加直连边的方式来提高信息的传播效率。假设在一个深度网络中，我们希望用一个非线性单元（可以为一层或多层的卷积层） $f(x, \theta)$ 去逼近一个目标函数 $h(x)$ 。如果目标函数拆分为两部分：恒等函数（Identity Function） x 和残差函数（Residue Function） $h(x)-x$ 。

$$h(x) = \underbrace{x}_{\text{恒等函数}} + \underbrace{(h(x) - x)}_{\text{残差函数}} \quad (10.17)$$

根据通用近似定理，一个由神经网络构成的非线性单元有足够的能力来近似逼近原始目标函数或残差函数，但是集中后者更容易学习 [He et al., 2016]。因此，原来的优化问题可以转换为：让非线性单元 $f(x, \theta)$ 去近似残差函数 $h(x)-x$ ，并用 $f(x, \theta) + x$ 去逼近 $h(x)$ 。图 10.22 给出了一个典型的残差单元示例。残差单元由多个级联的（等长）卷积层和一个跨层的直连边组成，再经过 ReLU 激活后得到输出。

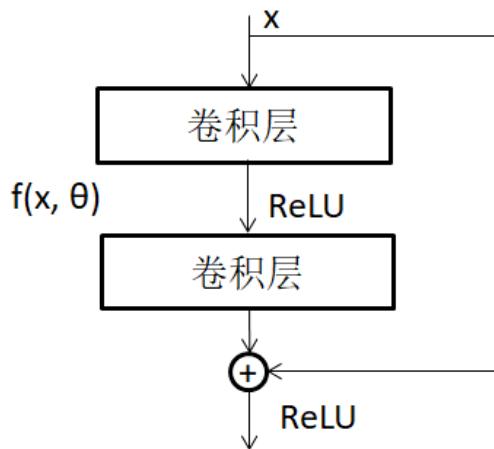


图 10.22: 一个简单的残差单元结构

10.7 训练过程注意事项

- 任务的数据集应该足够大,且训练集大小应占到原数据集的 65%,测试集则为 25%,剩余的 10% 则为验证集
- 为了增加模型的鲁棒性,可以给部分图片增加随机噪声,从而使得模型在预测部分有噪声的真实图片时性能更好.
- 网络层的权值初始化通常采用一个均值为 0, 方差较小的区间分布来初始化.
- 与机器学习任务类似,自适应的学习率能让神经网络的训练表现更佳,使其收敛加快且不易陷入局部最优解.
- 神经网络停止训练的条件与机器学习任务类似,最常用的是当损失值小于某个阈值或在多个迭代后没有进一步减小则停止训练.

10.8 其他卷积方式

除了之前查到的可以通过步长和零填充来进行不同的卷积操作外,小明还查到了其他的一些卷积方式。

10.8.1 空洞卷积

对于一个卷积层,如果希望增加输出单元的感受野,一般可以通过三种方式实现:(1)增加卷积核大小;(2)增加层数;(3)在卷积之前进行汇聚操作。前两种操作会增加参数数量,而第三种丢失一些信息。

空洞卷积(Atrous Convolution)是一种不增加参数数量,同时增加输出单元感受野的一种方法,也称为膨胀卷积(Dilated Convolution)。

空洞卷积通过给卷积核插入“空洞”来变相地增加其大小，也就是在原始卷积核元素之间插入 0。对于原始大小为 m 的卷积核，如果在卷积核的每两个元素之间插入 $d - 1$ 的空洞，那么卷积核的有效大小为

$$m' = m + (m - 1) \times (d - 1) \quad (10.18)$$

其中 d 被称为膨胀率 (Dilation Rate)。当 $d = 1$ 时卷积核为普通的卷积核。

图10.23给出了空洞卷积的示例。

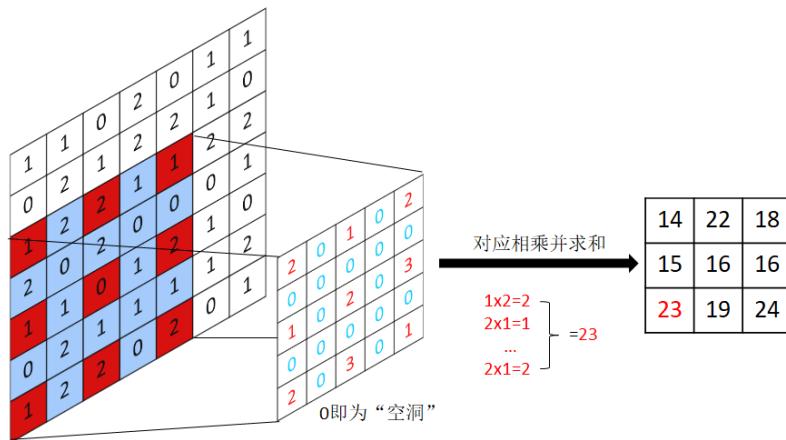


图 10.23: 空洞卷积示例，膨胀率为 2，步长为 1，没有零填充

10.8.2 转置卷积

有时候会遇到一些任务，需要将低维特征映射到高维特征，并且也希望通过卷积操作来实现。假设有一个高维向量为 $x \in R^d$ 和一个低维向量 $z \in R^p$, $p < d$ 。如果用仿射变换来实现高维到低维的映射， $z = Wx$ ，其中 $W \in R^{p \times d}$ 为转换矩阵。那么反过来不就是低维到高维的转换了吗？也就是 $x = W^T z$ 。

在全连接网络中，忽略激活函数，前向计算和反向传播就是一种转置关系。比如前向计算时，第 $l+1$ 层的净输入为 $z^{(l+1)} = W^{(l+1)}z^{(l)}$ ，反向传播时，第 l 层的误差项为 $\delta^{(l)} = (W^{(l+1)})^T \delta^{(l+1)}$ 。

卷积操作也可以写为仿射变换的形式。假设一个 5 维向量 x ，经过大小为 3 的卷积核 $w = [w_1, w_2, w_3]^T$ 进行卷积，得到 3 维向量 z 。卷积操作可以写成

$$\begin{aligned} z &= w \otimes x \\ &= \begin{bmatrix} w_1 & w_2 & w_3 & 0 & 0 \\ 0 & w_1 & w_2 & w_3 & 0 \\ 0 & 0 & w_1 & w_2 & w_3 \end{bmatrix} \cdot x \\ &= Cx \end{aligned} \quad (10.19)$$

其中 C 是一个稀疏矩阵，其非零元素来自于卷积核 w 中的元素。

如果要实现 3 维向量 z 到 5 维向量 x 的映射，可以通过仿射矩阵的转置来实现。

$$\begin{aligned}
 x &= C^T z \\
 &= \begin{bmatrix} w_1 & 0 & 0 \\ w_2 & w_1 & 0 \\ w_3 & w_2 & w_1 \\ 0 & w_3 & w_2 \\ 0 & 0 & w_3 \end{bmatrix} \cdot z \\
 &= \text{rot180}(w) \tilde{\oplus} z
 \end{aligned} \tag{10.20}$$

其中 $\text{rot180}(\cdot)$ 表示旋转 180 度。

从上两个公式可以看出，从仿射变换的角度来看两个卷积操作 $z = w \otimes x$ 和 $x = \text{rot180}(w) \tilde{\oplus} z$ 也是形式上的转置关系。因此，我们将低维特征映射到高维特征的卷积操作成为转置矩阵（Transposed Convolution），也称为反卷积（Deconvolution）。

在卷积网络中，卷积层的前向计算和反向传播也是一种转置关系。

对一个 p 维的向量 z ，和大小为 m 的卷积核，如果希望通过卷积操作来映射到高维向量，只需要对向量 z 进行两端补零 $p = m - 1$ ，然后进行卷积，可以得到 $p + m - 1$ 维的向量。

转置卷积同样适用于二维卷积。图10.24给出了步长 $s = 1$ ，零填充 $p = 1$ 的转置卷积。

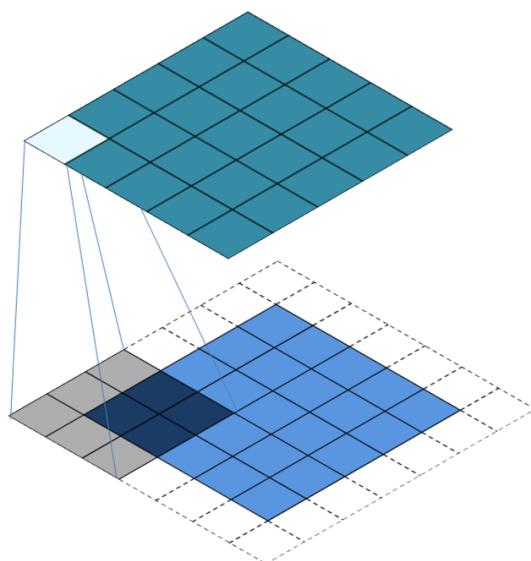


图 10.24: 转置卷积， $s = 1, p = 1$

10.8.2.1 微步卷积

通过增加卷积操作的步长 $s > 1$ 来实现对输入特征的降采样操作，大幅降低特征维数。同样的，也可以通过减少转置卷积的步长 $s < 1$ 来实现上采样操作，提高特征维数。

步长 $s < 1$ 的转置卷积也称为微步卷积（Fractionally-Strided Convolution）。为了实现微步卷积，可以在输入特征之间插入 0 来间接地使得步长变小。

如果卷积操作的步长 $s > 1$ ，希望其对应的转置卷积的步长为 $\frac{1}{s}$ ，需要在输入特征之间插入 $s - 1$ 个 0 来使得其移动的速度变慢。

以一维转置卷积为例，对一个 p 维的向量 z ，和大小为 m 的卷积核，通过对向量 z 进行两端补零 $p = m - 1$ ，并且在每两个向量元素之间插入 $s - 1$ 个 0，然后进行步长为 1 的卷积，可以得到 $s \times (p - 1) + m$ 维的向量。

图10.25给出了一个步长 $s = 1$ ，零填充 $p = 2$ 的微步卷积。

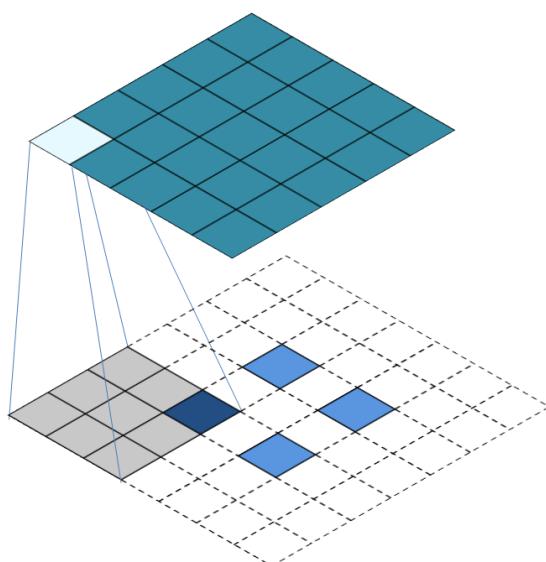


图 10.25: 转置卷积， $s = 1, p = 2$

10.9 Advanced Topic

10.9.1 互相关

原来，在机器学习和图像处理领域，卷积的主要功能是在图像的像素矩阵上滑动一个卷积核，通过该方式得到新的特征。在具体操作中，一般会使用互相关操作来代替卷积。互相关（Cross-Correlation）是一种衡量两个序列相关性的函数，用滑动窗口的点积计算来实现。给定图像 $X \in R^{M \times N}$ 和卷积核 $W \in R^{m \times n}$ ，则互相

关为

$$y_{ij} = \sum_{u=1}^m \sum_{v=1}^n w_{uv} \cdot x_{i+u-1, j+v-1} \quad (10.21)$$

互相关和卷积的区别在于卷积核是否翻转。但是翻转对于特征提取的能力没有影响。

学习了更多的知识点后，小明并没有满足，他希望能掌握更多关于卷积的性质，毕竟打好基础，才能更好的理解后面更难的内容。于是，他又开始查阅资料寻找关于卷积的更多内容。

10.9.2 卷积的数学性质

10.9.2.1 交换性

小明惊喜的发现，原来卷积也有交换性，即 $x \otimes y = y \otimes x$ 。而当输入信息和卷积核具有固定长度时，他们的宽卷积依然具有交换性。

10.9.2.2 导数

小明也具体学习了卷积函数的求导方法。假设 $Y = W \otimes X$ ，其中 $X \in R^{M \times N}$, $W \in R^{m \times n}$, $Y \in R^{(M-m+1) \times (N-n+1)}$, 函数 $f(Y) \in R$ 为一个标量函数，则

$$\begin{aligned} \frac{\partial f(Y)}{\partial w_{uv}} &= \sum_{i=1}^{M-m+1} \sum_{j=1}^{N-n+1} \frac{\partial y_{ij}}{\partial w_{uv}} \frac{\partial f(Y)}{\partial y_{ij}} \\ &= \sum_{i=1}^{M-m+1} \sum_{j=1}^{N-n+1} x_{i+u-1, j+v-1} \frac{\partial f(Y)}{\partial y_{ij}} \\ &= \sum_{i=1}^{M-m+1} \sum_{j=1}^{N-n+1} \frac{\partial f(Y)}{\partial y_{ij}} x_{i+u-1, j+v-1} \end{aligned} \quad (10.22)$$

其中 $y_{ij} = \sum_{u,v} w_{u,v} x_{i+u-1, j+v-1}$ 。

从上式可以得到， $f(Y)$ 关于 W 的偏导数为 X 和 $\frac{\partial f(Y)}{\partial Y}$ 的卷积

$$\frac{\partial f(Y)}{\partial W} = \frac{\partial f(Y)}{\partial Y} \otimes X \quad (10.23)$$

同理得到

$$\begin{aligned} \frac{\partial f(Y)}{\partial x_{st}} &= \sum_{i=1}^{M-m+1} \sum_{j=1}^{N-n+1} \frac{\partial y_{ij}}{\partial x_{st}} \frac{\partial f(Y)}{\partial y_{ij}} \\ &= \sum_{i=1}^{M-m+1} \sum_{j=1}^{N-n+1} w_{s-i+1, t-j+1} \frac{\partial f(Y)}{\partial y_{ij}} \end{aligned} \quad (10.24)$$

其中当 $(s - i + 1) < 1$, 或 $(s - i + 1) > m$, 或者 $(t - j + 1) < 1$, 或 $(t - j + 1) > n$ 时, $w_{s-i+1,t-j+1} = 0$ 。即相当于对 \mathbf{W} 进行了 $p = (M - m, N - n)$ 的零填充。

10.9.3 连接表

由于全连接层的每一个输出特征映射都依赖于所有的输入特征映射, 因此二者之间是全连接的关系。但是实际上这种全连接关系是不必要的。为此, 可以定义一个连接表 (Link Table) \mathbf{T} 来描述少量每个输出特征映射与少量输入特征映射之间的连接关系, 如表10.1。如果有依赖, 则表中对应位置为 1, 否则为 0。

$$Y^p = f\left(\sum_{\substack{d, \\ T_{p,d} = 1}} W^{p,d} \otimes X^d + b^p\right)$$

其中 \mathbf{T} 为 $P \times D$ 大小的连接表。假设表内非零个数为 K , 每个滤波器的大小为 $m \times n$, 那么所需参数为 $K \times m \times n + P$ 个。

	0	1	2	3	4	5	6	7	8	9	10	
0	X				X	X	X			X	X	
1	X	X				X	X	X			X	
2	X	X	X				X	X	X			
3		X	X	X				X	X	X		
4			X	X	X				X	X	X	
5				X	X	X				X	X	

表 10.1: LeNet-5 中 C3 层的连接表

10.10 总结

1959 年, David Hubel 和 Torsten Wiesel 发现猫的视觉皮层中存在有承担不同层次的视觉感知功能的两种细胞, 简单细胞只对感受野中特定角度 (orientation) 的光带敏感, 而复杂细胞则是对特定方向 (direction) 的光带敏感, 这一发现启发了日本学者福岛邦彦 (Kunihiko Fukushima), 他于 1979 年仿照生物的视觉皮层提出了具有深度结构的神经网络: neocognitron 模型。之后, 人工智能领域在反向传播算法 (Back-Propagation, BP) 的研究中取得了突破进展, 使得历史上第一个卷积神经网络: 1987 年由 Alexander Waibel 提出的应用于语音识别问题的时间延迟网络 (Time Delay Neural Network, TDNN) 得以使用 BP 框架进行学习。1988 年, 第一个二维卷积神经网络: 平移不变人工神经网络 (SIANN) 由 Wei Zhang 提出并应用于检测医学影像中。一年后, Yann LeCun 构建了 LeNet 的最初版本, 并对权重进行随机初始化以及使用随机梯度下降 (Stochastic Gradient

Descent, SGD) 进行学习。也是 LeCun 在论述 LeNet 的网络结构中首次使用了“卷积”一词。1998 年，在 LeNet 的基础上，Yann LeCun 构建了更加完备的卷积神经网络 LeNet-5 并在手写数字识别问题中取得成功。从此，卷积神经网络的应用得到了广泛关注，微软在 2003 年开始使用卷积神经网络开发光学字符读取 (Optical Character Recognition, OCR) 系统。其他包括人脸识别，手势识别的应用研究也得到展开。2006 年后，随着逐层学习和参数微调 (fine-tuning) 技术的出现，卷积神经网络在结构上不断加深，各类学习和优化理论不断出现。2012 年的 AlexNet 是第一个现代深度卷积网络模型，在图像分类上实现真正的突破，使用了比如 GPU、ReLU 激活函数、dropout 防止过拟合等新技术。自此之后又出现了很多优秀的卷积网络，比如 VGG, Inception v1, v2, v4 网络等。

卷积神经网络已经成为计算机视觉和自然语言处理领域的主流模型，并且在其他比如物理学、遥感科学与大气科学领域也都有所应用。在具体应用中，随着不断增加的网络层数，卷积核趋向于使用 1×1 , 3×3 大小，一些不规则的卷积操作也开始出现，比如空洞卷积。网络结构也不断增加卷积层，减少全连接层和汇聚层，构建全卷积网络 (Fully Convolutional Network, FCN)。

第 11 章 RNN & LSTM

11.0.0.1 本章基本概念(专有名词)

超参数 在开始学习过程之前设置值的参数，定义关于模型的更高层次的概念，如复杂性或学习能力。在神经网络中又叫网络参数，网络超参数。

隐藏层 网络中除输入层和输出层以外的其他各层，不直接接受外界的信号，也不直接向外界发送信号。

隐状态 隐藏层节点的状态，随时间变化而变化，又称隐藏层神经元的活性值，可以理解为一种神经元中存储的动态信息。

非线性激活函数 logistic 函数， \tanh 函数。

11.1 RNN

11.1.1 为什么需要循环神经网络

首先我们从一个简单问题讲起，你能看懂下面这句话吗？“神学络喜我。欢经网”，显然，即使你看得懂每一个字，但是你无法理解整句话的意思。但是如果写成下面这样呢，“我喜欢学神经网络。”，这下整个句子都通顺了，意思也很好理解。是的，在现实生活中，把某个整体割裂成众多部分，每一小部分单独理解，意义是浅薄的。而当这些小部分乱序地拼凑，然后当作整体来理解时，意义很可能是混乱的。而这种序列又广泛存在于现实生活中，比如音频、视频、文字信息。这些都说明了序列信息的重要性与广泛性。而读者通过前面章节的阅读，可能会有这样的疑问，能够几乎模拟“任意”连续函数的神经网络，又是否有处理序列信息的能力呢？

传统的神经网络的确有处理序列的能力，但很有限，比如只能处理特定长度的输入输出，而且无法记忆上一时刻的输入输出，更别说处理上一时刻的输入输出了。即使人为地用上一时刻的输入输出对这一时间的输入简单地进行处理，也无法达到很好的效果。而卷积神经网络的确有着对区域处理的能力，但是卷积神经网络认为各部分的区域之间是独立的，而序列信息则认为序列中各部分区域是有顺序联系的，显然卷积神经网络也不太适合处理序列信息。

那么现在就需要一个天然能处理序列的模型了。你以前可能听过自回归模型、延时神经网络等能够天然处理序列信息的模型，但这些都不是本章的重点，本章重点是循环神经网络（Recurrent Neutral Network, RNN），一个天然能处理序列的神经网络。

为什么循环神经网络能够很好地处理序列信息呢？类比我们人脑，当我们用大脑进行学习的时候，除非是婴儿，不然都不可能是大脑一片空白地进行学习。我们很多时候会忘记一些发生在我们身上不太重要的事情，但更多时候我们会带着过往的经历去学习新的内容。循环神经网络正是借鉴了这种思想，让神经网络带有可以自反馈的神经元，这种神经元可以利用上一时刻的自带信息与这一时刻的输入信息进行更新，这理论上可以处理任意长度的序列信息。但是该神经元在不断更新的过程中，必然会逐渐丢失掉过去太久的信息。故循环神经网络是一种具有短期记忆的神经网络。在下一节中，我们会简单介绍循环神经网络的结构和不同类型的循环神经网络。

11.1.2 不同类型的循环神经网络

在了解不同类型的循环神经网络之前，我们首先需要知道循环神经网络的一般结构是什么。假设存在 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ 长度为 T 的输入序列，一般来说， $\mathbf{x}_t \in \mathbb{R}^{d \times 1}$, $t \in \{1, 2, \dots, T\}$ 。那么在 t 时刻隐藏层状态为 \mathbf{h}_t ，而 t 时刻的隐藏层状态不止和这时刻的输入 \mathbf{x}_t 有关，还和上时刻的隐藏层状态 \mathbf{h}_{t-1} 有关，所以他们之间的关系如下。

$$\mathbf{z}_t = \mathbf{W}\mathbf{x}_t + \mathbf{U}\mathbf{h}_{t-1} + \mathbf{b} \quad (11.1)$$

$$\mathbf{h}_t = f(\mathbf{z}_t) \quad (11.2)$$

上式中的 \mathbf{z}_t 指的是未经激活的净输入， \mathbf{W} 是输入-状态权重矩阵， \mathbf{U} 是状态-状态权重矩阵， \mathbf{b} 是偏置项， f 是激活函数，如 tanh 和 ReLu 函数等。其中 $\mathbf{z}_t \in \mathbb{R}^{m \times 1}$, $\mathbf{W} \in \mathbb{R}^{m \times d}$, $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\mathbf{b} \in \mathbb{R}^{m \times 1}$, $\mathbf{h}_t \in \mathbb{R}^{m \times 1}$, m 由隐藏层的神经元个数决定。示意图如图11.1所示。

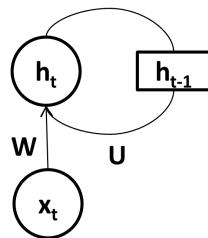


图 11.1: 循环神经网络的输入层和隐藏层

特别地，当隐藏状态 \mathbf{h} 还没有被初始化的时候，即 \mathbf{h}_1 的赋值存在如下关系

$$\mathbf{h}_0 = \mathbf{0} \quad (11.3)$$

$$\mathbf{h}_1 = f(\mathbf{W}\mathbf{x}_1 + \mathbf{U}\mathbf{h}_0 + \mathbf{b}) \quad (11.4)$$

$$= f(\mathbf{W}\mathbf{x}_1 + \mathbf{b}) \quad (11.5)$$

这里的 $\mathbf{0}$ 是维度与 \mathbf{Wx}_1 相同的零向量。

而当只有一层隐藏层的时候，在 t 时刻循环神经网络的输出为 \mathbf{o}_t ，它与隐藏层的关系如下

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h \quad (11.6)$$

上式中 \mathbf{V} 为状态-输出矩阵， \mathbf{b}_h 为偏置项。其中 $\mathbf{o}_t \in \mathbb{R}^{q \times 1}$, $\mathbf{V} \in \mathbb{R}^{q \times m}$, $\mathbf{b}_h \in \mathbb{R}^{q \times 1}$, q 的存在是因为假定训练集 \mathcal{T} 中在 t 时刻的标签值 $\mathbf{y}_t \in \mathbb{R}^{q \times 1}$ 。示意图如图11.2所示。为了示意图的简洁，该小节的示意图均省略偏置项。

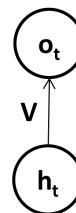


图 11.2: 循环神经网络的隐藏层和输出层

如果我们让循环神经网络凸显出循环之意，我们可以把循环神经网络按时间维度展开。我们可以清晰地看到循环神经网络中隐藏层参数是共享的，示意图如图11.3所示。

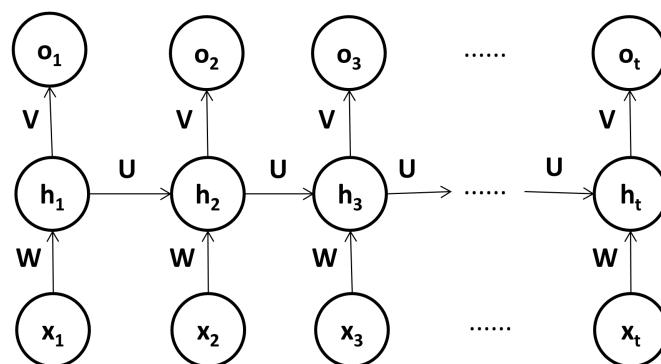


图 11.3: 按时间维度展开的循环神经网络

对于机器学习的不同任务，输入输出很可能结构不一样。那么根据这些任务特点不同，我们应用不同类型的的循环神经网络。下面我们来看下这几种不同类型的循环神经网络。

11.1.2.1 多对一循环神经网络

多对一的循环神经网络模型主要用于解决机器学习中的分类问题，如文本的分类、视频的分类。输入是序列信息，输出是类别。

例如，训练集 \mathcal{T} 的一个样本为 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$ ，标签值为 $y \in \{1, 2, \dots, C\}$ ，那么该样本 \mathbf{x} 按序列顺序输入到多对一的循环神经网络之后，可以得到 $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$ 一系列的隐藏状态。我们可以将 \mathbf{h}_T 最后用于输出，即

$$\mathbf{o} = \mathbf{V}\mathbf{h}_T + \mathbf{b}_h \quad (11.7)$$

上式的 \mathbf{o} 是整个多对一循环神经网络的输出。具体示意图如图11.4所示。

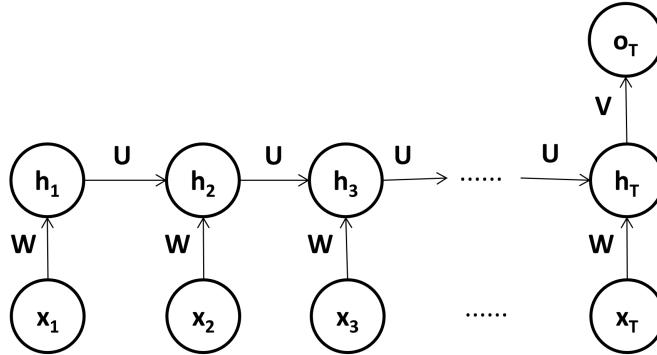


图 11.4: 多对一循环神经网络

当然，在一般情况下，会对 y 进行 one-hot 向量化， \mathbf{h}_T 最后连一层前馈神经网络，即利用 softmax 等函数对 \mathbf{o} 进行激活，最后分类。在本小节中就不详细介绍介绍了。

11.1.2.2 一对多循环神经网络

一对多的循环神经网络的典型应用是图片的描述，输入一张图片，输出文本序列。又或者是给定一个情感类别，生成一个音符序列。

训练集 \mathcal{T} 的一个样本为 \mathbf{x} ，对应标签值则是一个序列 $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T)$ ，那么样本 \mathbf{x} 输入到一对多循环神经网络之后，得到 $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$ 一系列的隐藏状态，但是每个隐藏状态都会进行输出，即

$$\mathbf{h}_1 = f(\mathbf{Wx} + \mathbf{b}) \quad (11.8)$$

$$\mathbf{h}_t = f(\mathbf{Uh}_{t-1} + \mathbf{b}), t \in \{2, \dots, T\} \quad (11.9)$$

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h, t \in \{1, \dots, T\} \quad (11.10)$$

具体示意图如图11.5所示。

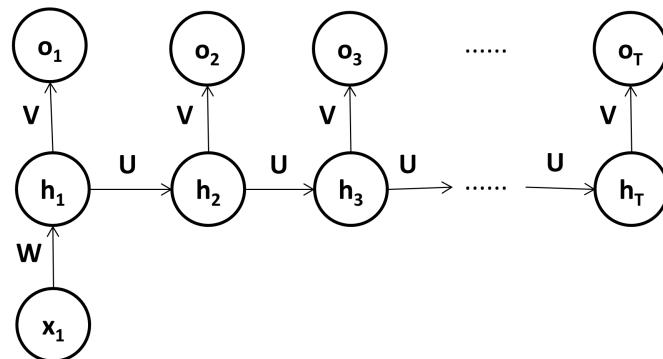


图 11.5: 一对多循环神经网络

11.1.2.3 同步的多对多循环神经网络

同步的多对多循环神经网络主要是用在序列标注，比如一个英语单词可能有多种词性，但是在句子的具体位置就只有一种词性了，对每一个单词进行词汇标注。或者是对一个视频的每帧进行分类，又或者用于一个字对下一个字的预测。

训练集的一个样本 $\mathbf{x} = (x_1, x_2, \dots, x_T)$ ，对应标签序列为 $\mathbf{y} = (y_1, y_2, \dots, y_T)$ ，样本按序列顺序输入到多对一的循环神经网络之后，可以得到 h_1, h_2, \dots, h_T 一系列的隐藏状态。每个隐藏状态都会进行输出，即

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h \quad (11.11)$$

上式中的 $t \in \{1, \dots, T\}$ ，具体示意图如图11.6所示。

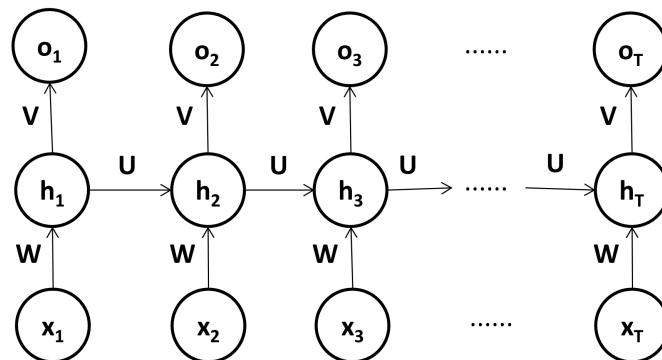


图 11.6: 同步的多对多循环神经网络

11.1.2.4 异步的多对多循环神经网络

异步的多对多循环神经网络又叫编码-解码模型 (Encoder-Decoder model) , 主要用于机器翻译。因为在翻译的过程中, 一种语言翻译成另一种语言的时候, 字符序列的数量大多数情况下都是不一致的。异步的多对多循环神经网络并不要求输入序列和输出序列有一一对应关系, 也不需要保持相同的长度。

训练集的一个样本 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{T_1})$, 标签序列为 $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_{T_2})$ 。那么样本 \mathbf{x} 按序列顺序输入到一个循环神经网络中, 最后得到隐藏状态 \mathbf{h}_{T_1} , 传进另一个循环神经网络中得到输出序列 $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_{T_2}$ 。具体关系如下

$$\mathbf{h}_t = f_1(\mathbf{W}\mathbf{x}_t + \mathbf{U}_1\mathbf{h}_{t-1} + \mathbf{b}_1), t \in \{1, 2, \dots, T_1\} \quad (11.12)$$

$$\mathbf{h}_{T_1+t} = f_2(\mathbf{U}_2\mathbf{h}_{T_1+t-1} + \mathbf{b}_2), t \in \{1, 2, \dots, T_2\} \quad (11.13)$$

$$\mathbf{o}_t = \mathbf{V}\mathbf{h}_{T_1+t} + \mathbf{b}_h, t \in \{1, 2, \dots, T_2\} \quad (11.14)$$

上式中的 f_1 和 f_2 分别代表两个循环神经网络的激活函数。 $\mathbf{U}_1, \mathbf{U}_2, \mathbf{b}_1, \mathbf{b}_2$ 分别为两个循环神经网络的对应参数。具体示意图如图11.7所示。

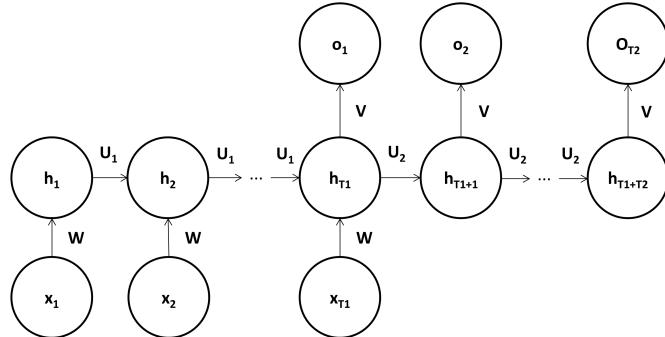


图 11.7: 异步的多对多循环神经网络

11.1.3 参数更新

循环神经网络的参数的更新, 一般需要梯度的计算, 可以通过随机梯度下降算法进行更新。

为了不失一般性, 我们利用同步的多对多循环神经网络辅以随机梯度下降算法, 进行参数更新的计算。那么给定训练集 \mathcal{L} 的一个样本 (\mathbf{x}, \mathbf{y}) , $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T)$, 输入序列长度为 T , $\mathbf{y} = (\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_T)$, 输出序列的长度也为 T 。若没有特殊说明, 本节出现的变量维度表示如下, $\mathbf{x}_t \in \mathbb{R}^{d \times 1}$, $\mathbf{y}_t \in \mathbb{R}^{q \times 1}$, $\mathbf{z}_t \in \mathbb{R}^{m \times 1}$, $\mathbf{W} \in \mathbb{R}^{m \times d}$, $\mathbf{U} \in \mathbb{R}^{m \times m}$, $\mathbf{b} \in \mathbb{R}^{m \times 1}$, $\mathbf{h}_t \in \mathbb{R}^{m \times 1}$, $\mathbf{o}_t \in \mathbb{R}^{q \times 1}$, $\mathbf{V} \in \mathbb{R}^{q \times m}$, $\mathbf{b}_h \in \mathbb{R}^{q \times 1}$ 。

我们定义 t 时刻的损失函数为

$$\mathcal{L}_t = \mathcal{L}(\mathbf{y}_t, g(\mathbf{o}_t)) \quad (11.15)$$

上式中的 g 一般为分类函数， \mathcal{L} 为损失函数，如交叉熵损失函数。整个序列的损失函数为

$$\mathcal{L} = \sum_{t=1}^T \mathcal{L}_t \quad (11.16)$$

因 $\mathbf{o}_t = \mathbf{V}\mathbf{h}_t + \mathbf{b}_h$ ，所以 \mathcal{L}_t 对参数 \mathbf{V} 第 i 行第 j 列元素 V_{ij} 的梯度为

$$\begin{aligned} \frac{\partial \mathcal{L}_t}{\partial \mathbf{V}_{ij}} &= \left(\frac{\partial (\mathbf{V}\mathbf{h}_t + \mathbf{b}_h)}{\partial \mathbf{V}_{ij}} \right)^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \\ &= \mathbb{I}_i(\mathbf{h}_t^j)^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \\ &= \left(\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \right)^i \mathbf{h}_t^j \end{aligned} \quad (11.17)$$

上式中的 $\mathbb{I}_i(\mathbf{h}_t^j)$ 是一个列向量，其第 i 行是列向量 \mathbf{h}_t 的第 j 个分量，其余行皆为 0。而 $\left(\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \right)^i$ 是列向量 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$ 的第 i 个分量，我们在前馈神经网络用过类似的处理方式。

那么 \mathcal{L}_t 对参数 \mathbf{V} 梯度为

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{V}} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \mathbf{h}_t^\top \quad (11.18)$$

$\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$ 由具体的损失函数和分类函数 g 所决定，这里暂不讨论。

所以整个序列的损失函数 \mathcal{L} 对参数 \mathbf{V} 梯度为

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{V}} &= \sum_{t=1}^T \frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \mathbf{h}_t^\top \\ &= \sum_{t=1}^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \mathbf{h}_t^\top \end{aligned} \quad (11.19)$$

而整个序列的损失函数 \mathcal{L} 对参数 \mathbf{b}_h 梯度为

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}_h} = \sum_{t=1}^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \quad (11.20)$$

整个序列的损失函数 \mathcal{L} 对参数 \mathbf{U} 、 \mathbf{W} 、 \mathbf{b} 的梯度却并不是那么好求的，下面我们介绍随时间反向传播算法来解决这个问题。

11.1.3.1 随时间反向传播算法

随时间反向传播（Backpropagation Through Time, BPTT）算法类似反向传播算法，但是，BPTT 算法把循环神经网络看成一个展开的多层神经网络，循环神经网络每个时刻的隐藏层相当于展开的多层神经网络的中的一层，但是层与层之间的参数是共享的。所以当进行反向传播计算时，共享参数的梯度是所有展开层对应参数梯度之和。

首先我们先计算 \mathcal{L} 对 \mathbf{U} 的梯度 $\frac{\partial \mathcal{L}}{\partial \mathbf{U}}$ 。

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial \mathbf{U}} &= \sum_{t=1}^T \frac{\partial \mathcal{L}}{\partial \mathcal{L}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{U}} \\ &= \sum_{t=1}^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{U}}\end{aligned}\quad (11.21)$$

而直接求 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}}$ 的话，难以计算，我们先求 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}_{ij}}$ 。在时刻 $k (1 \leq k \leq t)$ 中，存在下列关系

$$\mathbf{z}_k = \mathbf{Wx}_k + \mathbf{Uh}_{k-1} + b \quad (11.22)$$

$$\mathbf{h}_k = f(\mathbf{z}_k) \quad (11.23)$$

故 \mathcal{L}_t 对 \mathbf{U}_{ij} 的梯度是

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}_{ij}} &= \sum_{k=1}^t \left(\frac{\partial^* \mathbf{z}_k}{\partial \mathbf{U}_{ij}} \right)^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k} \\ &= \sum_{k=1}^t \mathbb{I}_i(\mathbf{h}_{k-1}^j)^T \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k}\end{aligned}\quad (11.24)$$

上式中的 $\frac{\partial^* \mathbf{z}_k}{\partial \mathbf{U}_{ij}}$ 是较为特殊的偏导数，即 $\mathbf{z}_k = \mathbf{Wx}_k + \mathbf{Uh}_{k-1} + b$ 中 \mathbf{h}_{k-1} 暂视为与 \mathbf{U}_{ij} 无关的变量，对 \mathbf{U}_{ij} 进行求偏导数。而 $\mathbb{I}_i(\mathbf{h}_{k-1}^j)$ 是一个列向量，其第 i 行是列向量 \mathbf{h}_{k-1} 的第 j 个分量，其余行皆为 0，我们在上面的章节有过相似的操作。

设 $\delta_{t,k} = \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k}$ ，借鉴反向传播算法的思路，可列

$$\begin{aligned}\delta_{t,k} &= \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_k} \\ &= \frac{\partial \mathbf{h}_k}{\partial \mathbf{z}_k} \frac{\partial \mathbf{z}_{k+1}}{\partial \mathbf{h}_k} \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_{k+1}} \\ &= \text{diag}(f'(\mathbf{z}_k)) \mathbf{U}^\top \delta_{t,k+1}\end{aligned}\quad (11.25)$$

上式 $\delta_{t,k} \in \mathbb{R}^{m \times 1}$, $f'(\mathbf{z}_k) \in \mathbb{R}^{m \times 1}$, $\text{diag}(f'(\mathbf{z}_k)) \in \mathbb{R}^{m \times m}$, 其中 $\text{diag}(f'(\mathbf{z}_k))$ 表示由向量 $f'(\mathbf{z}_k)$ 中的元素组成对角矩阵的矩阵, 非对角位置皆为 0。

特别地, 因为 $\delta_{t,t+1}$ 不存在, $\delta_{t,t}$ 的计算公式不能沿用上式, 故 $\delta_{t,t}$ 为

$$\begin{aligned}\delta_{t,t} &= \frac{\partial \mathcal{L}_t}{\partial \mathbf{z}_t} \\ &= \frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{h}_t} \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} \\ &= \text{diag}(f'(\mathbf{z}_t)) \mathbf{V}^\top \frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}\end{aligned}\quad (11.26)$$

那么 $\frac{\partial \mathcal{L}_t}{\partial U_{ij}}$ 则改写为

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial U_{ij}} &= \sum_{k=1}^t \mathbb{I}_i(\mathbf{h}_{k-1}^j)^\top \delta_{t,k} \\ &= \sum_{k=1}^t (\delta_{t,k})^i \mathbf{h}_{k-1}^j\end{aligned}\quad (11.27)$$

上式的 $(\delta_{t,k})^i$ 是列向量 $\delta_{t,k}$ 的第 i 个分量, \mathbf{h}_{k-1}^j 是列向量 \mathbf{h}_{k-1} 的第 j 个分量。

故 $\frac{\partial \mathcal{L}_t}{\partial U}$ 为

$$\frac{\partial \mathcal{L}_t}{\partial \mathbf{U}} = \sum_{k=1}^t \delta_{t,k} \mathbf{h}_{k-1}^\top \quad (11.28)$$

所以

$$\frac{\partial \mathcal{L}}{\partial \mathbf{U}} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} \mathbf{h}_{k-1}^\top \quad (11.29)$$

同理, 可以得到 \mathcal{L} 对 \mathbf{W} 和 \mathbf{b} 的梯度为

$$\frac{\partial \mathcal{L}}{\partial \mathbf{W}} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} \mathbf{x}_k^\top \quad (11.30)$$

$$\frac{\partial \mathcal{L}}{\partial \mathbf{b}} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} \quad (11.31)$$

11.1.3.2 损失函数与激活函数

在上面内容的讨论中, 我们发现因为损失函数和激活函数没有定义, 无法写出 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$ 和 $f'(\mathbf{z}_t)$ 的具体形式。为了说明 $\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$ 和 $f'(\mathbf{z}_t)$ 的常见形式, 设利用该循环神经网络解决的问题是分类问题。其中 \mathbf{y}_t 是 one-hot 列向量, 即当 \mathbf{y}_t 属于第 c 类

时, \mathbf{y}_t 的第 c 个分量为 1, 剩余分量皆为 0。分类函数用 softmax 函数, 损失函数设为交叉熵损失函数, 即

$$\begin{aligned}\mathcal{L}_t &= -\mathbf{y}_t^\top \ln(\text{softmax}(\mathbf{o}_t)) \\ &= -\mathbf{y}_t^\top \ln\left(\frac{\exp(\mathbf{o}_t)}{\mathbf{1}^\top \exp(\mathbf{o}_t)}\right) \\ &= -\mathbf{y}_t^\top (\ln(\exp(\mathbf{o}_t)) - \mathbf{1} \ln(\mathbf{1}^\top \exp(\mathbf{o}_t))) \\ &= -\mathbf{y}_t^\top \mathbf{o}_t + \mathbf{y}_t^\top \mathbf{1} \ln(\mathbf{1}^\top \exp(\mathbf{o}_t)) \\ &= -\mathbf{y}_t^\top \mathbf{o}_t + \ln(\mathbf{1}^\top \exp(\mathbf{o}_t))\end{aligned}\tag{11.32}$$

其中 $\mathbf{1}$ 是维度与 \mathbf{o}_t 相同的全 1 列向量, 因为 \mathbf{y}_t 是 one-hot 列向量, 故 $\mathbf{y}_t^\top \mathbf{1}$ 为标量 1。

此时, \mathcal{L}_t 对 \mathbf{o}_t 的梯度为

$$\begin{aligned}\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t} &= \frac{\partial (-\mathbf{y}_t^\top \mathbf{o}_t + \ln(\mathbf{1}^\top \exp(\mathbf{o}_t)))}{\partial \mathbf{o}_t} \\ &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \frac{\partial (\mathbf{1}^\top \exp(\mathbf{o}_t))}{\partial \mathbf{o}_t} \\ &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \left(\frac{\partial \mathbf{1}}{\partial \mathbf{o}_t} \exp(\mathbf{o}_t) + \frac{\partial \exp(\mathbf{o}_t)}{\partial \mathbf{o}_t} \mathbf{1} \right) \\ &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \left(\frac{\partial \exp(\mathbf{o}_t)}{\partial \mathbf{o}_t} \mathbf{1} \right) \\ &= -\mathbf{y}_t + \frac{1}{\mathbf{1}^\top \exp(\mathbf{o}_t)} (\text{diag}(\exp(\mathbf{o}_t)) \mathbf{1}) \\ &= -\mathbf{y}_t + \frac{\exp(\mathbf{o}_t)}{\mathbf{1}^\top \exp(\mathbf{o}_t)} \\ &= \text{softmax}(\mathbf{o}_t) - \mathbf{y}_t\end{aligned}\tag{11.33}$$

然后我们设该循环神经网络的激活函数为常见的 \tanh 函数, \tanh 的形式以及对标量的导数如下

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}\tag{11.34}$$

$$\begin{aligned}\tanh'(x) &= 1 - \left(\frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)} \right)^2 \\ &= 1 - \tanh^2(x)\end{aligned}\tag{11.35}$$

其中 $x \in \mathbb{R}$ 。

所以 $f'(\mathbf{z}_t)$ 为

$$\begin{aligned} f'(\mathbf{z}_t) &= \tanh'(\mathbf{z}_t) \\ &= \mathbf{1} - \tanh^2(\mathbf{z}_t) \end{aligned} \quad (11.36)$$

其中 $\mathbf{1}$ 是与 \mathbf{z}_t 维度相同的全 1 列向量。所以，该循环神经网络的 \mathbf{h}_t 对 \mathbf{z}_t 的梯度为

$$\frac{\partial \mathbf{h}_t}{\partial \mathbf{z}_t} = \text{diag}(\mathbf{1} - \tanh^2(\mathbf{z}_t)) \quad (11.37)$$

11.1.4 长期记忆中的问题

相信读者已经了解了循环神经网络的基本原理，它在理论上可以建立长时间间隔的隐状态之间的依赖关系，类似于我们在很久以前学到的知识到现在都还记得。然而，实际上我们并不能完全记得十年前这个时候是在上哪一节课，同理，简单的循环神经网络实际上也只能学到短期的依赖关系，即其记忆是短期的。为什么呢？让我们来弄清在实现长期记忆功能时具体会出现什么问题。

11.1.4.1 梯度爆炸问题

深度神经网络中基于梯度学习的根本问题是梯度不稳定的问题，当不同隐藏层进行更新的速度差异很大时，就容易产生问题。每一次状态更新，都要用到误差梯度，即确定正确的更新方向和更新的量，从而正确的更新网络权重。然而，训练过程中大的误差梯度可能不断累积，每个节点和层的误差梯度可能总是大于 1.0，呈现指数式增长，导致神经网络模型的权重出现大幅更新的问题，即梯度爆炸问题。这会使网络模型不稳定，某些权重的值可能会大到溢出系统最大数值，导致出现 NaN 值，从而无法继续更新。这就像一个人每天都活在昨天的世界里，放不下过去，看不到未来，最终被往事所吞灭。

让我们从公式的角度更准确地理解梯度爆炸。在 BPTT 算法中，我们最终得到了损失函数 \mathcal{L} 对参数 U 的梯度：

$$\frac{\partial \mathcal{L}}{\partial U} = \sum_{t=1}^T \sum_{k=1}^t \delta_{t,k} \mathbf{h}_{k-1}^\top \quad (11.38)$$

其中 \mathbf{h}_{k-1} 是 $k-1$ 时刻隐藏层状态，可以理解为隐藏层神经元在这一时刻的记忆信息，神经元利用它的记忆可以产生输出，即 $\mathbf{o}_t = V\mathbf{h}_t + \mathbf{b}_h$ 。为了保证隐藏状态 \mathbf{h}_{t-1} 稳定、合理的传递给 \mathbf{h}_t （避免突然失忆，进入“失忆态”，或者与之相反，避免沉溺于过去的记忆，新输入的信息基本被忽略，进入“抑郁态”）就要保证上式中的 $\delta_{t,k}$ 不能出现异常，比如不能在较长一段时间内一直严格单调递增，或者

不能长期严格单调递减。在上一节我们得到的 $\delta_{t,k}$ 计算公式为：

$$\delta_{t,k} = \text{diag}(f'(\mathbf{z}_k))U^\top\delta_{t,k+1} \quad (11.39)$$

注意到 $\delta_{t,k}$ 依赖于 $\delta_{t,k+1}$, $\delta_{t,k+1}$ 又依赖于 $\delta_{t,k+2}$, 最终会依赖于 $\delta_{t,t}$, 从而有：

$$\delta_{t,k} = \prod_{i=k}^{t-1} \left(\text{diag}(f'(\mathbf{z}_i))U^\top \right) \delta_{t,t} \quad (11.40)$$

可见，在反向传播时为了避免 $\delta_{t,k}$ 出现异常， $\text{diag}(f'(\mathbf{z}_i))U^\top$ （一个 m 行 m 列的矩阵）中的元素不能长期都大于 1.0。如果长期大于 1.0, $\delta_{t,k}$ 随着 k 的减小会一项比一项大，从而 \mathbf{h}_k 在损失函数 \mathcal{L} 对参数 U 的梯度的贡献中的比重也会随着 k 的减小而增大，即过去的记忆越来越重要，活在过去，逃避未来，这种“抑郁态”就是长期记忆中经典的梯度爆炸问题。

解决梯度爆炸问题有以下几种方法：

1. 修改网络模型与架构, 比如减少隐藏层层数、使用更小的批尺寸(batch_size)、在 t 与 k 相差较大时停止继续反向传播 (沿时间截断的反向传播);
2. 使用长短期记忆网络, 下一节将会对这种方法详细讲解;
3. 进行梯度截断, 将梯度的模或者将每个梯度值限制在一个范围内;
4. 在计算损失时加入权重正则化项, 检查网络权重的大小, 惩罚产生较大权重值的损失项。

11.1.4.2 梯度消失问题

梯度消失问题是指出向了与梯度爆炸问题相反的另一个极端。因为展开的循环神经网络中靠前的隐藏层梯度接近于 0, 所以参数更新时, 利用不到过去太久的信息进行学习。好比一个人每天都只保留前一天的记忆的百分之九十, 100 天后, 他可能连自己的姓名都不记得了。

从公式的角度来理解梯度消失, 即式11.40中矩阵 $\text{diag}(f'(\mathbf{z}_i))U^\top$ 中的元素长期小于 1.0, $\delta_{t,k}$ 随着 k 的减小会一项比一项小, \mathbf{h}_k 在损失函数 \mathcal{L} 对参数 U 的梯度的贡献中所占的比重 (式11.38) 也会随着 k 的减小而减小, 即当 k 时刻与 t 时刻相距很远时 $\frac{\partial \mathcal{L}}{\partial \mathbf{h}_k}$ 这个梯度消失了, 参数 U 的更新主要依赖于最近几个时刻的记忆 \mathbf{h} 。直观上看, 就好似当要用到某张银行卡的密码时, 由于这个密码上次用到还是一年前, 之后也不知道将密码保存在哪里了, 这种对长期信息进行记忆的功能未能实现, 在极端情况下, 就相当于“失忆”, 这种“失忆态”就是长期记忆中经典的梯度消失问题。

解决梯度爆炸的方法中修改模型 (调参等)、使用长期记忆网络、梯度截断 (给梯度设置一个下限) 也适用于解决梯度消失问题, 此外, 还可以从激活函数

的选择、简化模型的角度来考虑：

1. 注意到式11.40中有用到激活函数的导数 $f'(\mathbf{z}_i)$, 尝试对 logistic 函数和 tanh 函数求导, 可以发现, logistic 函数的导数值小于 0.25, 而 tanh 函数的导数值小于 1.0 (除了在零点为 1.0), 这使 $\text{diag}(f'(\mathbf{z}_i))U^\top$ 有小于 1.0 的趋势, 从而容易导致梯度消失问题。可以使用 ReLU、Leaky ReLU、ELU 等在正区间内导数正好为 1.0 的激活函数, 但各有所长也各有所缺, 这里不再细谈;
2. 回顾本章第一个公式 $\mathbf{h}_t = f(W\mathbf{x}_t + U\mathbf{h}_{t-1} + \mathbf{b})$, 依旧是为了使式11.40中 $\text{diag}(f'(\mathbf{z}_i))U^\top$ 的元素都趋于 1.0, 可以考虑对模型做些简化。比如将激活函数的导数设置为 1.0, 即 $f'(\mathbf{z}_i) = \mathbf{1}$, 然后将 U 简化为 \mathbf{I} , 则 $\mathbf{h}_t = f(\mathbf{h}_{t-1} + W\mathbf{x}_t + \mathbf{b})$, 这就使得 $\text{diag}(f'(\mathbf{z}_i))U^\top$ 直接简化为一个单位矩阵, 从理论上消除了梯度消失和梯度爆炸问题, 但同时也相当于将所有时刻的 \mathbf{h} 平等对待, 降低了模型的表示能力。

11.1.4.3 记忆容量问题

训练一个神经网络需要耗费的 CPU 资源与内存资源往往很大, 需要的时间也很长, 同时也不利于调参。如果把一个 RNN 比作一个大脑, RNN 的训练过程就是大脑思考一个现实问题的过程。由于问题的复杂性, 大脑可能不足以存储所需的记忆容量, 不能快速回忆到所记忆的信息, 甚至总是遗忘一些重要记忆, 这时就称遇到了“记忆容量问题”。为了在一定程度上缓解对记忆容量的高要求, 快速找到对解决问题有帮助的重要记忆, 记忆力机制 (Attention Mechanism) 的引入取得了很好的效果, 详情请见第12章。

11.2 LSTM

为了在有限的空间实现长期记忆功能, 选择性的遗忘以前的重要性低的记忆, 保留重要程度高的记忆从而“不断提高自己的智商”, 这就需要一种综合可行的解决机制——Long Short-term Memory(LSTM)。

11.2.1 L 还是 S?

我们知道, Long 是“长”的意思, Short 是“短”的意思, term 取“学期、期限”之意, Memory 则指我们所求的“记忆”。那么, LSTM 所指的记忆功能到底是长的还是短的呢?

我们记得我们十年前的事情, 记得我们上小学、初中、高中的事情, 这是一种长期的记忆, 但同时我们记得的事情都只是些片段, 并不确切的记得十年前的这个时候我们到底在做一件什么样的具体的事, 此外, 我们更多能回忆起来的是最近发生的事情, 比如上个月、上周我们学到了什么, 这些短片段和近期的记忆



即 short-term memory，其中那些发生在很久之前的记忆片段又称“长的短期记忆”。

让我们对 LSTM 的理解上升到机器学习的层面。在神经网络中，长期记忆隐含了通过长期的迭代传播、更新训练得到的经验、重要信息，而记忆单元在最近一段时期捕捉、更新得到的某些关键信息称为短期记忆。长期记忆的更新是缓慢的、比较稳定的，而短期记忆的更新是频繁的比较剧烈的。换言之，那年，那个夏天，或许有我们终身难忘的事，而昨天、今天我们的所见所闻可能即见即忘。

11.2.2 三门分立

11.2.2.1 门

神奇的 LSTM 到底是怎么实现更长期的记忆功能的呢？其中一个显著的特征就是基于门的机制。LSTM 中的门是一扇魔法门，能以一定的百分比让想通过这道门的信息通过，也即过滤掉一定百分比的信息，就好似我们在学习的时候会吸取百分之多少的信息，抛弃百分之多少的信息。

那么，该设置几道门呢？又怎么确定该让信息通过一扇门的比率呢？人们常说“过去的就让它过去吧”，这就是种遗忘门，简称忘门；我们刷淘宝时无视那些不感兴趣、不需要的商品，虽然看到了，但也只是一扫而过，这是种输入门，简称入门；我们做演讲、和别人谈话时要表达我们的思想、意思，决定什么话该说什么话不该说的，就是输出门，简称出门。三门分立，是怎么各司其职，怎么确定过滤信息的比率，又是怎么共组一个高效而“高情商”的循环单元的呢？下面三小部分将介绍三门的过滤信息比率，下一小节（11.2.3）将介绍什么样的信息会进入这三种门。

11.2.2.2 忘门

忘门，即“往事知多少”，控制多少回忆值得珍存。用 \mathbf{f}_t 表示保留上一层信息的比例， $\mathbf{f}_t \in [0, 1]$ ，其中 f 表示“unforget”， t 表示时刻 t 。 t 时刻会有新的信息输入网络，即 \mathbf{x}_t ，上一时刻的隐藏层的外部状态（循环单元的输出信息）也会影响到信息过滤比，即 \mathbf{h}_{t-1} ，再结合偏置项 \mathbf{b} 和我们的老朋友 logistic，自然的得到了如下计算“回忆珍存率”的式子：

$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + \mathbf{b}_f) \quad (11.41)$$

其中 $\mathbf{f}_t \in \mathbb{R}^{h \times n}$ (h 是隐藏单元的个数)， $\mathbf{x}_t \in \mathbb{R}^{d \times n}$ (d 是单个输入样本的维度)， $\mathbf{h}_{t-1} \in \mathbb{R}^{h \times n}$ ， W_f 和 U_f 是忘门的权重参数， $W_f \in \mathbb{R}^{h \times d}$ ， $U_f \in \mathbb{R}^{h \times h}$ ， \mathbf{b}_f 是偏差参数， $\mathbf{b}_f \in \mathbb{R}^{h \times 1}$ 。上式可用图11.8形象的表示。

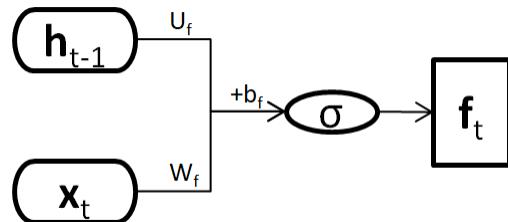


图 11.8: 忘门结构

当 $f_t = \mathbf{0}$ 时，忘门紧锁，记忆单元拒绝接受它的历史，获得“重生”；当 $f_t = 1$ 时，忘门完全敞开，记忆单元继承上一时刻的所有记忆。

11.2.2.3 入门

入门，即“今日知多少”，控制在 t 时刻外部输入信息可以成功流入记忆单元的比例，这个比例用 i_t 表示， $i_t \in [0, 1]$ ， i 指“input”。影响“input”的因素和遗忘门一样，都是 h_{t-1} 和 x_t ，不同的是其中的权重参数和偏差参数，故求 i_t 的公式为：

$$i_t = \sigma(W_i x_t + U_i h_{t-1} + b_i) \quad (11.42)$$

相关矩阵的维度与忘门对应的矩阵一致，上式也可以用图11.9形象的表示。

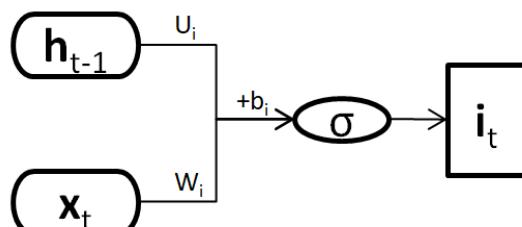


图 11.9: 入门结构

和忘门相似，当 $i_t = \mathbf{0}$ 时，“闭关锁国”，拒绝了一切的输入；当 $i_t = 1$ 时，“海纳百川”，接受所有的输入。

11.2.2.4 出门

出门决定哪些信息应该输出给外部状态 h_t ，用 o_t 表示输出信息的比例， $o_t \in [0, 1]$ ， o 指“output”。想通过出门出去的信息自然是成功进入忘门和入门的信息，但决定多少信息应该从出门出去的比例依旧受 h_{t-1} 和 x_t 影响，试想，我们的一言一语、一举一动，何曾不与当时当下的外界刺激和我们之前的行为所产生的影响有关呢？这样一来，求 o_t 的公式与数据流图呼之欲出：

$$\mathbf{o}_t = \sigma(W_o \mathbf{x}_t + U_o \mathbf{h}_{t-1} + \mathbf{b}_o) \quad (11.43)$$

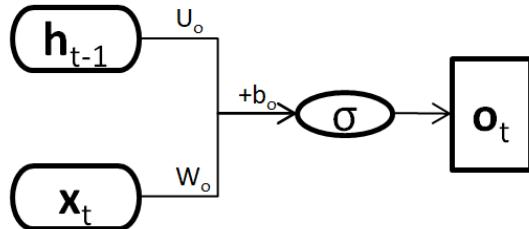


图 11.10: 出门结构

结合三门对信息的过滤比的公式11.41-11.43，可以发现所有门的比率都与当前直接的输入(\mathbf{x}_t)和上一时刻的输出(\mathbf{h}_{t-1} ，这可以理解为t时刻一种间接的输入)有关，即我们决定一条信息是否值得记忆或发出去时应该根据这一时刻接受的信息的质量和过去我们所做的事情的效果来判断。举个出门的例子：当我们确定是否去操场上跑步时，应该根据此时的天气情况、时间(即 \mathbf{x}_t)和我们上一次跑步的效果比如腿疼、喉咙疼、是否摔伤、是否突破个人纪录(即 \mathbf{h}_{t-1})来判断，至于是怎么利用这三道魔法门去确定我们是否去跑步或者我们真的去跑步了的概率的，请看下个回合——内外有别。

11.2.3 内外有别

现在，我们已经确定了三个门的“信息过滤比”，那么，哪些信息想进入这三门呢？即这三门应该安装在哪儿呢？这就要谈及LSTM与RNN另一个重要的不同——内部状态，用 \mathbf{c}_t 表示，其中 t 表示 t 时刻。

\mathbf{c}_t 作为一个存储单元内部的状态，或者说一个存储着信息的矩阵，它记录了到当前时刻为止的所有记忆，并专门负责进行线性的循环的信息传递，同时对本层神经单元的外部状态 \mathbf{h}_t (即对外部的输出)产生一种非线性的影响。

\mathbf{c}_t 就像我们现在大脑里知道的所有信息，有些是昨天大脑里所有的记忆(即 \mathbf{c}_{t-1})经过睡一晚(忘门)后所保留下来的，所以忘门应该作用在 \mathbf{c}_{t-1} 上；有些是今天我们上课、看书、刷手机所获得的(即 \mathbf{x}_t)，或是昨天我们在淘宝上下的订单(即 \mathbf{h}_{t-1} ，注意，下单是昨天发生的事件，是我们昨天的输出，相当于隐藏层在上一时刻的外部状态)，所以入门应该作用在整个存储单元所能获得的除了上一时刻记忆信息(\mathbf{c}_{t-1})之外的所有信息上。为了简便，我们引入一个叫候选状态的变量来表示前面所述的“整个信息”，用 $\tilde{\mathbf{c}}_t$ 表示这个候选状态，其计算公式为：

$$\tilde{\mathbf{c}}_t = \tanh(W_c \mathbf{x}_t + U_c \mathbf{h}_{t-1} + \mathbf{b}_c) \quad (11.44)$$

容易发现 $\tilde{\mathbf{c}}_t$ 与前面三门的过滤比的计算公式很相似，都依赖于当前的输入 \mathbf{x}_t 和上一时刻的输出 \mathbf{h}_{t-1} ，矩阵的维度也一一对应。

然而，门的过滤信息的比率是通过 logistic 直接算出来的一个 0 – 1 之间的百分比组成的矩阵，而候选状态是通过 tanh 激活函数将输入转化为 -1 到 1 之间的数值组成的矩阵，其意义是可以用于计算这一时刻到底该存储哪些信息作为一个存储单元的记忆，或者说，用于计算内部状态 (\mathbf{c}_t)，而这，也是候选状态之所以是“候选”的原因，就好像我们在真正把知识学到手之前要对知识进行不断的分析与筛选，得到我们的“干货” ($\tilde{\mathbf{c}}_t$)，再经过不可避免的遗忘 (\mathbf{f}_t) 与吸收 (\mathbf{i}_t) 过程，最终将知识转化为我们“大脑硬盘”里扎实的可以灵活运用的记忆。

说到这里，希望读者能理解或者写出自己“大脑硬盘”里存储着的记忆的计算公式，即我们隐藏层的内部状态：

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (11.45)$$

这里引入的一种新的运算， \odot ，表示逐元素相乘，即对应位上的值两两相乘，维度不变，都是 $(h \times n)$ ，其意义就是在每一位数据上获取相应比率的信息，体现了门的作用。如果读者仍有疑惑，可以再仔细理解一下前面的推理过程。

至此，我们知道了忘门和入门的“安装位置”，知道了候选状态和内部状态的求解公式，回想上一小节和本小节的主题，容易发现还有出门和外部状态没有解决。如果把内部状态比作我们的本领，外部状态比作我们的行为，那么出门则是连接二者之间的桥，本节最后一个公式由此而生：

$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \quad (11.46)$$

内有 $\tilde{\mathbf{c}}_t$ 、 \mathbf{c}_t ，外有 \mathbf{h}_t ，把握了 LSTM “内外有别”的特征，就把握了 LSTM 不同于 RNN 的精髓之所在。

11.2.4 LSTM 循环单元中的数据流

11.2.2 节有简单地孤立地描述三门的数据流图，让我们结合 11.2.3 节的思想从整体上理解一个小小的 LSTM 循环单元内部是怎样有序的运行的，如图 11.11 所示。

上图就像是由忘门、入门、出门和内外状态组成的一个小迷宫，我们可以通过以下三步的数据流动路径走出这个迷宫：

1. 由上一时刻的输出 \mathbf{h}_{t-1} （间接输入）和当前时刻的直接输入 \mathbf{x}_t ，得到当前时刻循环单元的总体输入，即内部候选状态 $\tilde{\mathbf{c}}_t$ ，同时计算出三个门的过滤比矩阵 \mathbf{f}_t 、 \mathbf{i}_t 、 \mathbf{o}_t ；
2. 通过忘门对上一时刻的内部状态 (\mathbf{c}_{t-1}) 的筛选、入门对当前时刻候选状态

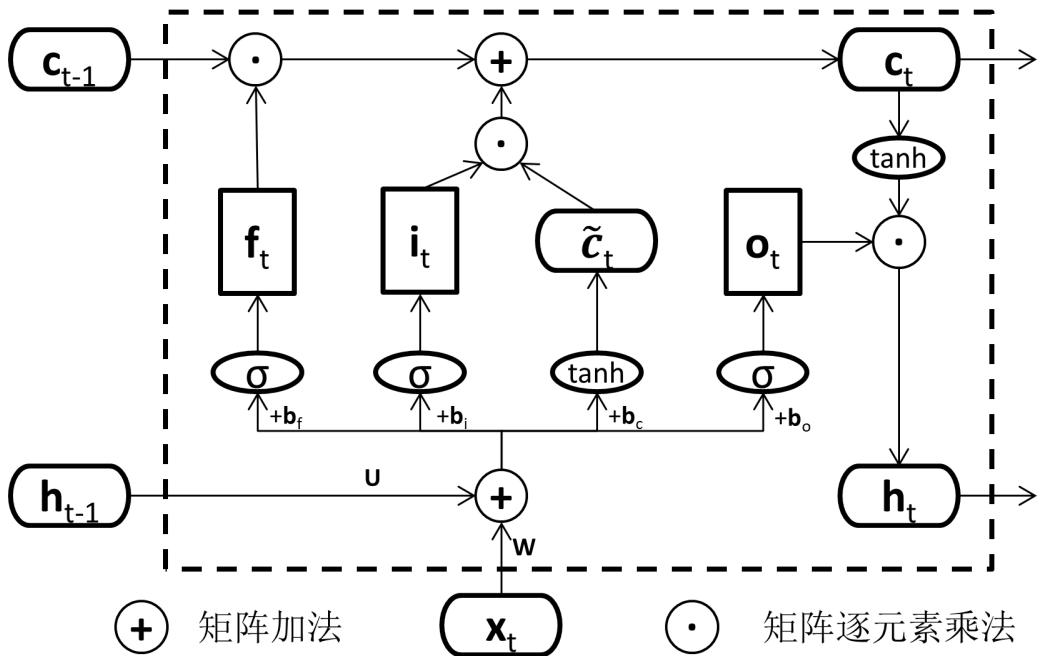


图 11.11: LSTM 隐藏层结点数据流图

信息 (\tilde{c}_t) 的过滤，得到当前时刻循环单元所应存储的内部状态信息，这一信息将被线性的传递给下一时刻的循环单元；

3. 通过出门对当前时刻的内部状态信息的选择作用，得到当前时刻循环单元的外部状态。

11.2.5 LSTM-变体

LSTM 自 1997 年提出以来，有了很多的变体，主要是对各种门的增减和计算方式的改变，主导思想都是模拟人的大脑来让机器进行学习，都可以理解为对人类神经元的简化。下面从对门的增、删、改三个角度介绍三种变体，点到即止，更多内容请移步参考文献。

11.2.5.1 增

ICLR 2019 高分论文前 Top10 《Ordered Neurons: Integrating Tree Structures into Recurrent Neural Networks》中提出了一种名为 Ordered Neurons LSTM 的变体，通过对神经元进行排序，引入更高级的输入门 \tilde{i} 和遗忘门 \tilde{f} 向量，保证了当更新一个单元时其之后的所有有序单元都将被更新。新引入的 \tilde{i} 向量中的值从 $0 \sim 1$ 单调递增， \tilde{f} 向量中的值则从 $1 \sim 0$ 单调递减，它们是通过 cumsum 函数和 softmax 函数复合组成的一种新的激活函数计算得到，最终计算隐藏层内部状态 c_t 时结合了新旧忘门和入门共四个门的影响。详细解释请读 [6]。

11.2.5.2 删

回顾计算内部状态的公式11.45:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (11.47)$$

可知把 \mathbf{c}_t 由忘门删选出的信息和入门筛选出的信息组成。如果把 \mathbf{i}_t 看成一个百分比，比如 $i\%$ ，决定 \mathbf{c}_t 中 $i\%$ 的信息，则 \mathbf{c}_t 中还剩下 $(1-i)\%$ 的“存储空间”，为了简化运算，直接用 $1 - \mathbf{i}_t$ 代替 \mathbf{f}_t ，则有：

$$\mathbf{c}_t = (1 - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t \quad (11.48)$$

这一简化相当于将忘门与入门耦合了，将两个门合并为一个门，即在原 LSTM 基础上删除了一个门。

11.2.5.3 改

peephole 连接，考虑到 \mathbf{c}_{t-1} 的影响。一种经典的对 LSTM 中门的修改的方式是使用“猫眼儿窥视”(peephole connection, 见 [3])，即在计算门的过滤比时考虑上一时刻隐藏层神经元内部状态 (\mathbf{c}_{t-1}) 的影响。比如，求忘门的过滤比时：

$$\mathbf{f}_t = \sigma(W_f \mathbf{x}_t + U_f \mathbf{h}_{t-1} + V_f \mathbf{c}_{t-1} + \mathbf{b}_f) \quad (11.49)$$

可以这样理解上式：信息 \mathbf{c}_{t-1} 想通过忘门 \mathbf{f}_t 进入 \mathbf{c}_t ， \mathbf{f}_t 为了筛选出有用的信息，直接通过猫眼儿窥视 \mathbf{c}_{t-1} 内部到底有多少是水分，有多少是干货，从而决定最终允许 \mathbf{c}_{t-1} 通过的比例，于是在计算 \mathbf{f}_t 时考虑 \mathbf{c}_{t-1} 的影响。

值得一提的是，许多论文在计算门的过滤比时会只在某些门上应用 peephole connection。这种方法可以使得 LSTM 在没有任何短期训练示例的帮助下，学习到长期的由离散时间步长分开的峰值序列之间的细微区别，生成稳定的高度非线性、具有精确时间峰值的序列，适合于时序预测问题 ([3])，但它增加了模型参数，使运算复杂度变大。

11.3 章末总结

本章主要介绍了循环神经网络及其变体。从 RNN 的基本作用与结构出发，在此基础上从网络的输入输出角度介绍了 RNN 的四种应用类型；然后，同其他章节类似，详细讲解了 RNN 中参数更新的运算过程，为了优化求导的计算时间复杂度，引入了随时间方向传播的算法，详细讲解了损失函数对隐藏状态 \mathbf{h} 的参数 U 的求导过程，并在11.1.3.2章节对参数更新过程中遗留的 $\delta_{t,k}$ 中的两项 ($\frac{\partial \mathcal{L}_t}{\partial \mathbf{o}_t}$)



和 $f'(\mathbf{z}_t)$ 的计算做了具体阐述。最后，对神经网络中常见的梯度爆炸问题与梯度消失问题从 RNN 实现长期记忆功能的角度进行了分析并给出了一些常见的解决方法。

RNN 的变体主要围绕 LSTM 展开描述，LSTM 一节主要介绍了 LSTM 的含义、LSTM 在 RNN 基础上的两个主要改进、LSTM 的三种变体。其中，两个主要改进是指 LSTM 中引入了门的机制和内部状态的概念，通过对遗忘、信息输入、信息输出三种活动的模拟，神经元更具“思维能力”了，而内部状态以及候选状态的概念是对记忆体或者是对某时刻的总体记忆的模拟，这让一个神经元更具独立性、完整性了。在此基础上，从增、删、改三个角度介绍了对循环单元结构的三种改进，一方面，是从理论上对人脑的一种模拟，另一方面，是联系实际情况所作的合适的调整。

本章总体知识框架图如下所示：

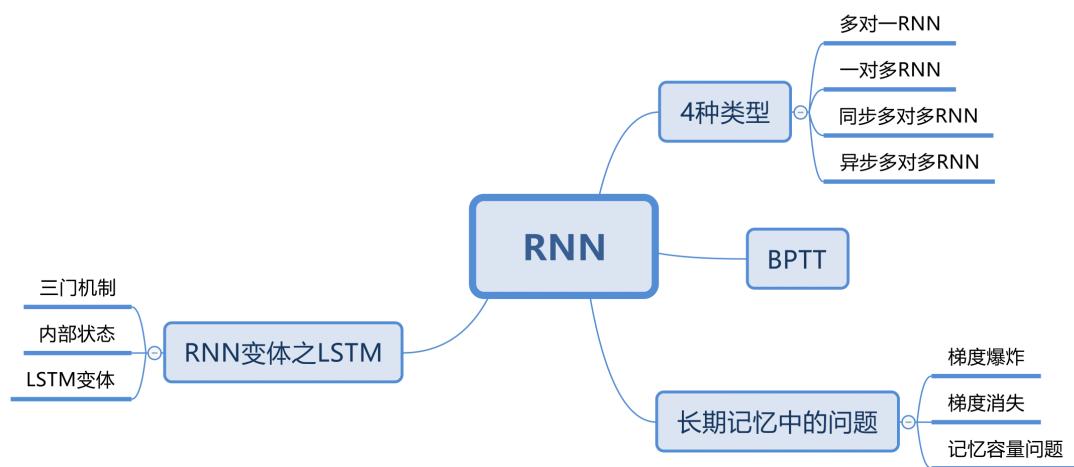


图 11.12: RNN 知识框架

11.4 Advanced Topic

GRUN 门控循环单元网络 (Gated Recurrent Unit Network) 同 LSTM 一样也是一种基于门的 RNN 变体，它没有引入内部状态的概念，而是在 RNN 的隐藏状态 (\mathbf{h}_t) 的基础上引入其候选状态 ($\tilde{\mathbf{h}}_t$)，即对当前时刻净输入的一种模拟，但计算 $\tilde{\mathbf{h}}_t$ 时考虑的因素与 LSTM 中的记忆态的候选状态 ($\tilde{\mathbf{c}}_t$) 不同，引入了一种重置门 (\mathbf{r}_t) 来决定是否需要考虑接受上一时刻的隐藏状态 (\mathbf{h}_{t-1}) 作为候选状态的组成部分之一，因为在 GRU 中更新内部状态 (\mathbf{h}_t) 时已经通过一种更新门 (即耦合的入门与忘门) 将 \mathbf{h}_{t-1} 考虑进去了，再计算候选状态时再考虑一次显得有些多余，而完全不考虑又显得有些不足，因为候选状态 ($\tilde{\mathbf{h}}_t$) 毕竟是对净输入经过简单处理后的候选状态。

DRNN 前面所介绍的都是只有一层隐藏层的 RNN，只有一个神经单元沿时间展开形成的层，如果在网络的输入和输出之间考虑到多个神经单元的信息传递作用，就形成了深层循环神经网络（Deep RNN）。如果层与层之间是从上一层到下一层依次地传递信息，则称为堆叠的循环神经网络（Stacked RNN）；如果层与层之间没有直接干扰，一层是顺着时间的流逝而传递信息，另一层是逆着时间的流逝而传递信息，即将当前时刻获取的信息传递给之前时刻的状态，再将两层的记忆综合起来考虑得到输出，则称这种网络为双向循环神经网络（Bidirectional RNN）。看到这里，读者是否还可以设计一些其他类型的深层神经网络？

RecNN 递归神经网络（Recursive Neural Network）是种基于有向无循环图的 RNN。比如，为了获取相邻时刻的输入之间紧密的关系，可以在计算某时刻的隐藏状态 (\mathbf{h}_t) 时考虑到多个时刻的输入信息。此外，LSTM 也可以基于图的机制来实现 ([8])。

参考文献

- [1] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.
- [2] Jeffrey L Elman. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.
- [3] Felix Gers and Jürgen Schmidhuber. Recurrent nets that time and count. volume 3, pages 189 – 194 vol.3, 02 2000.
- [4] Sepp Hochreiter and Jürgen Schmidhuber. Lstm can solve hard long time lag problems. In *Advances in neural information processing systems*, pages 473–479, 1997.
- [5] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.
- [6] Yikang Shen*, Shawn Tan*, Alessandro Sordoni, and Aaron Courville. Ordered neurons: Integrating tree structures into recurrent neural networks. 2018.
- [7] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.
- [8] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. 2015.
- [9] Paul J Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.
- [10] 邱锡鹏. 神经网络与深度学习, 2019.

第 12 章 Attention Mechanism

12.1 为什么需要注意力机制？

大学两年多来，你是否会因老师常说“这门课其实可以分成很多门课来讲，一个学期的课时是远远不够的”而背上一层沉重的包袱？你是否会因自己的毕业去向的选择过多而迷茫？你是否会因各种因素而无法入睡？其实，基于注意力机制的方法，你可以重点学习那门课所要求掌握的知识点而轻松地拿到一个期末好成绩，而不必从这门课所关联的多个领域的历史起源一直学到各个领域的世界前沿课题研究；你可以一头扎进研究生生涯，在某一个很小很小的研究方向上凿出一个个有影响力的成绩，而不必把大把大把的时间花在思考自己是否适合读研上；你可以轻闭眼睛集中注意力告诉自己现在要入眠、入眠，而不是胡思乱想。当信息量很大时，通过注意力机制可以快速获取到对解决问题最有帮助的信息，把有限的精力集中在有意义的事情上，从而让生活更加美好。

将注意力机制应用到神经网络中也能达到类似的效果。之前我们所学的卷积神经网络、循环神经网络在理论上具有很强的表达能力，可以捕获长距离依赖关系，但实际运用起来因模型复杂度太大、参数太多，常常要等上好几天才能得到一个结果，不可避免地会遇到对计算的需求突破现有计算能力上限的问题，这使得计算能力多年来一直是限制神经网络发展的瓶颈。为了解决网络容量问题或者说信息超载问题、将有限的计算资源使用在最重要的计算任务上，同时使得长期记忆问题得到更好的实现，注意力机制应运而生。

12.2 从仿生角度认识注意力

如果说神经网络是对人脑的一种简化和模拟，那么注意力机制则是对人的记忆（或者说，人获取信息的方式）的一种模拟。让我们想象一下如场景：小明在考研自习室做英语一的阅读理解题目，因为有计时，所以他注意力高度集中在文章和五个选择题上，以至于旁边有人走进自习室、发出开门和关门声，他都毫无察觉，这时，小明的妈妈打电话过来，手机震动，小明从题目中惊醒，立刻走出自习室接电话。仔细观察这个几秒钟的过程，相信读者很容易发现其中所体现的注意力，但读者能否试着把具体的注意力示例抽象出来，并给其分类呢？

小明在仔细读文章、做题，不受他人干扰，即有意地从周围大量的视觉、听觉、触觉、嗅觉等感官上带来的信息中选择一小部分信息（文章和题目）来重点

处理，同时忽略其他不重要信息，这显然可以是一种注意力，并且是有目标的集中注意力；小明被手机震动惊醒，即无意地中断了做题的过程，去处理接电话的事情，这也是一种注意力，一种受外界刺激所驱动、和当前任务没有直接联系的注意力。基于以上观察，我们可以把注意力分成两大类——前者称为聚焦式注意力（Focused Attention），是自上而下的、有意识的，后者称为显著性注意力（Saliency-Based Attention），是自下而上的、无意识的。

实际上，卷积神经网络中的最大汇聚（max pooling）、循环神经网络中的门控机制就可以近似为是一种自下而上的显著性注意力，当在当前环境下突然接受到一个非常重要的刺激信息时，就把网络的注意力转向这个刺激。本章接下来主要介绍聚焦式注意力，以人做阅读理解题目和机器完成阅读理解任务（主要是翻译）为例，“仿人之注意力，造机器之智慧”，讲解当前主流的几种注意力机制。

12.3 注意力评分函数

以阅读理解任务作为例子，给定一篇长文，对文章的内容进行提问，问题的答案可以由文章的关键词或者关键句子给出。假设提出的问题只和段落中的有限几个句子相关，与段落中的其他句子没有太大的关系。那么我们很自然地可以想到，要是只挑出相关片段让后续的神经网络进行处理，那么后续的神经网络的参数势必会大大减少。

我们把一篇文章进行编码预处理后，用 $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n], \mathbf{X} \in \mathbb{R}^{d \times n}$ 表示 n 个输入信息的矩阵，其中 $\mathbf{x}_i \in \mathbb{R}^{d \times 1}$ 。如果给定一个查询向量 \mathbf{q} ，那么应该如何挑选合适的输入信息片段呢？为此我们定义一个注意力分数以及注意力评分函数，来表示评定每段输入信息得到注意力程度的高低。

$$s_i = \text{score}(\mathbf{q}, \mathbf{x}_i) \quad (12.1)$$

其中 s_i 是注意力分数， $\text{score}(\mathbf{x}_i, \mathbf{q})$ 是注意力评分函数，注意力评分函数有几种形式。

(1) 加性模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \mathbf{v}^\top \tanh(\mathbf{W}\mathbf{x}_i + \mathbf{U}\mathbf{q}) \quad (12.2)$$

加性模型实际上是只含有一层隐藏层的前馈神经网络，使用 \tanh 函数作为激活函数。 \mathbf{v} 、 \mathbf{W} 和 \mathbf{U} 是可以通过学习训练的参数。

(2) 点积模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \mathbf{x}_i^\top \mathbf{q} \quad (12.3)$$

点积模型的复杂度与加性模型相差并不大，由于矩阵乘法的并行化，点积模型计算更快且更节省空间。

(3) 缩放点积模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \frac{\mathbf{x}_i^\top \mathbf{q}}{\sqrt{d}} \quad (12.4)$$

与点积模型相比，缩放点积模型多了一个缩放因子 $\frac{1}{\sqrt{d}}$ ，而 d 正是输入信息的维度。在后面的内容中，注意力评分函数之后会用到 softmax 函数进行计算。当 d 过大的时候，点积模型得到的值比较大，导致 softmax 函数的梯度过小，不利于训练。而缩放点积便可以解决这个问题。

(4) 双线性模型

$$\text{score}(\mathbf{q}, \mathbf{x}_i) = \mathbf{x}_i^\top \mathbf{W} \mathbf{q} \quad (12.5)$$

其中 \mathbf{W} 是可以通过学习训练的参数。其实上面的三种模型中，我们都默认 \mathbf{x} 和 \mathbf{q} 是同样的维度，但双线性模型中的 \mathbf{W} 可以不是方阵，这样的好处是便于不同维度的 \mathbf{x} 与 \mathbf{q} 对齐。

12.4 硬性注意力机制 (Hard Attention Mechanism)

还是类比做阅读理解，给定一个题目，硬性注意力就是选择文中最可能相关的一段进行解题，而不去参考别的段落。在上面，我们提到了注意力分数，那么怎么根据注意力分数来进行选择呢？为了更加客观地进行选择，我们引入概率。使用注意力变量 $z \in \{1, 2, \dots, n\}$ 来表示选择的输入信息的索引位置，也就是说 $z = i$ 表示选择了第 i 个输入信息。那么给定 \mathbf{X} 和 \mathbf{q} ，那么选择第 i 个输入信息的概率为 α_i ，即

$$\begin{aligned} \alpha_i &= p(z = i | \mathbf{X}, \mathbf{q}) \\ &= \frac{\exp(s_i)}{\sum_{j=1}^n \exp(s_j)} \end{aligned} \quad (12.6)$$

我们很容易发现，这是一个 softmax 的过程，可以很容易地表示概率。

那么设最后传入后续神经网络的注意力信息为 \mathbf{a} ，那么硬性注意力机制的实现有如下两种方式：

(1) 选取概率最高的输入信息，即

$$\mathbf{a} = \mathbf{x}_j \quad (12.7)$$

$$j = \arg \max_{i=1}^n \alpha_i \quad (12.8)$$

其中 j 是概率最大的输入信息的下标。

(2) 得到选取每个输入信息对应的概率后，通过随机取样进行输入信息的选取。

通过上面内容，可以发现硬性注意力机制，从直观上很符合人类对于注意力的理解。但硬性注意力机制通过最大采样或者随机采样来选择输入信息，这会导致最后的损失函数和注意力分布之间的函数关系不可导，那么这就无法利用通用的反向传播方法进行神经网络的训练了。硬性注意力机制需要通过强化学习等方法对其进行训练。为了易于训练，可以使用软性注意力机制来代替硬性注意力机制。

12.5 软性注意力机制 (Soft Attention Mechanism)

其实在阅读理解中，有些题目的解答并非只看某一段就可以了，甚至需要纵观全局才能进行解题。软性注意力机制与这种解题方式类似。同样设传入后续神经网络的注意力信息为 \mathbf{a} ，选择输入信息的概率为 α_i ，基于公式12.6，我们可以给出软性注意力机制的实现方式：

$$\mathbf{a} = \sum_{i=1}^n \alpha_i \mathbf{x}_i \quad (12.9)$$

在这里，注意力信息 \mathbf{a} 其实就是输入信息的加权平均，用概率大小来表示该段输入的重要程度。

软性注意力机制的示意图如图12.1所示。

我们额外介绍另一种与软性注意力机制很类似的注意力机制——全局注意力机制 (Global Attention Mechanism)，其实现方式如下：

$$\mathbf{c} = \sum_{i=1}^n \alpha_i \mathbf{x}_i \quad (12.10)$$

$$\mathbf{a} = \tanh(\mathbf{W}_c(\mathbf{c} \oplus \mathbf{q})) \quad (12.11)$$

其中 \mathbf{c} 是上下文信息变量， \mathbf{W}_c 为权重矩阵， \oplus 意为向量拼接。简单来说，全局注意力机制就是在软性注意力机制的基础上对查询向量 \mathbf{q} 进行拼接后加了一层神经网络，作为后续神经网络的输入。

与硬性注意力机制相比，软性注意力机制或者全局注意力机制最大的优点便

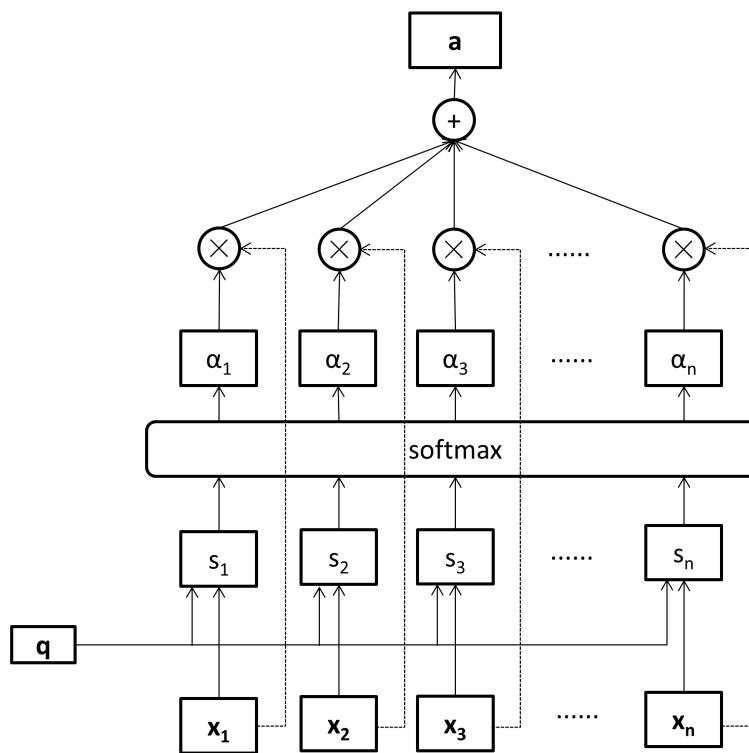


图 12.1: 软性注意力机制示意图

是函数关系可导，可以通过反向传播算法进行训练。但是存在一个不足之处便是，当输入信息的规模越加庞大的时候，对所有输入信息进行加权平均所需的计算力是昂贵的，且加权平均后得到的注意力信息，也很难说真正有效。在下一节，我们介绍一种软硬结合的注意力机制——局部注意力机制，在软硬之间折中，兼具两者优点。

12.6 软硬结合：局部注意力机制 (Local Attention Mechanism)

继续类比做阅读理解，有些题目的解答的不需要通读全文才能做出解答，但只局限于某一段文字也无法进行解答。那么如果进行折中，不进行全文的阅读，只阅读其中几段文字来进行解题，或许准确性和速度都会有不错的提升。局部注意力机制就是与这种方式类似，软硬结合。局部注意力机制，既然是关注于局部的输入信息，那么这个局部的窗口应该怎么确定呢？那我们设这个局部窗口的中心位置位置为 p ，整个窗口的范围是 $[p - D, p + D]$ ，其中 D 是根据实际情况通过经验挑选的长度，是一个正整数。关于局部注意力模型的实现，我们介绍两种模型：

(1) 单调对齐模型 (Monotonic alignment model)，我们可以简称为 local-m 模

型。此模型假设有序且不同的查询变量 \mathbf{q} 按顺序大致与输入信息 \mathbf{x}_i 相关，那么 p 则为：

$$p = i \quad (12.12)$$

那么此时选择第 i 个输入信息的概率与软硬注意力机制中的 α_i 类似，即

$$\alpha_i = \frac{\exp(s_i)}{\sum_{j=\max(1,p-D)}^{\min(n,p+D)} \exp(s_j)} \quad (12.13)$$

其中 $\max(1, p - D)$ 意为取两者中的最大值， $\min(n, p + D)$ 意为取两者中的最小值。 $i \in [\max(1, p - D), \min(n, p + D)]$ ，这意味着并不用计算所有输入信息被挑选的概率。

(2) 预测对齐模型 (Predictive alignment model)，我们可以简称为 local-p 模型。此模型通过神经网络预测位置 p ，具体如下：

$$p = n \cdot \text{sigmoid}(\mathbf{v}_p^\top \tanh(\mathbf{W}_p \mathbf{q})) \quad (12.14)$$

这里的 n 是输入信息的数量，sigmoid 这函数保证了取值在 $(0, 1)$ 之间，这确保了 p 在 $(0, n)$ 之间。然后 \mathbf{v}_p 是一个参数列向量， \mathbf{W}_p 是一个参数矩阵，他们都是通过学习训练的。

而在 local-p 模型中，选择第 i 个输入信息的概率与之前的有所不同，为了尽可能使位置 p 附近的输入信息被挑选的概率尽可能大些，以位置 p 为中心放置高斯分布，具体如下：

$$\alpha_i = \frac{\exp(s_i)}{\sum_{j=\max(1,\lceil p-D \rceil)}^{\min(n,\lfloor p+D \rfloor)} \exp(s_j)} \exp\left(-\frac{(i-p)^2}{2\sigma^2}\right) \quad (12.15)$$

其中 $\lceil p - D \rceil$ 意为 $p - D$ 的向上取整， $\lfloor p + D \rfloor$ 意为 $p + D$ 的向下取整， $i \in [\max(1, \lceil p - D \rceil), \min(n, \lfloor p + D \rfloor)]$ ，一般来说，设置 $\sigma = \frac{D}{2}$ 。

但不管是 local-m 模型还是 local-p 模型，上下文信息变量的计算公式如下：

$$\mathbf{c} = \sum_{i=\max(1,\lceil p-D \rceil)}^{\min(n,\lfloor p+D \rfloor)} \alpha_i \mathbf{x}_i \quad (12.16)$$

简单来说，上下文信息变量就是对窗口 $[p - D, p + D]$ 内的输入信息进行加权平均。

与全局注意力机制相似，最后我们可以得到注意力信息为

$$\mathbf{a} = \tanh(\mathbf{W}_c (\mathbf{c} \oplus \mathbf{q})) \quad (12.17)$$

其中 \oplus 为向量拼接。

由于 D 可以自行调节，即使输入信息的规模越加扩大，基本不会增加额外的计算，局部注意力机制充分保留了硬性注意力机制和软性注意力机制的优点。

12.7 自注意力机制 (Self-attention Mechanism)

自注意力机制由《Attention Is All You Need》 [5] 提出，这篇论文为了降低总的计算复杂度、提高算法的可并行成分、实现序列处理问题中更长期的记忆功能（或者说学习到更长期的依赖关系，见11.1.4），完全抛弃了我们前面所学的 CNN 和 RNN，取而代之的是，在基于“堆叠的编码器-解码器”（Encoder and Decoder Stacks）这一经典模型上，采用了自注意力机制和用于获取位置信息的全连接前馈神经网络。本节不对论文中的各种方法、模型和证明做详细介绍，而是重点解释论文中所用到的关于注意力的方法，同时也忽略了论文中“注意力机制是怎么应用到模型中”的相关内容。笔者强烈推荐这篇论文给初学者，希望本节的内容能激发读者对注意力的兴趣，为读者能读懂这篇论文铺路。

12.7.1 键值对注意力 (Key-Value Attention)

前面三小节主要介绍了三种注意力机制，从我们做阅读理解的角度用三句话来简略的概括就是：依据评分最高（最重要）的一段或者几段来答题（硬性注意力）、依据每一段的评分（重要程度）从每一段抽取不同量的信息来答题（软性注意力）、结合全文总体内容的评分找到切入点再联系切入点的上下文去答题（局部注意力）。

然而，我们做阅读理解时真的会去先花大量时间给每一段算出一个评分以表示其重要程度、然后再去做题吗？非也！我们是给每一段总结出关键词句并以此来代替上文所介绍的数值型评分的，大致步骤分以下四步：

1. 粗读每一段 v_i 的内容（一共 n 段），给其总结出几个关键词或者句子 k_i ，得到一对键值对向量，进而构成“键矩阵” K 和“值矩阵” V ，即 $(K, V) = [(k_1, v_1), \dots, (k_n, v_n)]$ ；
2. 读第一题 q_1 ，将 q_1 与每一段的关键词句 v_i 一一做比较，得到一系列模糊的相关程度，用 $score(q_1, k_1)$ 表示，最终可以得到 $score(q_1, K) = [score(q_1, k_1), score(q_1, k_2), \dots, score(q_1, k_n)]$ ；
3. 当某一段的相关程度 $score(q_1, k_i)$ 很高时，就从这一段 v_i 里多读几句、多获取一些信息，当很低时就只看首尾句、少获取一些信息，最终综合从每一段获取的信息得到对问题 q_1 的答案；
4. 对第二题、第三题... 一直到最后一题的做法和第一题一样，最终完成这篇阅读理解。

将以上这种做题方法应用到机器学习中，可得到如下注意力函数：

$$\text{Attention}(\mathbf{q}_i, (\mathbf{K}, \mathbf{V})) = \text{softmax}(\text{score}(\mathbf{q}_i, \mathbf{K}))\mathbf{V} \quad (12.18)$$

其中， $\text{score}(\mathbf{q}_i, \mathbf{K})$ 是一种可选的打分函数，比较的是问题向量和键矩阵，而不是之前的问题向量和输入向量。上式可理解为将问题向量与所有的键向量相比较后通过 softmax 函数将所有得出的评分归一化，并将评分作用到对应的值向量上，基于此，可以将上式的键矩阵 \mathbf{K} 按行展开：

$$\begin{aligned} \text{Attention}(\mathbf{q}_i, (\mathbf{K}, \mathbf{V})) &= \sum_{j=1}^n \alpha_j \mathbf{v}_j \\ &= \sum_{j=1}^n \frac{\exp(score(\mathbf{q}_i, \mathbf{k}_j))}{\sum_{p=1}^n \exp(score(\mathbf{q}_i, \mathbf{k}_p))} \mathbf{v}_j \end{aligned} \quad (12.19)$$

接下来让我们从矩阵维度的角度来验证上式的正确性并加深理解。设输入信息条数（文章的段落数）为 N ，每条输入信息的维度为 d_v ，对应的关键词句的维度为 d_k ，则值矩阵 $\mathbf{V} \in \mathbb{R}^{n \times d_v}$ ，键矩阵 $\mathbf{K} \in \mathbb{R}^{n \times d_k}$ ，每个问题向量应该与键矩阵能够相比较、匹配，故维度也是 d_k ，即 $\mathbf{q}_i \in \mathbb{R}^{1 \times d_k}$ 。

那么，式12.19中 $\exp(score(\mathbf{q}_i, \mathbf{k}_j))$ 是一个表示相关性程度或者评分的数值，每一个段落信息 \mathbf{k}_j 都与问题向量 \mathbf{q}_i 进行比较、匹配，得到 n 个评分，将这一系列评分通过 softmax 进行归一化后得到的值与维度为 $\mathbb{R}^{1 \times d_v}$ 的 \mathbf{v}_j 上每个数值相乘，得到的是从第 j 段信息或者从第 j 个输入中提取的有用信息。最后，将从每一段信息中提取的有用信息汇总，得到 N 段文字对问题 \mathbf{q}_i 的回答向量 $\text{Attention}(\mathbf{q}_i, (\mathbf{K}, \mathbf{V}))$ ，维度仍为 $\mathbb{R}^{1 \times d_v}$ 。显然，这获取了文章中每一段文字以及每一段的每一个维度上的信息。值得注意的是，这里的回答向量并不是最终模型给出的答案，它可以被看做是一个比较原始的答案，还需要思考、加工后才能成为最终的输出答案。

以上即是对键值对注意力机制的一种基于阅读理解的解释，既然有了公式，不如可视化一下，把公式背后的数据流显示出来，一目了然，见图12.2。

可见，键值对的注意力机制和软性注意力机制很相似——当不引入键矩阵 \mathbf{K} 或者说当 $\mathbf{K} = \mathbf{V}$ 时，键值对注意力就退化为软性的注意力机制。

12.7.2 多头注意力 (Multi-head Attention)

不知读者是否注意到，前面的问题向量 \mathbf{q}_i 一直带着个似乎可有可无的下标 i ，表示第 i 个题目。读者是不是可以大胆想象一下，结合做阅读理解题的经验，还可以有怎样的注意力呢？

试想，我们做题时有时候是不是为了加快解题速度，一次读多个题目，带着问题去读文章，而不是像上文键值对注意力一样机器式的做一题答一题再做一题

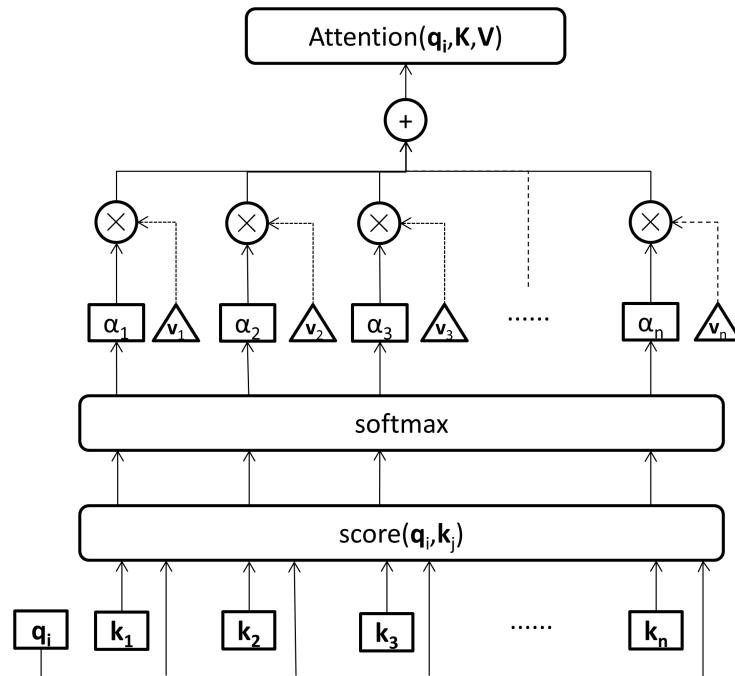


图 12.2: 键值对注意力机制流程图

再答一题？由此，便有了注意力机制中的多头注意力，每一头注意力关注不同的题目，从而能够并行化的答题，仿佛每一时刻有多个大脑在做题。于是，Attention 函数可以同时处理多个问题组成的矩阵 \mathbf{Q} 了：

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Attention}(\mathbf{q}_1, \mathbf{K}, \mathbf{V}) \oplus \dots \oplus \text{Attention}(\mathbf{q}_n, \mathbf{K}, \mathbf{V}) \quad (12.20)$$

\oplus 是将向量拼接成矩阵的符号，一般是按列拼接，但这里是按行拼接，每一行对应一个问题的阶段性解。

继续激发我们的想象力，多头难道就只能是对多个问题的注意吗？其实，多头注意力并不完全就是指关注多个问题， \mathbf{Q} 、 \mathbf{K} 、 \mathbf{V} 都有各自的维度 (d_k 、 d_k 、 d_v)，“多头”也可以指每一头关注一部分维度呀！这，就是下一节将介绍的自注意力机制的核心！聪明的你，是否可以将以上两种机制结合起来，设计出一种属于自己的注意力了呢？

12.7.3 Attention Is All You Need

前面两小节介绍了键值对注意力和多头注意力，似乎与本节的主题自注意力机制没什么关系。其实，自注意力机制本质上就是键值对注意力和多头注意力机制结合在一起以高效实现长期记忆功能的一种综合应用，下面就让我们正式学习论文 [5] 所提出的自注意力机制。

首先，从12.3节中选择一种合适的基于键值对的缩放点积评分函数：

$$\text{score}(\mathbf{Q}, \mathbf{K}) = \frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}} \quad (12.21)$$

其中， $\mathbf{Q} \in \mathbb{R}^{m \times d_k}$ ，表示 m 个查询，每个查询有 d_k 个维度， $\mathbf{K} \in \mathbb{R}^{n \times d_k}$ ，表示一共有 N 个段落或者说输入信息，每个段落总结出了 d_k 个维度的关键信息。二者相乘后 $\mathbf{Q}\mathbf{K}^\top \in \mathbb{R}^{m \times n}$ ，实现了将每个问题从 d_k 个角度与每个关键信息 K 的比较， $\mathbf{Q}\mathbf{K}^\top_{ij}$ 表示从关键信息来看第 j 段的内容与第 i 个问题的匹配程度。

之所以除以 $\sqrt{d_k}$ ，论文中的解释是当 d_k 的值很大时，将 $\mathbf{Q}\mathbf{K}^\top$ 矩阵的值看成一个随机变量后，每一个随机变量就是 d_k 个随机变量的乘积和，于是其方差会变大，从而其增长幅度很大，这会将 softmax 函数推向梯度很小的区域，不利于梯度下降，而除以 $\sqrt{d_k}$ 后可以在一定程度上抵消这种影响。

选定了评分函数后的注意力函数为：

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V} \quad (12.22)$$

其中， $\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)$ 按行对评分进行归一化，每一行的和是 1，其维度仍为 $\mathbb{R}^{m \times n}$ ，值矩阵 $\mathbf{V} \in \mathbb{R}^{N \times d_v}$ ，二者相乘即表示依据每个问题与每个段落的匹配程度，从每一段输入的 d_v 个维度中获取一定比例的信息，得到的每一行 $\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V})_i$ 代表第 i 个问题在 d_v 个维度上获得的初始答案。

接下来可以运用针对维度的多头注意力机制了，每一头关注问题与段落的不同维度，从而实现将问题“分治”，化为一个个简单些的小问题，同时也将段落的维度分层，针对同一个问题下不同的小问题选择对应层上的维度的段落信息来答题。假设一共有 h 个“头”，模型的输入向量（经过预处理后）和输出向量的维度为 d_{model} ，每一头平均分配到 $\frac{d_{model}}{h}$ 个维度上的输入并负责 $\frac{d_{model}}{h}$ 个维度上的输出。论文 [5] 中做了 $d_k = d_v = d_{model}$ 的简化，从而每一头注意力 h_i 的维度统一都是 $\frac{d_{model}}{h}$ 。在此，不失一般性，每一头注意力 h_i 上与 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ 对应的 $\mathbf{Q}'_i, \mathbf{K}'_i, \mathbf{V}'_i$ 矩阵的维度为： $\mathbf{Q}'_i \in \mathbb{R}^{m \times \frac{d_k}{h}}$, $\mathbf{K}'_i \in \mathbb{R}^{n \times \frac{d_k}{h}}$, $\mathbf{V}'_i \in \mathbb{R}^{n \times \frac{d_v}{h}}$ (d_k, d_v 仍是不使用多头注意力机制时的维度，与前两小节符号保持一致）。

那么，如何得到将 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ 降维后每一头 h_i 上的 $\mathbf{Q}'_i, \mathbf{K}'_i, \mathbf{V}'_i$ ？很简单，给每一头 h_i 赋予三个参数 W_i^Q, W_i^K, W_i^V ，用于对 $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ 作线性变换即可：

$$\begin{aligned} \mathbf{Q}'_i &= \mathbf{Q}W_i^Q \\ \mathbf{K}'_i &= \mathbf{K}W_i^K \\ \mathbf{V}'_i &= \mathbf{V}W_i^V \end{aligned} \quad (12.23)$$

易知： $W_i^Q \in \mathbb{R}^{d_k \times \frac{d_k}{h}}$, $W_i^K \in \mathbb{R}^{d_k \times \frac{d_k}{h}}$, $W_i^V \in \mathbb{R}^{d_v \times \frac{d_v}{h}}$ 。将 $\mathbf{Q}'_i, \mathbf{K}'_i, \mathbf{V}'_i$ 的表达式传入每一头

上的注意力函数，有：

$$\text{head}_i = \text{Attention}(\mathbf{Q}W_i^Q, \mathbf{K}W_i^K, \mathbf{V}W_i^V) \quad (12.24)$$

head_i 只关注 $\frac{1}{h}$ 的维度，即 $\text{head}_i \in \mathbb{R}^{m \times \frac{d_v}{h}}$ ，故最后直接将每个注意力头按列拼接：

$$\text{MultiHead}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \quad (12.25)$$

$W^O \in \mathbb{R}^{d_v \times d_{model}}$ ，是将初始答案的维度转化为欲输出维度的网络参数。容易发现上式的每个 head 的计算互不干扰，可以并行计算。

以上就是自注意力机制的核心内容：利用键值对使得模型更加科学合理，使用缩放点积评分函数在计算复杂度较低、可以应用高度优化并行实现的矩阵乘法的情况下避免局部梯度过小问题，采用多头注意力实现并行化计算每一层降低了维度后的注意力函数。

值得一提的是，自注意力机制没有使用 CNN 和 RNN，基于前面的公式得到的评分没有考虑到 (\mathbf{K}, \mathbf{V}) 中行与行之间前后位置排列的顺序，也就是将 (\mathbf{K}, \mathbf{V}) 中对应的两行互换后得到的结果是一样的，因此论文 [5] 中在使用自注意力机制前加入了位置编码信息来捕捉重要的序列信息。当然，自注意力机制也并不是一定要与 CNN 或 RNN 分道扬镳，论文 [4] 就将二者结合起来实现了很好的效果，本章的 Advanced Topic 中将对其进行简单介绍。

12.8 章末总结

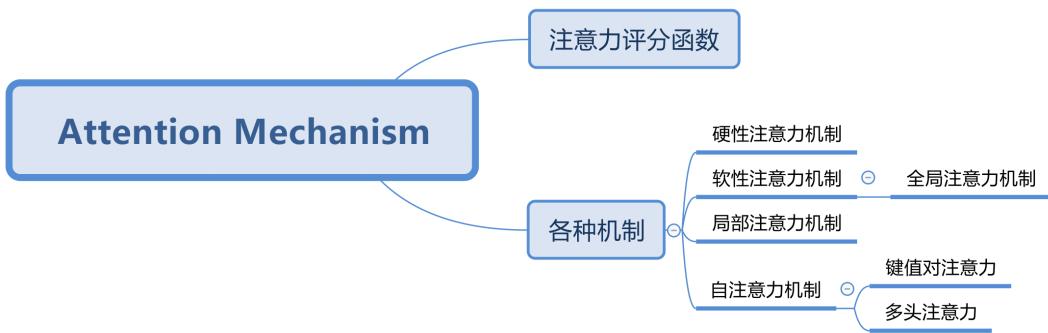


图 12.3: Attention Mechanism 知识框架

本章首先结合现实生活介绍了注意力机制的普遍存在，进而引入注意力机制在机器学习中的作用，随后从仿生的角度去认识注意力，将注意力分成了两大类——自上而下的聚焦式注意力和自下而上的显著性注意力。

显著性注意力类似门控机制的方法前面章节已有介绍，本章着重介绍比较容

易理解的四种聚焦式注意力机制，通过围绕现实生活中人做阅读理解题的例子，从本质上解释了四种聚焦式注意力机制所关注的不同点。

为了模拟人脑的决策判断过程、确定注意点或者注意方向，于是引入注意力评分函数来确定两组信息之间的相关程度，本章的四种注意力机制都是基于评分函数实现的。常见的评分模型有加性模型、点积模型、缩放点积模型、双线性模型，针对具体的问题选择不同的评分模型对注意力机制的效果有很大影响。

做阅读理解时快速锁定最有可能找到答案的一段或几段信息，比如题目中明确说了是从第3段至第7段得出答案，于是就只关注第3至7段，这就是种硬性注意力机制。

当问及通观全文可以看出作者对某某事是什么态度时，我们要综合每一段信息及其重要程度去理解，这时就用到了软性注意力机制。

有时我们做题时先从前几段比如1-3段得出第一题的答案，而后从5-7段、8-11段等等分别得出第二题、第三题以及后续题的答案，这就体现了一种结合软性硬性注意力、假设每题的答案在文章中是按阅读顺序分布的模型，即局部注意力机制中的单调对齐模型；而当根据题意而去判断答案最有可能出现的段落，然后结合从那段的上下文来得出答案时，就体现了局部注意力机制中的预测对齐模型。

本章介绍的第四种聚焦式注意力——自注意力机制，是对键值对注意力与多头注意力的综合运用。键值对注意力就类似给每个段落总结出关键信息，基于关键信息和问题得到评分，再运用软性注意力机制解题；多头注意力机制就类似先把所有题目理解透再去阅读，边读边解题，让解题并行化，同时做所有的题目。自注意力机制深化了多头注意力机制，将一个问题分解成多个子问题，即将问题向量的维度分层，每一头注意力关注对应的层上的维度，从而实现并行化运行。

至此，读者是否发现结合做阅读理解题的方法来理解机器学习中的注意力机制是一个很有效的方法、甚至觉得注意力机制的所有机理就是以做题的方法为依据而产生的？其实，只要我们留心观察，很多学术上的方法模型都可以在现实生活中找到它们的原型或者说“出处”！

12.9 Advanced Topic

Pointer Networks Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. Pointer networks. In Advances in Neural Information Processing Systems, pages 2692–2700, 2015. 指针网络，一种软性的自上而下注意力机制，也是序列到序列模型，输出序列是输入序列的下标（索引）。

Structured Attention Networks Yoon Kim, Carl Denton, Luong Hoang, and Alexander M Rush. Structured attention networks. arXiv preprint arXiv:1702.00887, 2017 结构化注意力，类比小学中学语文课上给好几页长的散文、故事分段

落层次，“形散神不散”

DiSAN [4]

参考文献

- [1] Karim Ahmed, Nitish Shirish Keskar, and Richard Socher. Weighted transformer network for machine translation. *arXiv preprint arXiv:1711.02132*, 2017.
- [2] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.
- [3] Minh-Thang Luong, Hieu Pham, and Christopher D Manning. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.
- [4] Tao Shen, Tianyi Zhou, Guodong Long, Jing Jiang, Shirui Pan, and Chengqi Zhang. DiSAN: Directional self-attention network for RNN/CNN-free language understanding. page 10, 2018.
- [5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. pages 5998–6008, 2017.
- [6] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. *arXiv preprint arXiv:1502.03044*, 2015.
- [7] 邱锡鹏. 神经网络与深度学习, 2019.

第 13 章 聚类

13.1 引言

我们在外出旅游，漫步于大自然中时，身边常常会看到各种各样的花朵，我们有的时候很难一一辨别出每朵花分别是具体哪种花，但是我们可以根据花的外表、形状等将看上去相近的花归为一类，例如，从尺寸上来说有一类花的叶 3-5 枚，花瓣宽 1-2 厘米，花被片长 5-7 厘米，柱头横截面积大约为 3 平方厘米。具有相同或相近特征的物品我们都可以认为它们更有可能是同一类东西。这是我们可以观察形体而得出的结论，那么当我们在面对数以亿计的数据呢？数字的特征远不如现实物品表现得那么明显，我们在面对这样子的数据时该如何将数据分类呢？这也就需要我们的聚类算法来实现了。

同时我们要考虑四个问题：1. 如何计算花与花之间的相似度（或特征差距）呢？2. 此为无监督学习，我们需要将它们分成几类，使得分类结果既有代表性又有差异性呢？3. 该机器学习任务的损失函数是什么呢？4. 在此无监督任务中，我们要如何评定模型分类效果的好坏呢？

13.2 聚类方法与性能度量指标

首先我们需要对聚类模型有一个大致的了解，聚类是将数据以多种方式划分为多个聚集的簇，使得每个数据实例均可以被划分至某个类中，或是被判为异常数据的无监督学习方法。按照划分的方式，我们可以将聚类分为 3 类：1. 划分聚类 2. 层次聚类 3. 密度聚类。在本章中，我们将会介绍这三种聚类方法，其中以 K-Means 为基础的各类聚类方法尤为重要。

当我们使用了上述的聚类方法完成了对数据的分类时，由于并没有这些数据的实际分类标签，所以无法使用有监督学习中的方法来判别最终分类的质量。在此，我们将介绍用于聚类的常见的两大方法：内部方法与外部方法。他们之间的区别在于外部方法有参考模型而内部方法没有参考模型。

内部方法：

假设数据集 $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$ ，模型对其聚类的结果为 $\{C_1, C_2, \dots, C_K\}$ ，每个簇的质心用 μ 表示，向量之间的距离用 $d(\mathbf{x}_i, \mathbf{x}_j)$ 表示（细节将在后面提及）。因此有以下四个度量变量的定义：

$$\text{同簇样本间平均距离} : avg(C_i) = \frac{2}{|C_i|(|C_i| - 1)} \sum_{1 \leq i \leq j \leq |C_i|} d(\mathbf{x}_i, \mathbf{x}_j), \quad (13.1)$$

$$\text{同簇样本间最大距离} : diam(C_i) = \max_{1 \leq i \leq j \leq |C_i|} d(\mathbf{x}_i, \mathbf{x}_j), \quad (13.2)$$

$$\text{簇间最近样本距离} : d_{min}(C_i, C_j) = \min_{\mathbf{x}_k \in C_i, \mathbf{x}_t \in C_j} d(\mathbf{x}_k, \mathbf{x}_t), \quad (13.3)$$

$$\text{簇间质心距离} : d_{cen}(C_i, C_j) = d(\mu_i, \mu_j). \quad (13.4)$$

根据以上四个度量变量, 定义两个内部指标: **Davies – Bouldin Index** 与 **Dunn Index** (简称为 **DBI** 与 **DI**), 定义如下:

$$DBI = \frac{1}{K} \sum_{i=1}^K \max_{j \neq i} \left(\frac{avg(C_i) + avg(C_j)}{d_{cen}(C_i, C_j)} \right), \quad (13.5)$$

$$DI = \min_{1 \leq i \leq K} \left\{ \min_{i \neq j} \left(\frac{d_{min}(C_i, C_j)}{\max_{1 \leq l \leq K} diam(C_l)} \right) \right\}. \quad (13.6)$$

在聚类完成后可以计算 **DBI** 与 **DI** 的值, **DBI** 的值越小表示聚类效果越好, **DI** 的值越大表示聚类效果越好。

外部方法:

假设数据集 $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, 模型对其聚类的结果为 $\{C_1, C_2, \dots, C_K\}$, 参考模型对该数据集 \mathcal{T} 的划分结果 $C^* = \{C_1^*, C_2^*, \dots, C_K^*\}$. 令 λ 与 λ^* 分别表示与 C 和 C^* 对应的簇标记向量. 定义以下变量:

$$a = |SS| \quad SS = \{(\mathbf{x}_i, \mathbf{x}_j) | \lambda_i = \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \quad (13.7)$$

$$b = |SD| \quad SD = \{(\mathbf{x}_i, \mathbf{x}_j) | \lambda_i = \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}, \quad (13.8)$$

$$c = |DS| \quad DS = \{(\mathbf{x}_i, \mathbf{x}_j) | \lambda_i \neq \lambda_j, \lambda_i^* = \lambda_j^*, i < j\}, \quad (13.9)$$

$$d = |DD| \quad DD = \{(\mathbf{x}_i, \mathbf{x}_j) | \lambda_i \neq \lambda_j, \lambda_i^* \neq \lambda_j^*, i < j\}. \quad (13.10)$$

基于上面的定义, 可以导出下面这些常用的聚类性能度量外部指标, 这些指标值越大则代表聚类结果越好:

$$\text{Jaccard 系数 JC} : JC = \frac{a}{a + b + c}, \quad (13.11)$$

$$\text{Fowlkes and Mallows 指数 FMI} : FMI = \sqrt{\frac{a}{a+b} \cdot \frac{a}{a+c}}, \quad (13.12)$$

$$\text{Rand 指数 RI} : RI = \frac{2(a+d)}{m(m-1)}. \quad (13.13)$$

13.3 划分聚类

由于观察记录的无标签数据均在欧氏空间中且为连续值 [6], 令数据集 $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$. \mathbf{x}_i 为第 i 个植物的样本向量, \mathbf{x}_i^j 即为第 i 个植物第 j 个特征尺寸的

大小. 那么样本 i 与样本 k 之间的差距即可用两个向量之间最基本的欧式距离来表示:

$$d(\mathbf{x}_i, \mathbf{x}_k) = \|(\mathbf{x}_i - \mathbf{x}_k)\|_2. \quad (13.14)$$

上式 (1.14) 也就解决了引言中提出的第一个问题. 那么现在就可以依据计算而得的向量之间的距离, 将非常接近的点划为同一类中, 从而使得最终整个数据集 \mathcal{T} 被划分为几个簇, 这种聚类方式就是划分聚类, 在这一节中将重点介绍 K-Means 聚类算法 [4][9] 及其改进, 其中的 K 即为算法最终要可划分的类别数目.

13.3.1 K-Means

在 K-Means 算法中, 需要先初始化 K 个簇, 然后在训练过程中根据损失函数不断改变簇的分布, 使得最后损失函数达到最优. 在这里, 我们就需要思考三个问题: 1. 最终需要将物品分为多少个类呢? 2. 如何表达一个簇的位置或状态呢? 3. 如何初始化 K 个簇呢?

首先可以假设观察到的花朵共分为三个类, 即 $K = 3$. 当然之后若有更具体的需求, 可以将 K 置为任意正数. 若 $K =$ 样本数 N , 则认为每个样本均为独自一个类, 属于极限状况, 所以 K 的值应在 $[1, N]$ 之间.

针对第二个问题, 常使用一个簇的质心 (centroid) 来表示一个簇的位置, 即该簇的均值向量, 其计算方式如下:

$$\mu_j = \frac{\sum \mathbf{x}_i}{|\mathbf{C}_j|} \quad s.t. \quad \mathbf{x}_i \in \mathbf{C}_j. \quad (13.15)$$

为了直观地在图中显示一个簇的质心, 仅选取样本中的两个特征来进行计算, 具体如图13.1所示.

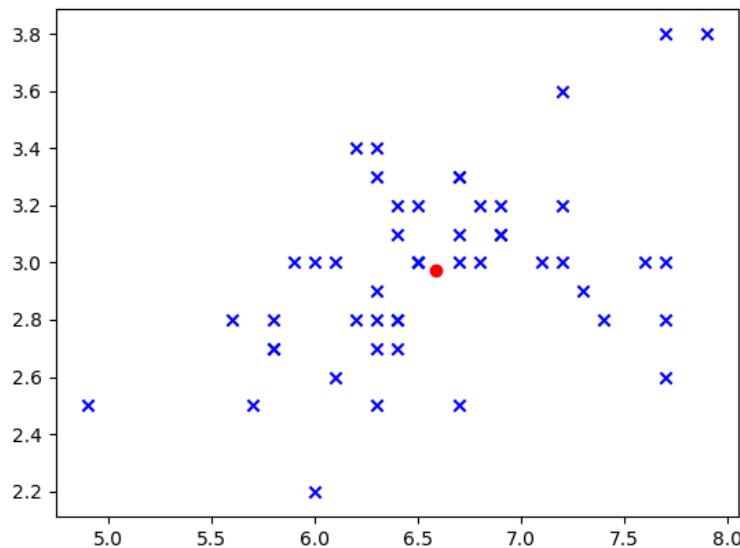


图 13.1: 簇质心位置图

至此已经解决了第一个与第二个问题。第三个问题的解决方法有很多. 例如最简单的方法就是多次随机初始化 [5], 每一次均随机抽取 K 个点作为初始质心,

然后进行训练。最后选择损失函数最小的模型中的初始质心即可。但该方法在数据量大时耗费的时间非常大，且效果好坏的不确定性非常依赖于我们是否可以随机取得性能较好的初始质心。初始质心选取的好坏对算法最终性能影响很大，比如，若初始质心都非常靠近彼此，那么训练所需的收敛时间非常长。所以，我们后面需要思考耗时更少且性能更佳的方法，但现在先使用这种最简单的方法来初始化质心。

接下来需要定义该模型的损失函数。在分类问题中，理想的情况是同一个簇内的样本能够尽可能地接近该簇的质心，即同一个类中的样本相似度更高，它们都属于同一个类的可能性更高。由此，给出如下的平方误差损失函数 SSE 定义 [2]:

$$SSE = \sum_{j=1}^K \sum_{\mathbf{x}_i \in C_j} \|\mathbf{x}_i - \mu_j\|_2. \quad (13.16)$$

由于要找到该数据集中使得损失函数达到全局最优值的划分方法需要考虑样本集所有的组合方式，所以这是一个 NP 难问题 [7]。所以 K-Means 采用了贪心的方法，通过迭代优化 [3] 来逐步找出使得损失函数最少或满足特定条件便终止的划分方案。K-Means 算法描述如下图所示。

Algorithm 21: K-Means

Input: 数据集 $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, 指定簇数 K

1 从 \mathcal{D} 中随机选择 K 个样本作为初始质心向量 $\{\mu_1, \mu_2, \dots, \mu_K\}$

2 **repeat**

3 令 $C_i = \emptyset (1 \leq i \leq K)$

4 **for** $j = 1, 2, \dots, n$ **do**

5 计算样本 \mathbf{x}_j 与各质心向量 $\mu_i (1 \leq i \leq K)$ 的距离: $d_{ji} = \|\mathbf{x}_j - \mu_i\|_2$

6 根据最近的质心确定 \mathbf{x}_j 的簇标记: $\lambda_j = \arg \min_{i \in \{1, 2, \dots, K\}} d(\mathbf{x}_j, \mu_i)$

7 将样本 \mathbf{x}_j 划入相应簇: $C_{\lambda_j} = C_{\lambda_j} \cup \{\mathbf{x}_j\}$

8 **end**

9 **for** $i = 1, 2, \dots, K$ **do**

10 计算新的质心向量: $\mu'_i = \frac{1}{|C_i|} \sum_{\mathbf{x} \in C_i} \mathbf{x}$

11 **if** $\mu'_i \neq \mu_i$ **then**

12 将当前均值向量 μ_i 更新为 μ'_i

13 **else**

14 保持当前均值向量不变

15 **end**

16 **end**

17 **until** 当前质心向量均未更新;

Output: 簇划分: $\{C_1, C_2, \dots, C_K\}$

在上述流程描述中，可以注意到当这次迭代之后的簇质心与上一次的簇质心



完全一致时，算法则直接停止训练。当然这种情况的发生需要非常多的训练轮数才可以发生，所以为了避免运行时间过长，还可以手动设置最大训练轮数或最小质心变化阈值。当目前轮数达到最大训练轮数或调整幅度小于变化阈值时，则可以停止训练。当然也有其他停止条件 [8]，并不局限于本文提到的这几种。

现在已经得到一个可以用于分类的聚类算法了，但是我们仔细考虑一下，其实该方法在某些细节上还是太粗糙了，而且似乎只能用于符合某种条件的数据集中。我们可以做更多的修改与拓展，使得该方法可以达到更好的效果，以及可用于更多种类的数据中。以下是可以进行优化与拓展的地方：

- 确定初始质心的方法除了多次随机初始化以外还有一些更加可靠的方法吗？因为原方法中我们进行的有限尝试中不一定就能找到最佳的初始化质心，这是很靠运气的一件事情。(Tips: 可以从实际意义中考虑簇与簇之间应该满足什么关系可以使得不同类之间的差距越大，从而使得分类结果的置信度更高？)
- 当前的任务或是一些明确了类别数的任务中， K 几乎是可以固定为某个数值的。那么对于一些无法使用先验知识确定类别的任务呢？ K 值不仅对最终分类的效果影响很大，还与数据实际意义关联很大。假设当 $K=N$ 时，平方损失损失函数值为 0，但这种每个实例各属于一个类的分类结果显然是不符合任务需求的。那么我们又该如何在不使用暴力穷举的方法下高效确定性能最佳的 K 值呢？
- 在面对数据中的无序离散属性时，欧式距离的使用似乎就不符合该属性的意义。因为其仅表示该属性的类别不同，而无数值上的大小关系。例如，我们在样本特征中加上一个花瓣形状的离散型属性，因此，面对此类属性，我们需要定义新的距离用于计算。同时，对于有序属性，我们是否还可以用其他距离来提高模型的准确度呢？
- 我们在提出的普通 K-Means 方法中可以看出，当数据量很大时，其所耗费的时间非常大。有没有其他的方法可以加速该方法的计算呢？

针对问题 1，我们希望最后划分的簇集合中，每个簇内的样本尽量离其质心近，且簇与簇之间的距离尽量大（使用两个簇的质心距离作为衡量）。这样可以使得类与类之间的差距更大，同类之间的样本相似度更高，分类结果的置信度更高。基于这种思想，K-Means++[1] 即在初始化质心的问题上进行了优化。算法流程如下图所示。

这种方法并不是完美的，它在选取新的初始质心时，确保其与已确定的质心最远的点，从而使得初始质心是散开的，利于训练后得到的簇也是较为分散的。但是该方法容易选中离群点（异常点），使得分簇效果受异常数据影响而性能较差。同时该选取初始质心的方法导致整个模型训练的时间开销远大于 K-Means。该方法在一定程度上解决了初始化质心的问题，但更加适合用于数据量小的任务中。

Algorithm 22: K-Means++

Input: 数据集 $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, 指定簇数 K

- 1 从数据集中随机选择一个点为第一个聚类中心 μ_1
- 2 **while** 仍未选好共 K 个初始质心 **do**
- 3 对于数据集中每一个 \mathbf{x}_i , 计算其与已选择的质心之间最近的距离, 即 $D(\mathbf{x}_i) = \arg \min \| \mathbf{x}_i - \mu_r \|_2 (r = 1, 2, \dots, k_{selected})$
- 4 选择一个新的点作为新的初始质心, 选择原则是: 选择 $D(\mathbf{x}_i)$ 最大的点 \mathbf{x}_i
- 5 作为新的初始质心
- 6 **end**
- 7 使用选出的 K 个质心作为 K-Means 的初始质心并运行 K-Means

Output: 簇划分: $\{C_1, C_2, \dots, C_K\}$

针对问题 2, 对于无法较准确预先确定 K 值的任务, 可以使用以下方法来完成 K 值的确定。一个是手肘法, 另一个则是轮廓系数法

手肘法: 利用误差平方和的损失函数, 不断增加 K 的数值, 并画出 X 轴为 K , Y 轴为 SSE 的图像, 选择 SSE 下降幅度减小较明显的 K 值即为较优聚类数。其原理是, 随着聚类数的增加, 当 K 小于真实聚类数时, K 值增大可以增加每个簇的聚合程度, 从而使得 SSE 下降较明显。当 K 值大于真实聚类数时, 分类过于细化, 得到的回报会变小, SSE 变化幅度也会变小。我们可以使用 Iris 数据集进行模拟, 效果如图 13.2 所示。

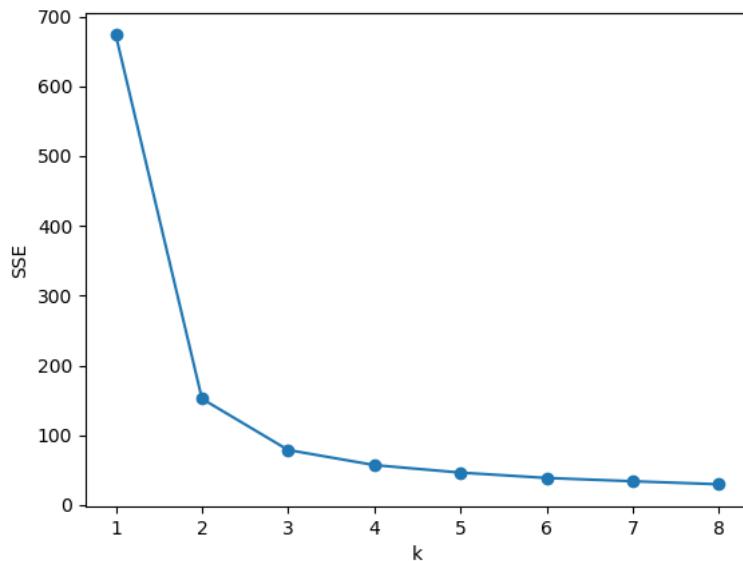


图 13.2: 手肘法

从图中我们看出, 在 $K = 3$ 之后 SSE 的变化幅度明显变小, 因此我们根据上述的方法, 选择 $K = 3$ 作为我们的 K 值。由于这张图很像人的手肘, 所以也成为手肘法。事实上, Iris 数据集是有标签的, 一共有三类, 因此我们由此可以得出这个方法一定的可靠性。

轮廓系数法 该方法使用轮廓系数作为衡量选取 K 值效果好坏的指标。首先针对某个样本 \mathbf{x}_i 定义簇内不相似度 与 簇间不相似度。簇内不相似度指的是簇内

各样本 \mathbf{x}_i 到同簇其他样本 \mathbf{x}_j 的平均距离 $d_{avg}(\mathbf{x}_i, \mathbf{x}_j)$. 簇间不相似度指的是 样本 \mathbf{x}_i 到其他簇 C_j 所有样本的平均距离中的最小值 $\min d(\mathbf{x}_i, \mathbf{x}_k) (\mathbf{x}_k \in C_j, j = 1, 2, \dots, K)$

根据样本 \mathbf{x}_i 的簇间不相似度 $a(\mathbf{x}_i)$ 与簇间不相似度 $b(\mathbf{x}_i)$, 定义样本 \mathbf{x}_i 的轮廓系数 $s(\mathbf{x}_i)$:

$$s(\mathbf{x}_i) = \frac{b(\mathbf{x}_i) - a(\mathbf{x}_i)}{\max \{a(\mathbf{x}_i), b(\mathbf{x}_i)\}}. \quad (13.17)$$

对于样本 \mathbf{x}_i , $s(\mathbf{x}_i) \in [-1, 1]$, 若 $s(\mathbf{x}_i)$ 接近 1, 则表示样本 \mathbf{x}_i 聚类结果合理, 若 $s(\mathbf{x}_i)$ 接近-1, 则表示样本 \mathbf{x}_i 聚类结果不合理, 应该被分至其他类. 数据集聚类结果的轮廓系数是所有样本的轮廓系数均值 $\frac{\sum_{\mathbf{x}_i \in \mathcal{T}} s(\mathbf{x}_i)}{n}$, 该值越大则代表分类结果越好. 针对我们的数据集, 该方法的效果如图 3 所示. 虽然在理论中轮廓系数越大的对应的 K 值越好, 但是该方法并不绝对, 因为还需要考虑数据中是否有异常点等情况, 所以轮廓系数法与手肘法都只能作为选择 K 值的参考方法, 并不能精确地确定 K 值.

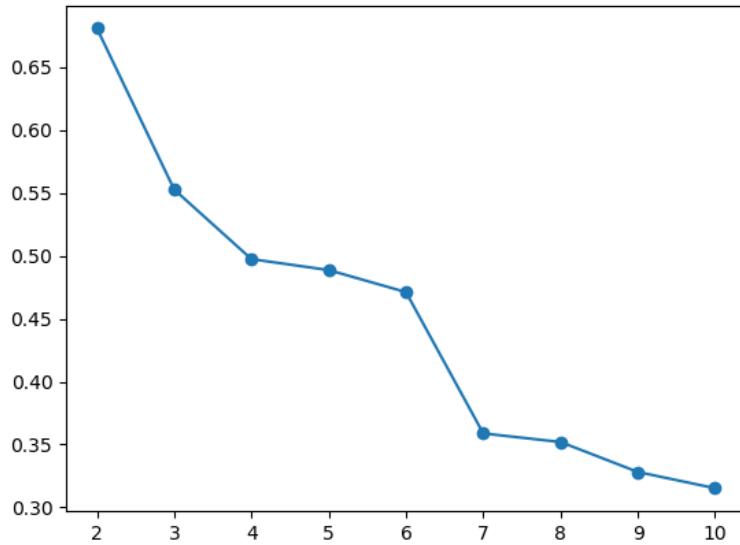


图 13.3: 轮廓系数法

至于问题 3, 在有序属性上除了使用欧式距离, 我们还可以使用其他距离。例如: 1. 闵可夫斯基距离 2. 马氏距离. 对于两个 m 维变量 \mathbf{x}_i 与 \mathbf{x}_j , 其闵可夫斯基距离定义如下:

$$\text{Minkowski Distance: } d(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^m |\mathbf{x}_i^k - \mathbf{x}_j^k|^p \right)^{\frac{1}{p}}, \quad (13.18)$$

其中 p 为可变参数, 当 $p = 1$ 时则为曼哈顿距离, $p = 2$ 时则为欧式距离, $p \rightarrow +\infty$ 时则为切比雪夫距离. 但是该距离有非常大的缺点即其将所有特征的量纲视为一致且没有考虑各特征的分布不同导致的同样数值上的差值代表着对最终距离的不同影响的问题. 在该问题上, 我们可以使用马氏距离来解决. 假设 \mathbf{x}_i 与 \mathbf{x}_j 服从于同一分布.

$$d(\mathbf{x}_i, \mathbf{x}_j) = \sqrt{(\mathbf{x}_i - \mathbf{x}_j)^T \mathbf{S}^{-1} (\mathbf{x}_i - \mathbf{x}_j)}. \quad (13.19)$$

其中矩阵 \mathbf{S} 是数据集的协方差矩阵. 我们取数据集中的某两个实例数据作出计算, $\mathbf{x}_i = [5.1, 3.5, 1.4, 0.2]$, $\mathbf{x}_j = [4.9, 3.0, 1.4, 0.2]$. 该数据集计算所得的协方差矩阵的逆矩阵 \mathbf{S}^{-1} 如下:

$$\begin{pmatrix} 10.31469875 & -6.71318923 & -7.31448253 & 5.739951 \\ -6.71318923 & 11.05841725 & 6.48058913 & -6.17093237 \\ -7.31448253 & 6.48058913 & 10.03167858 & -14.5137665 \\ 5.739951 & -6.17093237 & -14.5137665 & 27.69363502 \end{pmatrix}$$

那么最终计算而得的两个实例之间的距离 $d(\mathbf{x}_i, \mathbf{x}_j) = 1.35446$.

马氏距离不受量纲的影响, 且考虑了不同属性对距离的影响, 抵消了属性之间的相关性, 使得距离计算更加适用于更复杂的数据集. 但是它也有缺点: 它要求数据集的样本数大于向量维度数且易夸大只有微小变化的特征的影响.

为了计算无序属性上的距离度量, 可以使用 VDM 计算. 令 $m_{u,a,i}$ 表示簇 C_i 在属性 u 上取值为 a 的样本数, K 为簇总数. 则属性 u 上两个离散值 a 与 b 之间的 VDM 距离为:

$$VDM_p(\mathbf{x}_i, \mathbf{x}_j) = \sum_{i=1}^K \left\| \frac{m_{u,a,i}}{m_{u,a}} - \frac{m_{u,b,i}}{m_{u,b}} \right\|^p. \quad (13.20)$$

p 为可选参数, 可与稍后介绍的闵可夫斯基距离共同使用. 然后可以将 VDM 与闵可夫斯基距离结合以处理混合属性 (m 个有序属性, $n-m$ 个无序属性):

$$MinkovDM_p(\mathbf{x}_i, \mathbf{x}_j) = \left(\sum_{k=1}^m |\mathbf{x}_i^k - \mathbf{x}_j^k|^p + \sum_{k=m+1}^n VDM_p(\mathbf{x}_i, \mathbf{x}_j) \right)^{\frac{1}{p}}. \quad (13.21)$$

使用上述介绍的额外的距离, 聚类算法可以更好地适应各种各样的数据集. 当然, 具体使用哪一种距离, 我们需要具体分析数据集的特点来选择.

至此, 我们也完成了引言中的第二个问题. 再来关注问题 4, 在观察一开始提出的 K-means 算法过程后, 我们可以得出, 该算法耗时最多的就是每个样本逐个计算与每个当前簇质心的距离部分. 为了减少该部分的计算量, 我们可以使用 Elkan K-means 算法, 其借助三角形性质简化了距离计算量, 即下面两个法则:

法则 1: 对于样本 \mathbf{x} 与质心 μ_{j1}, μ_{j2} , 若已计算 $d(\mu_{j1}, \mu_{j2})$, 则当发现 $2d(\mathbf{x}, \mu_{j1}) \leq d(\mu_{j1}, \mu_{j2})$ 时, 则表示 $d(\mathbf{x}, \mu_{j2}) \geq d(\mathbf{x}, \mu_{j1})$, 也就不需要计算 $d(\mathbf{x}, \mu_{j2})$.

法则 2: 若已知数据点 \mathbf{x} 与两个不同簇的质心 μ_i 与 μ_j , 则有下式成立:
 $d(\mathbf{x}, \mu_j) \geq \max 0, d(\mathbf{x}, \mu_i) - d(\mu_i, \mu_j)$.

同时, 还可以采用 K-means++ 的并行版本 K-Means|| 来加速计算. K-Means++ 最主要的缺点在于其需要遍历数据集 K 才可得到 K 个初始质心, 且当前质心计算依赖于前面所得质心结果, 故难以并行化. 而 K-Means|| 则每次遍历取样 $O(K)$ 个样本, 重复该过程约 $O(\log n)$ 次, 共取得 $O(K \log n)$ 个样本点组成的集合. 该集合以常数因子近似于最优解. 然后再对这 $K \log n$ 个点聚类形成 K 个质心. 这样子即

可用并行的方法来选取初始质心。

至此，我们可以设计出性能较为优秀、更加贴合实际需求的 K-means 聚类算法了。当然后面还会介绍更多其他类型的聚类方式，但是我们还要清楚 K-means 的优点与缺点，在实际使用中根据需求来决定是否要使用该算法。

K-Means 的优点：

1. 容易理解与使用以及实现。
2. 时间复杂度为 $O(tkn)$ ，可近似看作线性算法。

K-Means 的缺点：

1. 对异常数据点非常敏感，聚类效果易受其影响。故可用簇的中位数向量代替均值向量作为质心向量。
2. 该算法的效果依赖于 k 值，需要使用者不断尝试从而得出较好的 k 值。选择 k 值的方法已在前面给出。

13.4 层次聚类

除了划分聚类，还有将数据集分为不同层次，形成树状聚类结构，这种聚类方法称为层次聚类。它包括自底向上与自顶向下两种聚类方法。

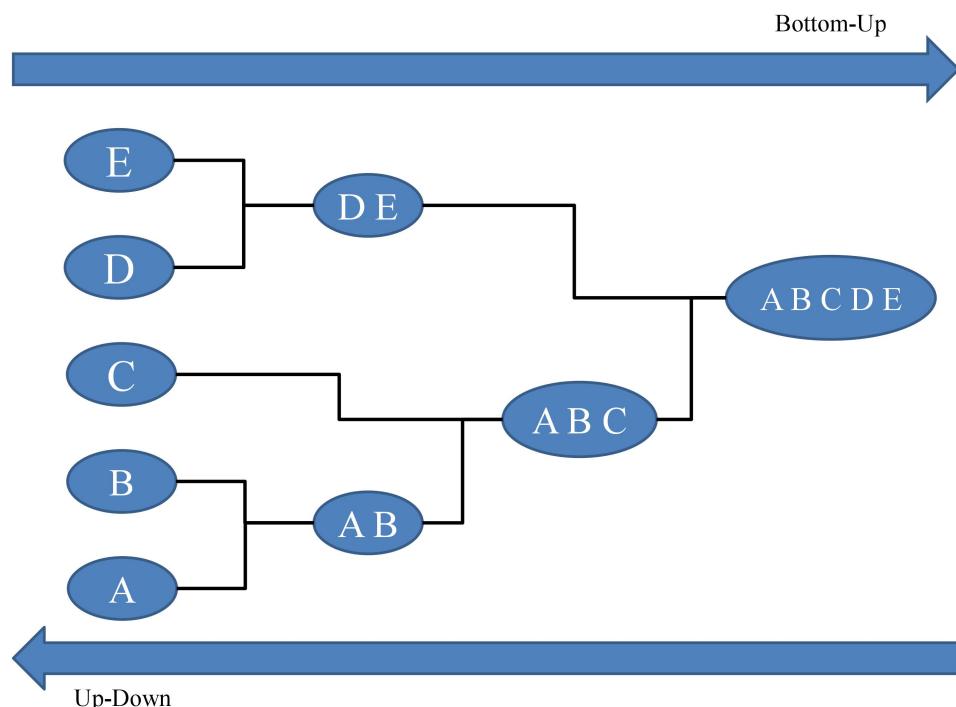


图 13.4: 层次聚类的类别

其代表是 AGNES，一种自底向上聚合的层次聚类方法。该算法的核心即为找到距离最近的两个聚类簇并将其合并。那么簇与簇之间的距离计算方法有多种：

$$\text{最小距离} : d_{min}(C_i, C_j) = \min_{\mathbf{x} \in C_i, \mathbf{z} \in C_j} d(\mathbf{x}, \mathbf{z}), \quad (13.22)$$

$$\text{最大距离} : d_{max}(C_i, C_j) = \max_{\mathbf{x} \in C_i, \mathbf{z} \in C_j} d(\mathbf{x}, \mathbf{z}), \quad (13.23)$$

$$\text{平均距离} : d_{avg}(C_i, C_j) = \frac{1}{|C_i||C_j|} \sum_{\mathbf{x} \in C_i} \cdot \sum_{\mathbf{z} \in C_j} d(\mathbf{x}, \mathbf{z}). \quad (13.24)$$

使用这三种不同的计算方法来衡量簇距离时，AGNES 算法也将相应被称为“单链接”、“全链接”、“均链接”三种形式。对应的算法过程如下：

Algorithm 23: AGNES 算法

Input: 数据集 $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, 聚类簇距离度量函数 f , 聚类数 K

```

1 for  $j = 1, 2, \dots, n$  do
2    $C_j = \mathbf{x}_j$ 
3 end
4 for  $i = 1, 2, \dots, n$  do
5   for  $j = i+1, \dots, n$  do
6      $M(i, j) = d(C_i, C_j)$ 
7      $M(j, i) = M(i, j)$ 
8   end
9 end
10 设置当前聚类簇个数:  $q = n$ 
11 while  $q > K$  do
12   找出距离最近的两个聚类簇  $C_{i^*}$  与  $C_{j^*}$ 
13   合并  $C_{i^*}$  与  $C_{j^*}$ :  $C_{i^*} = C_{i^*} \cup C_{j^*}$ 
14   for  $j = j^* + 1, j^* + 2, \dots, q$  do
15     将簇  $C_j$  重编号为  $C_{j-1}$ 
16   end
17   删除距离矩阵  $M$  的第  $j^*$  行与第  $j^*$  列
18   for  $j = 1, 2, \dots, q-1$  do
19      $M(i^*, j) = d(C_{i^*}, C_j)$ 
20      $M(j, i^*) = M(i^*, j)$ 
21   end
22    $q = q - 1$ 
23 end

```

Output: 簇划分: $\{C_1, C_2, \dots, C_K\}$

层次聚类的优点是，可以通过人工设置不同的参数，从而得到不同粒度的层次聚类结构，可以用于任意形状的聚类。

其缺点则是，大部分算法的时间复杂度大，且聚类的结构在迭代中是不可逆的，即一旦将该簇与另一个簇合并，就不可以再将其拆分从而优化聚类性能。同时它也需要人为地设置停止聚类的条件，如聚类的个数等。

13.5 密度聚类

该方法假设聚类结构可以通过样本分布的紧密程度确定，一般情况下，密度聚类算法从样本密度的角度考察样本之间的可连接性，并基于可连接性不断扩展聚类簇从而获得最终结果。代表算法是 DBSCAN.

该算法有几个比较重要的概念需要介绍一下。假设给定一个数据集 \mathcal{T} :

- **ε -邻域:** 对 $\mathbf{x}_j \in \mathcal{T}$, 其 ε -邻域包含数据集 \mathcal{T} 中与 \mathbf{x}_j 的距离不大于 ε 的样本, 即 $\mathbf{N}_\varepsilon(\mathbf{x}_j) = \{\mathbf{x}_i \in \mathcal{T} | d(\mathbf{x}_i, \mathbf{x}_j) \leq \varepsilon\}$.
- **core-object:** 若 \mathbf{x}_j 的 ε -邻域至少包含 $MinPts$ 个样本, 即 $|\mathbf{N}_\varepsilon(\mathbf{x}_j)| \geq MinPts$, 则 \mathbf{x}_j 为一个核心对象.
- **directly density-reachable:** 若 \mathbf{x}_j 位于 \mathbf{x}_i 的 ε -邻域中, 且 \mathbf{x}_i 是 core-object, 则 \mathbf{x}_j 可由 \mathbf{x}_i 密度直达.
- **density-reachable:** 对 \mathbf{x}_i 与 \mathbf{x}_j , 若存在样本序列 $\mathbf{p}_1, \mathbf{p}_2, \dots, \mathbf{p}_n$, 其中 $\mathbf{p}_1 = \mathbf{x}_i$, $\mathbf{p}_n = \mathbf{x}_j$ 且 \mathbf{p}_{i+1} 由 \mathbf{p}_i 密度直达, 则 \mathbf{x}_j 由 \mathbf{x}_i 密度可达.
- **density-connected:** 对 \mathbf{x}_i 与 \mathbf{x}_j , 若存在 \mathbf{x}_k 使得 \mathbf{x}_i 与 \mathbf{x}_j 均由 \mathbf{x}_k 密度可达, 则称 \mathbf{x}_i 与 \mathbf{x}_j 密度相连.

然后在该算法中, 簇的定义是: 由密度可达关系导出的最大的密度相连样本集合, 即给定邻域参数 $(\varepsilon, Minpts)$, 簇 C 满足下列性质:

- Connectivity: $\mathbf{x}_i \in \mathbf{C}, \mathbf{x}_j \in \mathbf{C} \Rightarrow \mathbf{x}_i, \mathbf{x}_j$ 密度相连.
- Maximality: $\mathbf{x}_i \in \mathbf{C}, \mathbf{x}_j$ 密度可达 $\Rightarrow \mathbf{x}_j \in \mathbf{C}$.

为了从 D 中找到满足条件的聚类簇, 只需找到由 core-object 密度可达的所有样本组成的集合即可. 算法流程如下所示.

DBSCAN 的主要优点有:

1. 可以对任意形状的稠密数据集进行聚类, 相比之下,K-Means 只可以用于凸数据集.
2. 可以在聚类的同时发现异常点并对其不敏感.
3. 相比于 K-Means, 聚类结果并不对初始值敏感.

主要缺点有:



Algorithm 24: DBSCAN 算法

Input: 数据集 $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n\}$, 邻域参数 $(\varepsilon, MinPts)$

- 1 初始化 core-object 集合: $\Omega = \emptyset$ **for** $j = 1, 2, \dots, n$ **do**
- 2 | 确定样本 \mathbf{x}_j 的 ε – 邻域 $N_\varepsilon(\mathbf{x}_j)$ **if** $|N_\varepsilon(\mathbf{x}_j)| \geq MinPts$ **then**
- 3 | | 将样本 \mathbf{x}_j 加入核心对象集合: $\Omega = \Omega \cup \mathbf{x}_j$
- 4 | **end**
- 5 **end**
- 6 初始聚类簇数: $k = 0$ 初始未访问样本集合: $\Gamma = \mathcal{T}$ **while** $\Omega \neq \emptyset$ **do**
- 7 | 记录当前未访问的样本集合: $\Gamma_{old} = \Gamma$
- 8 | 随机选择一个 core-object $\mathbf{o} \in \Omega$, 初始化队列 $Q = < \mathbf{o} >$
- 9 | $\Gamma = \Gamma \setminus \mathbf{o}$
- 10 | **while** $Q \neq \emptyset$ **do**
- 11 | | 取出队列 Q 中的首个样本 \mathbf{q}
- 12 | | **if** $|N_\varepsilon(\mathbf{q})| \geq MinPts$ **then**
- 13 | | | 令 $\Delta = N_\varepsilon(\mathbf{q}) \cap \Gamma$
- 14 | | | 将 Δ 中的样本加入队列 Q
- 15 | | | $\Gamma = \Gamma \setminus \Delta$
- 16 | | **end**
- 17 | | **end**
- 18 | | $k = k + 1$, 生成聚类簇 $C_k = \Gamma_{old} \setminus \Gamma$
- 19 | | $\Omega = \Omega \setminus C_k$
- 20 **end**

Output: 簇划分: $\{C_1, C_2, \dots, C_K\}$

1. 不适用于非稠密分布的数据集中.
2. 样本集较大时, 聚类收敛时间较长.
3. 参数调整比 K-Means 复杂一些, 参数值对最终的聚类性能影响较大.

13.6 Advanced Topics

13.6.1 Birch 聚类

为了解决传统 K-Means 方法中需要多次重复遍历数据集中所有数据点所带来的性能开销问题, Birch 聚类则只需单遍扫描数据集即可完成聚类任务, 在介绍其原理前, 我们需要介绍一些重要概念.

聚类特征 Clustering Feature(CF): 设一个簇中有 N 个 D 维数据点 $\{\mathbf{x}_n | n = 1, 2, \dots, N\}$. 该簇的 $CF = (N, LS, SS)$, 其中 N 为簇中样本数, LS 为各样本向量的线性求和 $\left(\sum_{n=1}^N \mathbf{x}_n^1, \sum_{n=1}^N \mathbf{x}_n^2, \dots, \sum_{n=1}^N \mathbf{x}_n^D\right)^T$, SS 是各样本向量的平方和 $\sum_{n=1}^N \sum_{d=1}^D (\mathbf{x}_n^d)^2 = \sum_{n=1}^N \mathbf{x}_n^T \cdot \mathbf{x}_n$.

各簇的 CF 有可加性, 可用于表示多个簇(不相交)可合并为一个簇. $CF' = CF_1 + CF_2 + \dots + CF_n = (N_1 + N_2 + \dots + N_n, LS_1 + LS_2 + \dots + LS_n,$

$\mathbf{SS}_1 + \mathbf{SS}_2 + \dots + \mathbf{SS}_n$). 该方法可以有效对数据进行压缩, 基于上述性质可以推导出簇的统计量与距离. 假设簇的质心为 \mathbf{x}_0 , 半径为 \mathbf{R} , 直径为 \mathbf{D} .

$$\mathbf{x}_0 = \frac{\sum_{i=1}^N \mathbf{x}_i}{N} = \frac{LS}{N}, \quad (13.25)$$

$$\mathbf{R} = \frac{|\sum_{i=1}^N (\mathbf{x}_i - \mathbf{x}_0)|}{\sqrt{N}} = \frac{N \cdot \mathbf{SS} - \mathbf{LS}^2}{N}, \quad (13.26)$$

$$D = \sqrt{\frac{1}{N(N-1)} \sqrt{\sum_{i=1}^N \sum_{j=1}^N (\mathbf{x}_i - \mathbf{x}_j)^2}} = \sqrt{\frac{2(N \cdot \mathbf{SS} - \mathbf{LS}^2)}{N(N-1)}}. \quad (13.27)$$

R 与 D 反应了簇内相似度, 不同簇之间的距离度量可以使用曼哈顿距离计算, 假设簇一中有 N_1 个样本而簇二中有 N_2 个样本:

$$d(C_i, C_j) = \sqrt{\frac{\sum_{i=1}^{N_1} \sum_{j=N_1+1}^{N_1+N_2} (\mathbf{x}_i - \mathbf{x}_j)^2}{N_1 N_2}} = \sqrt{\frac{\mathbf{SS}_1}{N_1} + \frac{\mathbf{SS}_2}{N_2} - \frac{2LS_1 LS_2}{N_1 N_2}}, \quad (13.28)$$

根据聚类特征 CF, 我们可以构造出一个树状结构来将数据结构化并用于聚类, 该结构名为聚类特征树 **CF – Tree**. 该树的每个节点均由若干个聚类特征 (CF) 组成, 非叶子节点的 CF 则还有指向孩子节点的指针, 每个叶子节点均由一个双向链表连接起来, 同时该树也是等高树. 结构如图:

前面介绍的 CF 线性可加性若放在 CF-Tree 中, 则表示每个父节点中的 CF 节点的三元组 (N, LS, SS) 的值等于其所有下一层的子节点三元组之和. 因此在该结构中, 有几个较重要的参数需要我们注意. 第一个是每个内部节点的最大 CF 数 B , 第二个参数是每个叶子节点最大 CF 值 L , 第三个则是对于叶子节点中任一 CF 对应的簇而言, 其半径必须小于等于 T , 即该簇为一个最大半径为 T 的超球体.

CF – Tree 的生成

先定义 CF-Tree 的结构参数 $B = 2$, $L = 3$ 与 T . 在最开始时, 该树为空, 没有任何样本. 先从数据集 \mathcal{T} 中读入第一个样本, 将其放入一个新的 CF 节点 A 中, 并计算该节点三元组的数值与质心. 现在的树结构如图13.5:

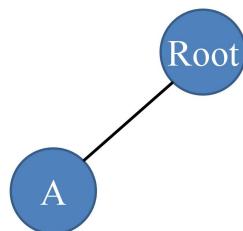


图 13.5: 层次聚类

然后我们继续读入第二个样本, 假设该样本与第一个样本的距离小于 T , 证明

第二个样本也属于 CF 节点 A. 故将第二个样本也放入节点 A 中并更新节点 A 的三元组值与质心. 结构如图13.6所示:

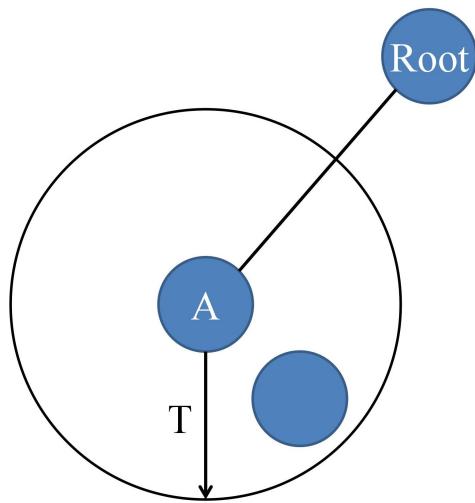


图 13.6: 层次聚类

接着我们读入第三个样本, 假设该样本与节点 A 对应的簇质心距离大于 T , 则表示该样本不可归入节点 A 中. 则创建新的一个 CF 节点 B 来容纳该新样本. 即如图13.7:

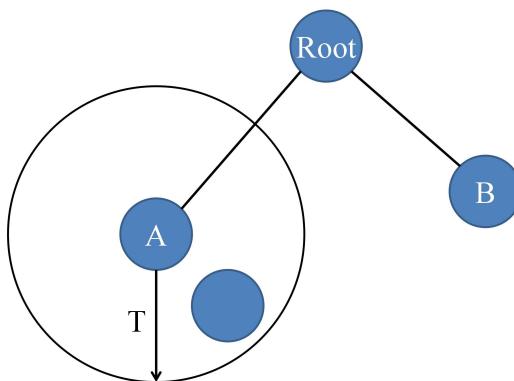


图 13.7: 层次聚类

重复上述过程直至当某个节点需要分裂为止. 假设我们现在的 CF Tree 如图13.8, 叶子节点 LN1 有三个 CF, LN2 和 LN3 各有两个 CF。我们的叶子节点的最大 CF 数 $L=3$ 。此时一个新的样本点来了, 我们发现它离 LN1 节点最近, 因此开始判断它是否在 sc1,sc2,sc3 这 3 个 CF 对应的超球体之内, 但是很不幸, 它不在, 因此它需要建立一个新的 CF, 即 sc8 来容纳它。问题是我们的 $L=3$, 也就是说 LN1 的 CF 个数已经达到最大值了, 不能再创建新的 CF 了, 怎么办? 此时就要将 LN1 的叶子节点分裂。

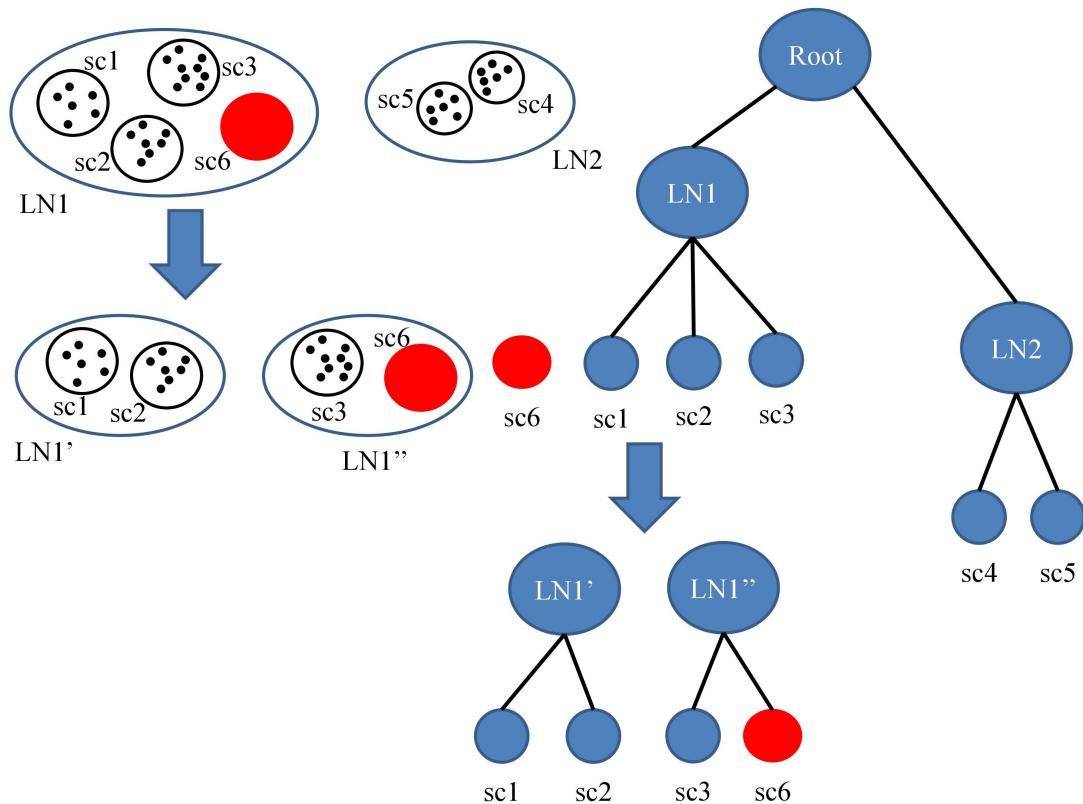


图 13.8: 层次聚类

但是倘若我们直接将分裂而得的两个新节点 $LN1'$ 与 $LN1''$ 直接接于 $Root$ 节点上时, 则会超出 $B = 2$ 的限制, 则需要在 $Root$ 的最左边分支处插入一个新的内部节点来作为 $LN1'$, $LN1''$ 的父节点. 同时为了满足等高性, $Root$ 最右分支也应插入新的内部节点. 最终的结构如图13.9所示:

CF Tree 是 Birch 算法中最重要的结构, 我们使用数据集中的样本构建了 CF Tree 后即基本完成了 Birch 算法, 对应的输出即为若干个 CF 节点, 每个 CF 节点即为一个聚类簇. 但是我们还有一些可选步骤来提升 Birch 算法的性能或将其作为其他聚类算法的预处理过程.

- 在建立的 CF Tree 中去除异常的 CF 节点. 一般这些节点离其他节点很远且样本数少.
- 利用如 K-Means 等聚类算法对所有的 CF 元组进行聚类. 从而得出一棵新的 CF Tree, 其结构比原来的结构更加合理.
- 利用第二点中生成的 CF Tree 中所有 CF 节点的质心作为 K-Means 的初始质心并运行 K-Means. 这样可以提升 K-Means 的性能.

总而言之, 在我们了解了其原理后, 我们可以发现 Birch 算法有以下优点:

- 可以不需要人为设定 K 值. 最终 CF 元组的数目即为 K 值. 若输入 K 值, 则会将所有 CF 元组按元组距离进行合并.
- 适合样本量很大的任务.

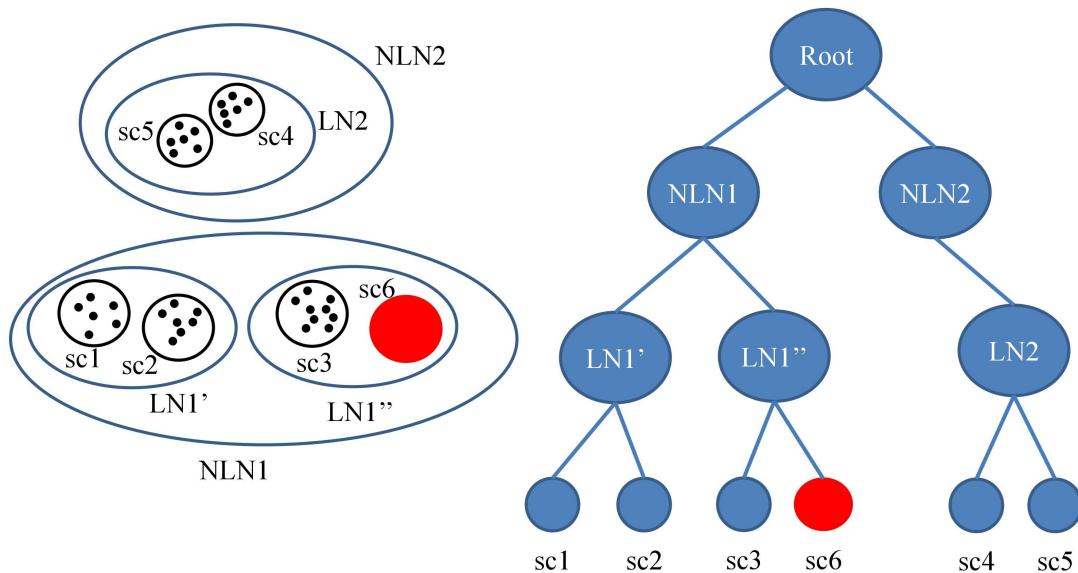


图 13.9: 层次聚类

- 可以自动识别异常数据, 且可用于其他模型的预处理.

当然, Birch 算法也有一些缺点, 需要我们针对实际任务的情况来选择是否要使用它:

- 超参数较多, 其对聚类结果影响较大.
- 对高维特征的数据聚类效果不好.
- 当数据集样本的分布簇不类似于超球体或不为凸时, 聚类效果较差.

参考文献

- [1] D. ARTHUR. k-means++ : The advantages of careful seeding. *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, 2007, pages 1027–1035, 2007.
- [2] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, New York, NY, USA, 2000.
- [3] Allen Gersho and Robert M Gray. *Vector quantization and signal compression*, volume 159. Springer Science & Business Media, 2012.
- [4] John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 99th edition, 1975.
- [5] L. Kaufman and Peter Rousseeuw. Finding groups in data: An introduction to cluster analysis. 1990.
- [6] Shehroz S. Khan and Amir Ahmad. Cluster center initialization algorithm for k-means clustering. *Pattern Recognition Letters*, 25(11):1293 – 1302, 2004.
- [7] Meena Mahajan, Prajakta Nimborkar, and Kasturi Varadarajan. The planar k-means problem is np-hard. In Sandip Das and Ryuhei Uehara, editors, *WALCOM: Algorithms and Computation*, pages 274–285, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
- [8] Joaquín Pérez O, Rodolfo Pazos R, Laura Cruz R, Gerardo Reyes S, Rosy Basave T, and Héctor Fraire H. Improving the efficiency and efficacy of the k-means clustering algorithm through a new convergence condition. In Osvaldo Gervasi and Marina L. Gavrilova, editors, *Computational Science and Its Applications – ICCSA 2007*, pages 674–682, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [9] MAC QUEEN. Some methods for classification and analysis of multivariate observations(methods for classification and analysis of multivariate observations). 1966.

第 14 章 降维

正如前面所提到的，小明想知道自己机器学习这门课最有可能得到什么样的成绩。在线性回归的章节中，小明通过其他科目的分数来预测机器学习的分数；在逻辑回归的章节中，小明通过其他科目及格与否的情况来预测机器学习及格的概率；在推荐系统的章节中，小明通过其他科目的成绩得到了他可能取得高分的未修选修课清单……

无论用哪个模型、何种方法，都离不开小明已经修过的课程的成绩数据。和大多数计算机相关专业的学生一样，大三的小明已经修过了将近五十门课。而不同学生选修课程的情况各异，总共有上百门课程供学生学习。如果直接用所有课程的数据训练模型参数，整个模型将变得十分复杂。更重要的是，很多课程的成绩实际上和机器学习能取得的成绩并没有什么相关性，比如体育成绩、马克思主义原理等等。如果我们能够找出和机器学习真正有关的若干门成绩，不仅能节约训练模型的时间，还将有效提高模型预测的准确度。在统计学和机器学习中，这样的通过获得一组主要变量来减少所考虑的随机变量数量的过程称为降维，它主要分为特征选择和特征提取。

14.1 降维的必要性

在实际的生活生产中，收集到的数据可能包含一些相关性非常高的变量和一些噪音信息，这些冗余的数据不能提供利于区分不同样本的关键特征信息，却大大增加了待分析数据的维度。例如对于一个年级所有学生的成绩数据，如表14.1所示：

每个学生有五门成绩一个总分，但是我们能看出总分从第一到第八的学生在

班级	姓名	语文	数学	英语	理论	实践	总分	名次
1 班	张 1	81	98	98	99	97	473	1
2 班	张 36	82	88	93	99.5	98	461	2
1 班	张 11	74	93	97	99.5	97	461	2
3 班	张 2	83	97	85	100	95	460	4
1 班	张 8	72	94	97	97.5	98.5	459	5
4 班	张 21	78	91	95	95.5	99	459	6
2 班	张 37	74	88	94	98.5	100	455	9
4 班	张 12	72	93	96	98.5	97	457	8

表 14.1：学生成绩

英语、理论、实践这三门取得的成绩差别很小，语文和数学成绩的差别则明显很多。从数学角度来说，各位同学的语文和数学成绩的方差比较大，其他三门方差较小。因此，在这个小数据集中，最重要的两个变量是语文和数学成绩，而其他三门的成绩则是相对来说区分度不大的。那么我们如何找出最重要的两个变量：语文和数学，以此来减小运算成本以及提高模型准确率呢？可以用降维算法。

降维算法是机器学习算法中重要的一部分，常用于减少冗余信息所造成的误差，提高识别精度。除此之外，降维算法还可以通过寻找数据内部的本质结构特征，加速后续计算的速度，解决数据的稀疏问题等。

14.2 特征选择

特征选择方法即试图找到原始变量的子集。好的特征选择能够提升模型的性能，帮助研究者理解数据特点和底层结构，这对改善模型有着重要作用。特征选择方法一般分为三类：过滤法、包装法和嵌入法。我们假设待处理的数据以矩阵 $\mathbf{X} \in \mathbb{R}^{n \times m}$ 表示，每行为一个样本，每列为样本的一个特征，共有 n 个样本， m 个特征。目标变量为 $\mathbf{y} \in \mathbb{R}^n$

14.2.1 过滤法

过滤法即按照发散性和相关性对各个特征进行评分，设置过滤的阈值，接着过滤掉评分低于阈值的几个特征。其中用于评分的标准有很多供选择，例如方差选择法、相关系数法、卡方检验等。

方差选择法主要应用于回归问题中。针对每一列特征 \mathbf{X}_j ，我们求出它的方差 $Var(\mathbf{X}_j) = \sum_{i=1}^n (\mathbf{X}_j^i - \bar{\mathbf{X}}_j)^2$ ，并设定一个阈值 $threshold$ 。由于方差越大说明此特征包含的信息量越大，所以如果 $Var(\mathbf{X}_j) \geq threshold$ ，保留 \mathbf{X}_j ，如果 $Var(\mathbf{X}_j) < threshold$ ，舍弃 \mathbf{X}_j 。这个方法运算起来简单且速度快，可以快速摒弃掉明显冗余的一些特征。但是它对于噪声和离群点十分敏感，而且没有考虑到特征与目标变量的相关性，效果欠佳。

相关系数法也主要应用于回归问题中，它主要通过计算各个特征 \mathbf{X}_j 与目标变量 \mathbf{y} 的 Pearson 相关系数 [10]，计算公式为：

$$\rho_{\mathbf{X}_j, \mathbf{y}} = \frac{cov(\mathbf{X}_j, \mathbf{y})}{\sigma_{\mathbf{X}_j} \sigma_{\mathbf{y}}} = \frac{\sum_{i=1}^n (\mathbf{X}_j^i - \bar{\mathbf{X}}_j) \times (\mathbf{y}_i - \bar{\mathbf{y}})}{\sqrt{\sum_{i=1}^n (\mathbf{X}_j^i - \bar{\mathbf{X}}_j)^2 \times \sum_{i=1}^n (\mathbf{y}_i - \bar{\mathbf{y}})^2}}. \quad (14.1)$$

于是我们选择相关系数最大的若干个特征 \mathbf{X}_j 作为研究的对象。此方法计算速度也很快，而且可以减少特征之间的多重共线性关系，但是它也有一些问题：对于一些明显的噪声数据，它不能够很好的消除影响；对于非线性相关的关系十分不敏感。

卡方检验是以 χ^2 分布为基础的一种常用假设检验方法，一般指的是皮尔森卡方检验 [9]，主要应用于分类问题。它的无效假设 H_0 是：观察频数与期望频数没有差别。基本思想是首先假设 H_0 成立，在此前提下计算出观察值和理论值的偏差程度 χ^2 。我们以一个具体的案例来说明卡方检验。

某研究部门为了了解肝炎与喝酒是否有关，进行一次抽样调查，共调查了 500 个成年人，其中喝酒的 210 人，不喝酒的 290 人。调查结果是喝酒的 210 人中有 42 人患肝炎，168 人未患肝炎；不喝酒的 290 人中有 23 人患病，267 人未患肝炎。我们首先给出无效假设 H_0 ：喝酒与患肝炎之间没有关系。调查数据如表所示：

	患肝炎	未患肝炎	合计
喝酒	42	168	210
不喝酒	23	267	290
合计	65	435	500

我们以 a 、 b 、 c 、 d 分别表示喝酒患病的人数、喝酒未患病的人数、不喝酒患病的人数、不喝酒未患病的人数。那么如果假设 H_0 正确，喝酒患病的概率与不喝酒患病的概率几乎相等，即：

$$\frac{a}{a+b} \approx \frac{c}{c+d}, \quad (14.2)$$

由 (14.2) 可得：

$$ad - bc \approx 0.$$

假设 $n = a + b + c + d$ 为接受调查的总人数， $p(A), p(B), p(C), p(D)$ 分别表示喝酒、患病、不喝酒、不患病的概率，那么喝酒患病的概率为 $p = p(AB) = p(A) \times p(B)$ ，同理可以计算出喝酒不患病、不喝酒患病、不喝酒不患病的概率。接着计算卡方值来刻画实际值和理论值之间的差异：

$$\chi^2 = \frac{n(ad - bc)^2}{(a+b)(a+c)(b+d)(c+d)}, \quad (14.3)$$

往公式 (14.3) 代入数据我们算出卡方值为 15.7。

得到了卡方值，我们离概率结果已经不远了。还需要计算当前调查的自由度 V ，计算公式为：

$$V = (x - 1)(y - 1), \quad (14.4)$$

其中 x 表示数据表中调查数据的行数， y 表示数据表中调查数据的列数。

有了自由度，我们便可以查卡方分布表得到假设的概率结果，卡方分布表的部分数值如表 14.2 所示，由于数据是两行两列的，所以计算出自由度为 1。所以结

F	0.99	0.98	0.90	0.50	0.10	0.05	0.02	0.01	0.001
1	0.000	0.001	0.016	0.045	2.71	3.84	5.41	6.64	10.83
2	0.020	0.040	0.211	1.36	4.61	5.99	7.82	9.21	13.82
3	0.115	0.185	0.584	2.366	6.25	7.82	9.84	11.34	16.27
4	0.297	0.429	1.064	3.357	7.78	9.49	11.67	13.28	18.47
5	0.554	0.752	1.610	4.351	9.24	11.07	13.39	15.09	20.52
6	0.874	1.134	2.204	5.35	10.65	12.69	15.03	16.81	22.46
7	1.239	1.564	2.833	6.35	12.02	14.07	16.62	18.48	24.37

表 14.2: 卡方分布表

果在第一行中找，又因为卡方值为 15.7 大于最右边的 10.83，因此我们可以得到：

$$p(\chi^2 > 10.83) = 0.001$$

于是我们可以拒绝原假设并认为喝酒和患肝炎是相关的，而且拒绝假设的这个决定出错的概率仅为 0.001。因此得出结论：喝酒和患肝炎是相关的。利用卡方检验，我们能有效地找出与分类无关的一些冗余特征，从而降低数据的维度。

14.2.2 包装法

包装法指选择一个目标函数来一步步地筛选特征，最常用的包装法是递归消除特征法。递归特征消除的主要思想是反复的构建模型然后选出最差的几个特征，然后舍弃掉这些特征，接着在剩余的特征上重复这个过程直到所有特征都遍历了一遍。在这个过程中被舍弃的顺序就是特征从劣到优的排序。因此这是一种寻找最优特征子集的贪心算法。

我们以经典的 SVM 模型 [3] 为例来具体说明这个算法。首先，在第一轮训练过程中，选择所有的特征 \mathbf{X}_j 进行训练，可以得到初步分类的超平面 $\mathbf{W}^\top \mathbf{X} + b = 0$ ，其中 $\mathbf{W} \in \mathbb{R}^m$ 。接着选出使得 $|\mathbf{W}_j|^2$ 最小的 j 对应的特征，去除此特征。现在剩下 $m - 1$ 个特征，继续训练和选择的过程，直到剩下的特征数目满足要求。当然也可以一次删除 k ($1 < k < m$) 个特征， k 的值根据具体情况而定。

14.2.3 嵌入法

嵌入法选择特征是将特征选择过程与训练过程融为一体，两者在同一优化过程中完成。最常见的嵌入式特征选择方法是正则化方法，即将额外的因子加入到损失函数中，对模型的复杂度进行一定的约束，这在防止模型的过拟合中有着广泛的运用。主要有岭回归（Ridge Regression）和套索回归（Lasso Regression）两种方法。

首先，对于普通的最小二乘线性回归，损失函数为：

$$\mathcal{L} = \sum_{i=1}^n (\mathbf{y}_i - b - \sum_{j=1}^p \mathbf{w}_j \mathbf{x}_j^i)。 \quad (14.5)$$

岭回归是一种专用于共线性数据分析的有偏估计回归方法，实质上是一种改良的最小二乘估计法，通过放弃最小二乘法的无偏性，以损失部分信息、降低精度为代价获得回归系数更为符合实际、更可靠的回归方法，对病态数据的拟合更强。对于线性回归中的最优解 $\mathbf{W} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}$ ，当 $\mathbf{X}^\top \mathbf{X}$ 不是列满秩时，或者某些列之间的线性相关性比较大时， $(\mathbf{X}^\top \mathbf{X})^{-1}$ 的计算误差会很大，传统的最小二乘法缺乏稳定性和可靠性。因此我们为损失函数加上一个正则化项，损失函数变成了：

$$\mathcal{L} = \sum_{i=1}^n (\mathbf{y}_i - b - \sum_{j=1}^p \mathbf{w}_j \mathbf{x}_j^i) + \lambda \sum_{j=1}^p \mathbf{w}_j^2, \quad (14.6)$$

其中 λ 是一个非负的调节参数，用于调节惩罚的作用强弱。

套索回归方法的特点是在拟合广义线性模型的同时进行变量筛选和复杂度调整。因此不论目标变量是连续的还是二元或者多元离散的，都可以用套索回归建模然后预测。这里的变量筛选是指不把所有变量都放入模型中进行拟合，而是有选择地把变量放入模型从而得到更好的性能参数。复杂度调整是指通过一系列参数控制模型的复杂度，从而避免过度拟合。对于线性模型来说，复杂度与模型的变量数有直接的关系，变量数越多，模型复杂度越高。一般来说，变量数大于数据点数量很多，或者某一个离散变量有太多独特值时，都有可能过拟合。套索回归的正则项就是把岭回归中的二次项改成了 L1 范数，具体的损失函数为：

$$\mathcal{L} = \sum_{i=1}^n (\mathbf{y}_i - b - \sum_{j=1}^p \mathbf{w}_j \mathbf{x}_j^i) + \lambda \sum_{j=1}^p |\mathbf{w}_j|。 \quad (14.7)$$

由于一次项的求导可以抹去变量本身，因此套索的回归系数可以为 0，这样可以起到真正的特征筛选效果。更详细的介绍请参考线性回归章节中的相关内容。

14.3 特征提取

特征提取可以看作特征选择方法的一般化。特征选择假设在原始数据中特征数目繁多，但只有部分特征真正起作用；特征提取则是通过线性或非线性的方式将原来的数据从高维空间变换到一个新的低维空间。最常见的特征提取方法有主成分分析法、线性判别分析和局部线性嵌入等，我们先以主成分分析法为例来具体分析。

14.3.1 矩阵与线性变换

在介绍 PCA 方法之前，有必要先解释一下线性变换和矩阵运算之间的关系。先考虑最经典的二维笛卡尔直角坐标系，那么对于坐标系中任意一个向量 $(x, y)^\top$ ，我们都可以写成：

$$x(1, 0)^\top + y(0, 1)^\top, \quad (14.8)$$

这里的 $(1, 0)$ 和 $(0, 1)$ 称为单位基向量， x 和 y 称为这个点在基向量上的投影（矢量）。因此，我们只需确定一组基向量并找出每个向量在基向量方向上的投影，便可以准确描述每个向量。在二维笛卡尔坐标系中，通常省略确定基向量的环节，默认以 $(1, 0)$ 和 $(0, 1)$ 为基向量。

但是如果我们要用另外两个向量作为基向量呢？当然也是可行的。例如用 $(1, 1)$ 和 $(-1, 1)$ 作为一组新的基，为了让其单位化以方便投影的计算，我们得到 $\left(\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ 和 $\left(-\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right)$ 。现在想获得原坐标系中 (x, y) 的坐标，该怎么办呢？很简单，只需要计算出它在两个基方向上的投影。通过计算它和两个单位基的内积，得到新的坐标为 $\left(\frac{x+y}{\sqrt{2}}, \frac{y-x}{\sqrt{2}}\right)$ 。回想一下上面的例子，其实可以用矩阵的乘法更简洁地表示这个坐标变换：

$$\begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ -\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{x+y}{\sqrt{2}} \\ \frac{y-x}{\sqrt{2}} \end{pmatrix}. \quad (14.9)$$

因此由基向量按行组成的矩阵乘以原向量，结果刚好为该向量在新的基方向上的坐标。一般地，假设有 m 个 n 维向量，我们想将它们变换到 s 个 n 维向量表示的新线性空间中。只需将 s 个基按行组成一个矩阵，再将初始向量按列组成另一个矩阵，它们的乘积即是变换后的结果：

$$\begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_s \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \cdots & \mathbf{x}_m \end{pmatrix} = \begin{pmatrix} p_1 \mathbf{x}_1 & p_1 \mathbf{x}_2 & \cdots & p_1 \mathbf{x}_m \\ p_2 \mathbf{x}_1 & p_2 \mathbf{x}_2 & \cdots & p_2 \mathbf{x}_m \\ \vdots & \vdots & \ddots & \vdots \\ p_s \mathbf{x}_1 & p_s \mathbf{x}_2 & \cdots & p_s \mathbf{x}_m \end{pmatrix}. \quad (14.10)$$

在 (14.10) 中， p_i 是 n 维的行向量，表示第 i 个基向量； \mathbf{x}_j 是 n 维的列向量，表示第 j 个原始向量； $p_i \mathbf{x}_j$ 表示 p_i 和 \mathbf{x}_j 内积运算的结果。如果这里的 $s < n$ ，我们实际上是将一组 n 维数据变换到了 s 维空间中，也就是实现了一次降维变换。

因此如果从线性变换的角度来看矩阵乘法，两个矩阵相乘其实相当于将右边矩阵中的每个列向量变换到以左边矩阵中每个行向量为基所表示的空间。甚至可以说，一个矩阵即表示一种线性变换。

14.3.2 主成分分析法 (Principal Component Analysis)

主成分分析法由 Karl Pearson 在 1901 年发明，是一种线性降维方法，高维空间（维数为 D）的某个点 $\mathbf{x}_i = (x_1, x_2, \dots, x_n)^\top$ 通过矩阵运算变换到低维空间（维数为 d, $d < D$ ）。例如图 14.1 中的三维向量 \mathbf{x} ，经过变换转换成二维向量 \mathbf{x}

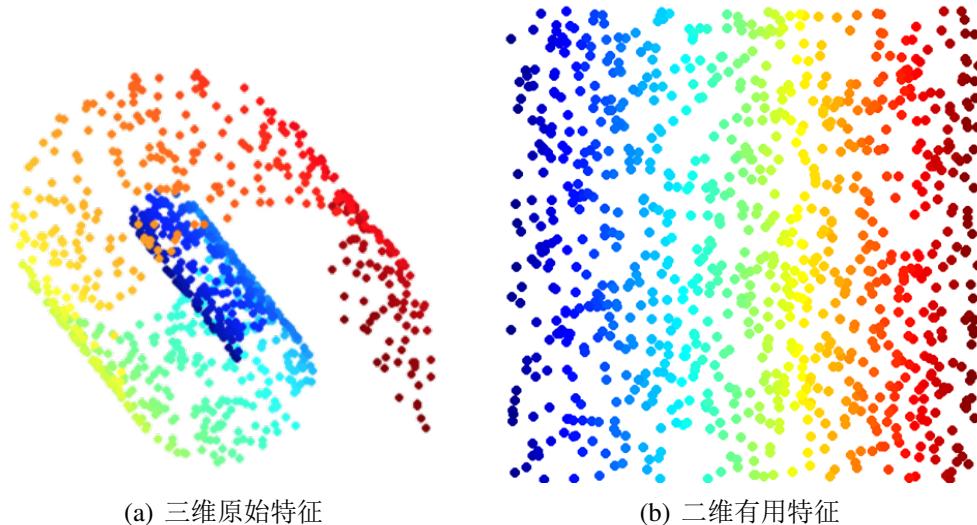


图 14.1: 降维示例

主成分分析法的核心目标是找到一种线性变换，将高维数据变换到一个低维空间。由上节的讨论我们可知，这等价于找到一个合适的矩阵 \mathbf{W} 对每个样本数据 $\mathbf{x} \in \mathbb{X}$ 进行线性变换 $\mathbf{W}\mathbf{x} = \mathbf{z}$ ，最终将高维数据 $\mathbf{A} \in \mathbb{R}^{n \times d}$ 转换成低维数据 $\mathbf{Z} \in \mathbb{R}^{n \times s}$ ($0 < s < d$)。但是降维必然面临着信息损失的问题，那么如何选择基向量的方向才能尽可能多地保留原始信息呢？换句话说，如何选择基向量才能使不同数据之间更容易区分呢？由于数据向量由在各个基向量方向上的投影值组成，直观来说，不同数据的投影值越发散，我们越容易区分不同的数据向量。

在数学中，我们用方差来描述数据的分散程度。因此如果想把数据降成一维，我们只需找到使得投影值方差最大的方向，于是目标函数为：

$$\begin{aligned} \mathbf{w}^1 &= \arg \max_{\mathbf{w}} \text{Var}(\mathbf{z}_1), \\ \text{s.t. } &\|\mathbf{w}^1\|_2 = 1 \end{aligned} \tag{14.11}$$

将 $\mathbf{w}^1 \cdot \mathbf{x} = \mathbf{z}_1$ 代入 (14.11) 式我们可以得到：

$$\begin{aligned} Var(\mathbf{z}_1) &= \frac{1}{N} \sum_x (\mathbf{w}^1 \cdot \mathbf{x} - \mathbf{w}^1 \cdot \bar{x})^2 \\ &= \frac{1}{N} \sum (\mathbf{w}^1 \cdot (\mathbf{x} - \bar{\mathbf{x}}))^2 \\ &= \frac{1}{N} \sum (\mathbf{w}^1)^T (\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{w}^1 \\ &= (\mathbf{w}^1)^T \frac{1}{N} \sum (\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T \mathbf{w}^1 \end{aligned} \quad (14.12)$$

其中 $\frac{1}{N} \sum (\mathbf{x} - \bar{\mathbf{x}})(\mathbf{x} - \bar{\mathbf{x}})^T$ 刚好是数据矩阵 \mathbf{A} 的协方差矩阵 $Cov(\mathbf{A})$ ，因此可以将原函数写成 $Var(\mathbf{z}_1) = (\mathbf{w}^1)^T Cov(\mathbf{A}) \mathbf{w}^1$ ，记 $\mathbf{S} = Cov(\mathbf{A})$ 。于是我们的目标变成寻找矩阵 \mathbf{w}^1 使得 $(\mathbf{w}^1)^T \mathbf{S} \mathbf{w}^1$ 最大，其中 $\|\mathbf{w}^1\|_2 = (\mathbf{w}^1)^T \mathbf{w}^1 = 1$ 。

对于此问题，我们可以采用拉格朗日乘数法 [2] 来解决，于是有：

$$L(\mathbf{w}^1) = (\mathbf{w}^1)^T \mathbf{S} \mathbf{w}^1 - \alpha((\mathbf{w}^1)^T \mathbf{w}^1 - 1). \quad (14.13)$$

令 $\frac{\partial L(\mathbf{w}^1)}{\partial \mathbf{w}_1^1} = 0$ 、 $\frac{\partial L(\mathbf{w}^1)}{\partial \mathbf{w}_2^1} = 0$ 、 \dots 、 $\frac{\partial L(\mathbf{w}^1)}{\partial \mathbf{w}_m^1} = 0$ ，可以得到 $\mathbf{S} \mathbf{w}^1 = \alpha \mathbf{w}^1$ 。由线性代数 [7] 的相关知识， α 恰好是 \mathbf{S} 的一个特征值， \mathbf{w}^1 则是协方差矩阵 \mathbf{S} 的一个特征向量，对应特征值 α 。

这是把数据降到一维的情况，如果我们想获得更高维的降维数据该怎么办？假如还是按照之前的方法操作，单纯选择方差最大的方向，得到的结果将和第一个方向几乎重合，而这显然不是我们想要的。我们想要的是在这个新方向上的投影值尽可能发散的同时，它和第一个方向不存在线性相关性。这样才能避免两者表示重复的信息。在数学中，两个向量的相关性可以用协方差来描述，当协方差为 0 时，表示两者完全独立。

经过上述的讨论，如果我们想将一组 d 维数据向量降为一组 s 维向量 ($1 < s < d$)，需要新向量中每个维度内的方差尽可能大，并且各个基向量之间两两协方差为 0。因此，我们需要选择 s 个标准正交基（两两之间的投影都为 0），并使得这 s 个维度内的方差是最大的。

再回到求解一维情况时向量 \mathbf{x} 的协方差矩阵 \mathbf{S} ，根据线性代数的知识，它的第 (i, j) ($i \neq j$) 项表示 \mathbf{A} 中第 i 个维度和第 j 维度之间的协方差，第 (i, i) 项表示第 i 个维度内的方差。因此我们要找的 \mathbf{W} 是能让变换后数据矩阵 \mathbf{Z} 的协方差矩阵 $Cov(\mathbf{Z})$ 对角化的矩阵，即让它除对角线以外的数据均为 0，并且对角线上的元素按从大到小的顺序从上到下排列。

根据协方差矩阵本身的性质很容易知道它是一个实对称矩阵，而实对称矩阵的性质包括：不同特征值对应的特征向量必然正交；所有特征值都是实数； s 个特征值对应的线性无关的特征向量恰好有 s 个。因此对于 d 行 d 列的实对称矩阵

\mathbf{S} , 我们一定可以找到 d 个单位正交特征向量, 将其按列组成矩阵为:

$$\mathbf{W} = (\mathbf{w}_1 \quad \mathbf{w}_2 \quad \cdots \quad \mathbf{w}_d)。 \quad (14.14)$$

根据矩阵对角化相关知识, 我们得到:

$$\mathbf{W}^T \mathbf{S} \mathbf{W} = \Lambda = \begin{pmatrix} \lambda_1 & & & \\ & \lambda_2 & & \\ & & \ddots & \\ & & & \lambda_d \end{pmatrix}, \quad (14.15)$$

其中 Λ 为对角矩阵, 对角元素为各特征向量对应的特征值。到这里离我们的目标就不远了, 只需按从大到小的顺序选取 s 个特征值, 将它们对应的特征向量按顺序从上到下排列, 就构成了我们所需要的转换矩阵 $\mathbf{W} \in \mathbb{R}^{s \times d}$ 最后我们用 \mathbf{W} 中每个特征向量 $\mathbf{w}_i \in \mathbb{R}^{d \times 1}$ 与每个样本 $\mathbf{x} \in \mathbb{R}^{1 \times d}$ 相乘, 便得到了新维度空间下的数据 \mathbf{x} :

$$\mathbf{x} = \begin{bmatrix} \mathbf{x}\mathbf{w}_1 \\ \mathbf{x}\mathbf{w}_2 \\ \vdots \\ \mathbf{x}\mathbf{w}_s \end{bmatrix} \in \mathbb{R}^s, \quad (14.16)$$

其中的单位向量 $\mathbf{w}_1, \dots, \mathbf{w}_s$ 也就叫做数据集的前 s 个主成分(principal components)

14.3.3 算法实例

为了准确预测学生机器学习的成绩, 我们选取五名学生的 Java 语言程序设计和形势与政策两门成绩, 构成的初始数据矩阵为:

$$\mathbf{A} = \begin{pmatrix} 82 & 90 \\ 45 & 92 \\ 62 & 93 \\ 77 & 86 \\ 99 & 89 \end{pmatrix}, \quad (14.17)$$

其中 \mathbf{A} 中每行代表一个学生的成绩数据, 第一列表示学生的 Java 语言程序设计成绩, 第二列表示学生的形势与政策成绩。我们现在用上文中的 PCA 方法将这组二维数据降为一维。

为了计算的方便, 对于原始数据首先进行去中心化操作, 即在每个数据上减

去所在列的均值。于是我们得到：

$$\mathbf{A} = \begin{pmatrix} 9 & 0 \\ -28 & 2 \\ -11 & 3 \\ 4 & -4 \\ 26 & -1 \end{pmatrix}, \quad (14.18)$$

由于现在每个维度的均值都为 0，因此数据 \mathbf{A} 的协方差矩阵为：

$$\begin{aligned} \mathbf{S} = \frac{1}{N} \mathbf{A}^T \mathbf{A} &= \frac{1}{5} \begin{pmatrix} 9 & -28 & -11 & 4 & 26 \\ 0 & 2 & 3 & -4 & -1 \end{pmatrix} \begin{pmatrix} 9 & 0 \\ -28 & 2 \\ -11 & 3 \\ 4 & -4 \\ 26 & -1 \end{pmatrix} \\ &= \begin{pmatrix} 335.6 & -26.2 \\ -26.2 & 6 \end{pmatrix}. \end{aligned} \quad (14.19)$$

接下来我们需要求出 \mathbf{S} 的特征值和特征向量（具体过程可参考线性代数），得到特征值为：

$$\lambda_1 \approx 337.7, \lambda_2 \approx 3.9, \quad (14.20)$$

分别对应的特征向量为：

$$\mathbf{w}_1 = \begin{pmatrix} -262 \\ 21 \end{pmatrix}, \mathbf{w}_2 = \begin{pmatrix} 21 \\ 262 \end{pmatrix}. \quad (14.21)$$

进行标准化（模长为 1）处理后的结果为：

$$\mathbf{w}_1 = \begin{pmatrix} -0.997 \\ 0.080 \end{pmatrix}, \mathbf{w}_2 = \begin{pmatrix} 0.080 \\ 0.997 \end{pmatrix}, \quad (14.22)$$

由于我们希望将原始的二维数据降为一维，所以选取较大的特征值 λ_1 对应的特征向量 \mathbf{w}_1 形成转换矩阵：

$$\mathbf{W} = \begin{pmatrix} -0.997 \\ 0.080 \end{pmatrix}, \quad (14.23)$$

最后我们用原始数据 \mathbf{A} 和转换矩阵 \mathbf{W} 相乘即可得到降维后的数据：

$$\mathbf{Z} = \mathbf{AW} = \begin{pmatrix} 9 & 0 \\ -28 & 2 \\ -11 & 3 \\ 4 & -4 \\ 26 & -1 \end{pmatrix} \begin{pmatrix} -0.997 \\ 0.080 \end{pmatrix} = \begin{pmatrix} -8.973 \\ 28.076 \\ 11.207 \\ -4.308 \\ -26.002 \end{pmatrix}。 \quad (14.24)$$

从 \mathbf{W} 的具体值我们可以看出，Java 语言程序设计的成绩数据在预测中的重要性远大于形势与政策的成绩数据，学生们的机器学习的成绩可以说和形势与政策的成绩几乎无关。

14.3.4 延伸话题

14.3.4.1 线性判别分析（Linear Discriminant Analysis）

在上文讨论的 PCA 中，降维过程没有把类别标签考虑进来，因此属于无监督的降维方法。这一小节我们介绍一种有监督的降维方法——线性判别分析。

线性判别分析是对费舍尔线性判别方法 [5] 的一种归纳，首先来看相对较为简单的二分类问题。假设有 N 个 d 维样本，其中 N_1 个属于类别 ω_1 ， N_2 个属于类别 ω_2 。为了更便于理解，我们先尝试将数据维度降到只有一维。也就是说，需要找到一个 \mathbb{R}^d 空间的向量 \mathbf{W} ，使得每个样本 \mathbf{x} 与其相乘之后得到降维后的一维数据 \mathbf{z} ：

$$\mathbf{z} = \mathbf{W}^\top \mathbf{x}。 \quad (14.25)$$

首先我们希望降维之后的这两类数据之间的区别应越明显越好，这个区别可以用两类样本中心点之间的距离来衡量。因此我们计算出第 i 类样本降维前后的中心点 μ_i 和 $\tilde{\mu}_i$ ：

$$\mu_i = \frac{1}{N_i} \sum_{\mathbf{x} \in \omega_i} \mathbf{x}, \quad (14.26)$$

$$\tilde{\mu}_i = \frac{1}{N_i} \sum_{\mathbf{z} \in \omega_i} \mathbf{z} = \frac{1}{N_i} \sum_{\mathbf{x} \in \omega_i} \mathbf{W}^\top \mathbf{x} = \mathbf{W}^\top \mu_i。 \quad (14.27)$$

接着我们用两类中心点之间的欧式距离来衡量两类样本的距离：

$$D(\mathbf{W}) = \|\tilde{\mu}_1 - \tilde{\mu}_2\|^2 = \|\mathbf{W}^\top (\mu_1 - \mu_2)\|^2。 \quad (14.28)$$

我们希望这个距离越大越好，但是仅仅考虑两类之间的距离是不够的。如果两类之间距离很大，但是每一类内部的样本分布很分散，降维之后两类依然很有可能出现重叠交叉的情况。因此我们同时也希望在每个类别内样本之间的方差尽可能

小，专业上通常使用散列值（Scatter）来衡量类内样本的分散程度。在数值上，散列值等于方差乘以样本数量。

于是我们求出降维之后第 i 类的散列值：

$$\tilde{s}_i^2 = \sum_{\mathbf{z} \in \omega_i} (\mathbf{z} - \tilde{\mu}_i)^2, \quad (14.29)$$

前文说过，我们希望每个类别内的样本越集中越好，也就是散列值越小越好。综合最大化不同类别之间距离的目标，我们可以将两者结合起来进行优化：

$$J(\mathbf{W}) = \frac{\|\tilde{\mu}_1 - \tilde{\mu}_2\|^2}{\tilde{s}_1^2 + \tilde{s}_2^2}. \quad (14.30)$$

现在只需寻找使得 $J(\mathbf{W})$ 最大的 \mathbf{W} 即可。对于 (14.29)，进行进一步展开：

$$\begin{aligned} \tilde{s}_i^2 &= \sum_{\mathbf{z} \in \omega_i} (\mathbf{z} - \tilde{\mu}_i)^2 \\ &= \sum_{\mathbf{x} \in \omega_i} (\mathbf{W}^\top (\mathbf{x} - \mu_i))^2 \\ &= \sum_{\mathbf{x} \in \omega_i} \mathbf{W}^\top (\mathbf{x} - \mu_i) (\mathbf{x} - \mu_i)^\top \mathbf{W} \end{aligned}, \quad (14.31)$$

令 $\mathbf{S}_i = \sum_{\mathbf{x} \in \omega_i} (\mathbf{x} - \mu_i) (\mathbf{x} - \mu_i)^\top$ ，则 $\tilde{s}_i^2 = \mathbf{W}^\top \mathbf{S}_i \mathbf{W}$ 。又令 $\mathbf{S}_w = \mathbf{S}_1 + \mathbf{S}_2$ ，则目标函数的分母为：

$$\tilde{s}_1^2 + \tilde{s}_2^2 = \mathbf{W}^\top \mathbf{S}_1 \mathbf{W} + \mathbf{W}^\top \mathbf{S}_2 \mathbf{W} = \mathbf{W}^\top (\mathbf{S}_1 + \mathbf{S}_2) \mathbf{W} = \mathbf{W}^\top \mathbf{S}_w \mathbf{W}, \quad (14.32)$$

其中 \mathbf{S}_w 称为类内离散度矩阵（Within-Class Scatter Matrix）[8]。

对于目标函数的分子：

$$\begin{aligned} \|\tilde{\mu}_1 - \tilde{\mu}_2\|^2 &= \|\mathbf{W}^\top (\mu_1 - \mu_2)\|^2 \\ &= \mathbf{W}^\top (\mu_1 - \mu_2) (\mu_1 - \mu_2)^\top \mathbf{W} \end{aligned}, \quad (14.33)$$

令 $\mathbf{S}_b = (\mu_1 - \mu_2) (\mu_1 - \mu_2)^\top$ ，则 $\|\tilde{\mu}_1 - \tilde{\mu}_2\|^2 = \mathbf{W}^\top \mathbf{S}_b \mathbf{W}$ 。其中 \mathbf{S}_b 称为类间离散度矩阵（Between-Class Scatter Matrix）[8]。

因此目标函数变成了：

$$J(\mathbf{W}) = \frac{\mathbf{W}^\top \mathbf{S}_b \mathbf{W}}{\mathbf{W}^\top \mathbf{S}_w \mathbf{W}}. \quad (14.34)$$

由于此处的 \mathbf{W} 可以乘上任何常数而不改变目标函数值，我们有必要对分母进行归一化：

$$\|\mathbf{W}^\top \mathbf{S}_w \mathbf{W}\| = 1. \quad (14.35)$$

现在问题等价于在此约束下求分子的最大值，可以运用拉格朗日乘数法：

$$L(\mathbf{W}) = \mathbf{W}^\top \mathbf{S}_b \mathbf{W} - \lambda (\mathbf{W}^\top \mathbf{S}_w \mathbf{W} - 1), \quad (14.36)$$

对 \mathbf{W} 求导并令其为零：

$$\frac{\partial L}{\partial \mathbf{W}} = 2\mathbf{S}_b \mathbf{W} - 2\lambda \mathbf{S}_w \mathbf{W} = 0, \quad (14.37)$$

我们得到：

$$\mathbf{S}_b \mathbf{W} = \lambda \mathbf{S}_w \mathbf{W}, \quad (14.38)$$

因此：

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{W} = \lambda \mathbf{W}. \quad (14.39)$$

但是在实际应用中，经常出现训练样本不够的现象，例如人脸识别等图像识别问题。导致这里的 \mathbf{S}_w 经常是不可逆的，遇到这种情况我们该怎么办呢？一种方法是我们可以先采用 PCA 进行降维之后再进行 LDA 降维，另一种方法则是对 \mathbf{S}_w 进行奇异值分解 [5][6]：

$$\mathbf{S}_w = U \Sigma V^\top, \quad (14.40)$$

这样得到的 \mathbf{S}_w^{-1} 为：

$$\mathbf{S}_w^{-1} = V \Sigma^{-1} U^\top. \quad (14.41)$$

我们再观察一下 $\mathbf{S}_b \mathbf{W} = (\mu_1 - \mu_2)(\mu_1 - \mu_2)^\top \mathbf{W}$ ，由于 $(\mu_1 - \mu_2)^\top \in \mathbb{R}^{1 \times d}$, $\mathbf{W} \in \mathbb{R}^{d \times 1}$ ，因此对任意一组样本这两者的乘积是常数，所以可以表示成：

$$\mathbf{S}_b \mathbf{W} = \lambda_w (\mu_1 - \mu_2), \quad (14.42)$$

λ_w 为和 \mathbf{W} 有关的常数。将 (14.42) 代入 (14.39) 我们可以得到：

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{W} = \lambda_w \mathbf{S}_w^{-1} (\mu_1 - \mu_2) = \lambda \mathbf{W}. \quad (14.43)$$

由于对 \mathbf{W} 乘上任何常数不影响结果，我们可以约去式子中的常数 λ 和 λ_w ，最后得到：

$$\mathbf{W} = \mathbf{S}_w^{-1} (\mu_1 - \mu_2). \quad (14.44)$$

多分类问题的处理方式和二分类是类似的，假设总共有 C 个类别，我们计划将原始的 d 维数据降为 k 维，那么所需的转换矩阵 $\mathbf{W} \in \mathbb{R}^{d \times k}$ ，通过 $\mathbf{z} = \mathbf{W}^\top \mathbf{x}$ 得到最终的每个样本数据。

多分类问题中的第 i 类中心点 μ_i 和类内离散度矩阵 \mathbf{S}_{wi} 的计算公式和二分类

中一样，总共的 \mathbf{S}_w 为：

$$\mathbf{S}_w = \sum_{i=1}^C \mathbf{S}_{wi}。 \quad (14.45)$$

但是类间离散度矩阵就不能简单的采用两类中心点之间的距离了，这里用每类中心点相对于样本总体中心点的散列情况来衡量类间的离散度。但是不同类的样本数目分布不一定均匀，对于样本数较多的类别，我们应该着重考虑，这样才能更好地刻画出样本总体在类别之间的离散程度，因此我们加入每个类别的样本数作为该类的权重：

$$\mathbf{S}_b = \sum_{i=1}^C N_i (\mu_i - \mu)(\mu_i - \mu)^\top, \quad (14.46)$$

其中 μ 表示样本总体的中心点，计算公式为：

$$\mu = \frac{1}{N} \sum_{\forall \mathbf{x}} \mathbf{x} = \frac{1}{N} \sum_{\mathbf{x} \in \omega_i} N_i \mu_i。 \quad (14.47)$$

对于降维后的数据，每个类的中心点 $\tilde{\mu}_i$ 以及总体中心点 $\tilde{\mu}$ 为：

$$\tilde{\mu}_i = \frac{1}{N_i} \sum_{\mathbf{z} \in \omega_i} \mathbf{z}, \quad (14.48)$$

$$\tilde{\mu} = \frac{1}{N} \sum_{\forall \mathbf{z}} \mathbf{z}。 \quad (14.49)$$

于是我们计算出类内离散度矩阵的总和 \mathbf{S}_w 以及类间离散度矩阵总和 \mathbf{S}_b ：

$$\mathbf{S}_w = \sum_{i=1}^C \sum_{\mathbf{z} \in \omega_i} (\mathbf{z} - \tilde{\mu}_i)(\mathbf{z} - \tilde{\mu}_i)^\top, \quad (14.50)$$

$$\mathbf{S}_b = \sum_{i=1}^C N_i (\tilde{\mu}_i - \tilde{\mu})(\tilde{\mu}_i - \tilde{\mu})^\top。 \quad (14.51)$$

和二分类中同理，用于刻画不同类间的距离函数 $D(\mathbf{W})$ 和刻画类内离散程度的离散度函数 $S(\mathbf{W})$ 为：

$$D(\mathbf{W}) = \mathbf{W}^\top \mathbf{S}_b \mathbf{W}, \quad (14.52)$$

$$S(\mathbf{W}) = \mathbf{W}^\top \mathbf{S}_w \mathbf{W}。 \quad (14.53)$$

因此我们得到最终的目标函数：

$$J(\mathbf{W}) = \frac{D(\mathbf{W})}{S(\mathbf{W})} = \frac{\mathbf{W}^\top \mathbf{S}_b \mathbf{W}}{\mathbf{W}^\top \mathbf{S}_w \mathbf{W}}。 \quad (14.54)$$

根据前文提到的拉格朗日乘数法加上分母归一化，我们可以得到相似的结论：

$$\mathbf{S}_w^{-1} \mathbf{S}_b \mathbf{W} = \lambda \mathbf{W}. \quad (14.55)$$

对于 \mathbf{S}_w 不可逆的问题这里就不再赘述。根据线性代数的知识，行列式的值是矩阵对应矩阵特征值的积，而目标函数的分子分母都表示的是散列矩阵，特征值表示主要方向上的发散程度（详见主成分分析章节）。因此我们可以对分子分母都取行列式将目标函数中的矩阵变为实数：

$$J(\mathbf{W}) = \frac{|\mathbf{W}^\top \mathbf{S}_b \mathbf{W}|}{|\mathbf{W}^\top \mathbf{S}_w \mathbf{W}|}, \quad (14.56)$$

再将 (14.55) 代入 (14.56) 中我们得到：

$$\mathbf{W} = \operatorname{argmax} |\lambda|. \quad (14.57)$$

因此我们只需选取矩阵 $\mathbf{S}_w^{-1} \mathbf{S}_b$ 中绝对值较大的 k 个特征值所对应的特征向量按列组成转换矩阵 \mathbf{W} 即可。此处应注意由于 \mathbf{S}_{bi} 的秩为 1，因此 \mathbf{S}_b 的秩至多是 C ，又因为最后一个 $\mu_i - \mu$ 可以由前 $C - 1$ 个的线性组合得到 ($\sum_{i=1}^C (\mu_i - \mu) = 0$)，所以 \mathbf{S}_b 的秩至多为 $C - 1$ ，我们降维后的数据的维度 $k \leq C - 1$ 。

14.3.4.2 局部线性嵌入 (Locally Linear Embedding)

局部线性嵌入 (Locally Linear Embedding) [11] 也是非常重要的降维方法。和前文中 PCA, LDA 等关注样本方差的降维方法相比，LLE 关注降维时保持样本局部的线性特征，由于 LLE 在降维时保持了样本的局部特征，它广泛的用于图像识别，高维数据可视化等领域。

LLE 属于流形学习 (Manifold Learning) 的一种。那么什么是流形学习呢？流形学习的观点认为，我们所能观察到的数据实际上是由一个低维流形映射到高维空间上的。由于数据内部特征的限制，一些高维中的数据会产生维度上的冗余。基于流行的降维算法就是将流形从高维到低维的降维过程，在降维的过程中我们希望流形在高维的一些特征可以得到保留。一个形象的流形降维过程如图 14.2。我们有一块卷起来的布，希望将其展开到一个二维平面，同时希望展开后的布能够在局部保持布结构的特征，就像两个人将其拉开一样。

LLE 首先假设数据在较小范围的局部内是线性的，也就是说，某一个数据可以由它邻域中的几个样本来线性表示。比如我们有一个样本 \mathbf{x}_1 ，我们在它的原始高维邻域里用 K 近邻算法 [1] 找到和它最近的三个样本 $\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ 。然后假设 \mathbf{x}_1 可

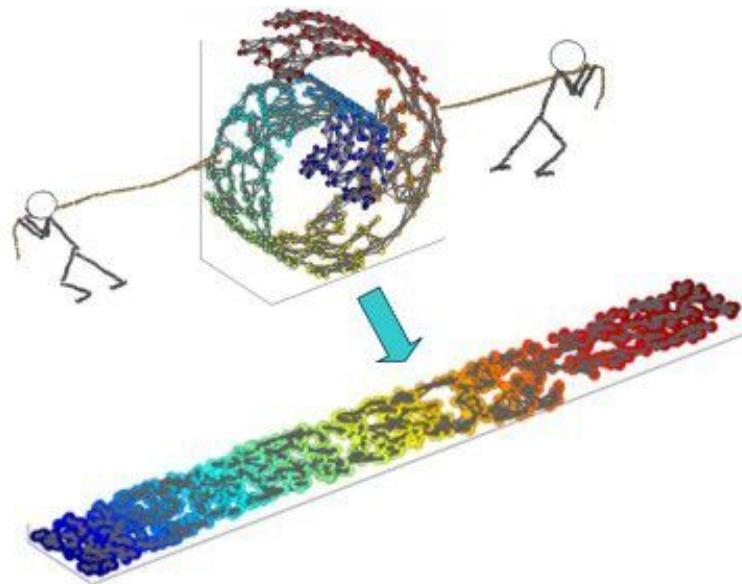


图 14.2: 流形降维过程

以由 $\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ 线性表示, 即:

$$\mathbf{x}_1 = \mathbf{w}_{21}\mathbf{x}_2 + \mathbf{w}_{31}\mathbf{x}_3 + \mathbf{w}_{41}\mathbf{x}_4, \quad (14.58)$$

其中 \mathbf{w} 为权重系数。在通过 LLE 降维后, 我们希望 \mathbf{x}_1 在低维空间对应的投影 \mathbf{z}_1 和 $\mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4$ 对应的投影 $\mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4$ 也尽量保持同样的线性关系, 即

$$\mathbf{z}_1 \approx \mathbf{w}_{21}\mathbf{z}_2 + \mathbf{w}_{31}\mathbf{z}_3 + \mathbf{w}_{41}\mathbf{z}_4. \quad (14.59)$$

也就是说, 权重系数在降维前后是尽量不变的。

首先, 我们将数据输入到 KNN 模型 (具体可参考本书相关章节) 中, 得到每个样本的邻域样本 $\mathbf{N}_i = KNN(\mathbf{x}_i, k) = (\mathbf{x}_{1i}, \mathbf{x}_{2i}, \dots, \mathbf{x}_{ki})$ 。接下来我们需要找到对应的权重系数, 来确定每个样本和邻域样本之间的线性关系。假设我们的数据 \mathbf{A} 由 N 个 d 维的样本 $(\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n)^T$ 组成, 损失函数可以表示为:

$$\mathcal{L}(\mathbf{W}) = \sum_{i=1}^N \left| \left| \mathbf{x}_i - \sum_{j=1}^k \mathbf{w}_{ji} \mathbf{x}_{ji} \right| \right|^2, \quad (14.60)$$

为了计算方便, 我们对权重系数做归一化处理, 对于每个样本 i :

$$\sum_{j=1}^k \mathbf{w}_{ji} = 1. \quad (14.61)$$

对于损失函数 (14.60), 我们做进一步的推导:

$$\begin{aligned}
 \mathcal{L}(\mathbf{W}) &= \sum_{i=1}^N \left\| \mathbf{x}_i - \sum_{j=1}^k \mathbf{w}_{ji} \mathbf{x}_{ji} \right\|^2 \\
 &= \sum_{i=1}^N \left\| \sum_{j=1}^k \mathbf{w}_{ji} \mathbf{x}_i - \sum_{j=1}^k \mathbf{w}_{ji} \mathbf{x}_{ji} \right\|^2, \\
 &= \sum_{i=1}^N \left\| \sum_{j=1}^k \mathbf{w}_{ji} (\mathbf{x}_i - \mathbf{x}_{ji}) \right\|^2 \\
 &= \sum_{i=1}^N \mathbf{W}_i^\top (\mathbf{X}_i - \mathbf{N}_i)^\top (\mathbf{X}_i - \mathbf{N}_i) \mathbf{W}_i
 \end{aligned} \tag{14.62}$$

其中 $\mathbf{W}_i = (\mathbf{w}_{i1}, \mathbf{w}_{i2}, \dots, \mathbf{w}_{ik})^\top$, $\mathbf{X}_i = (\mathbf{x}_i, \mathbf{x}_i, \dots, \mathbf{x}_i)$ (共 k 个 \mathbf{x}_i)。令 $\mathbf{S}_i = (\mathbf{X}_i - \mathbf{N}_i)^\top (\mathbf{X}_i - \mathbf{N}_i)$ 。在 (14.61) 中, 我们将其矩阵化: $\mathbf{W}_i^\top \mathbf{1}_k = 1$, 其中 $\mathbf{1}_k$ 为 $k \times 1$ 的全 1 向量。

为了最小化损失函数, 我们使用拉格朗日乘数法:

$$\mathbf{L}(\mathbf{W}_i) = \mathbf{W}_i^\top \mathbf{S}_i \mathbf{W}_i + \lambda (\mathbf{W}_i^\top \mathbf{1}_k - 1), \tag{14.63}$$

对 \mathbf{W}_i 进行求导并令其为 0:

$$\frac{\partial \mathbf{L}(\mathbf{W}_i)}{\partial \mathbf{W}_i} = 2\mathbf{S}_i \mathbf{W}_i + \lambda \mathbf{1}_k = 0, \tag{14.64}$$

再加上归一化的限制, 我们可以得到:

$$\mathbf{W}_i = \frac{\mathbf{S}_i^{-1} \mathbf{1}_k}{\mathbf{1}_k^\top \mathbf{S}_i^{-1} \mathbf{1}_k}, \tag{14.65}$$

其中分子表示对 \mathbf{S}_i^{-1} 所有行分别求和得到的列向量, 分母表示对该矩阵所有值的求和结果。

由于权重矩阵 \mathbf{W} 看作不变, 降维之后的损失函数为:

$$\mathcal{L}(\mathbf{Z}) = \sum_{i=1}^N \left\| \mathbf{z}_i - \sum_{j=1}^k \mathbf{w}_{ji} \mathbf{z}_{ji} \right\|^2. \tag{14.66}$$

其中 $\mathbf{Z} = (\mathbf{z}_1, \mathbf{z}_2, \dots, \mathbf{z}_N)$ 。同样的, 我们需要加上标准化限制: $\sum_{i=1}^N \mathbf{z}_i = 0$, $\sum_{i=1}^N \mathbf{z}_i \mathbf{z}_i^\top = N \mathbf{I}_{d \times d}$ 。为了矩阵运算的方便, 我们将权重矩阵 $\mathbf{W} \in \mathbb{R}^{k \times N}$ 扩展成 $N \times N$ 的维度。对于非邻域内的点样本对 $(\mathbf{z}_i, \mathbf{z}_j)$, 设置 $\mathbf{w}_{ji} = 0$ 。因此损失函数可以写成:

$$\mathcal{L}(\mathbf{Z}) = \sum_{i=1}^N \|\mathbf{Z}(I_i - \mathbf{W}_i)\|^2. \tag{14.67}$$

由于对于一个任意矩阵 $\mathbf{A} = (\mathbf{a}_1, \mathbf{a}_2, \dots, \mathbf{a}_N)$:

$$\sum_i (\mathbf{a}_i)^2 = \sum_i \mathbf{a}_i^\top \mathbf{a}_i = \text{tr}(\mathbf{A}\mathbf{A}^\top)$$

，其中 tr 表示矩阵的迹。因此我们可以得到:

$$\mathcal{L}(\mathbf{Z}) = \text{tr}(\mathbf{Z}(I_i - \mathbf{W}_i)(I_i - \mathbf{W}_i)^\top \mathbf{Z}^\top), \quad (14.68)$$

令 $\mathbf{M} = (I_i - \mathbf{W}_i)(I_i - \mathbf{W}_i)^\top$ ，则 $\mathcal{L}(\mathbf{Z}) = \text{tr}(\mathbf{Z}\mathbf{M}\mathbf{Z}^\top)$ 。再次运用拉格朗日乘数法:

$$\mathbf{L}(\mathbf{Z}) = \mathbf{Z}\mathbf{M}\mathbf{Z}^\top + \lambda(\mathbf{Z}\mathbf{Z}^\top - NI), \quad (14.69)$$

对 \mathbf{Z} 求导并令其为 0:

$$\frac{\partial \mathbf{L}(\mathbf{Z})}{\partial \mathbf{Z}} = 2\mathbf{M}\mathbf{Z}^\top + 2\lambda\mathbf{Z}^\top = 0, \quad (14.70)$$

于是我们得到:

$$\mathbf{M}\mathbf{Z}^\top = \hat{\lambda}\mathbf{Z}^\top (\hat{\lambda} = -\lambda). \quad (14.71)$$

最后，与 PCA 中的方法同理，我们只需选取 \mathbf{M} 中最小的 s 个非零特征值对应的特征向量组成数据矩阵，即能得到降维之后的数据。



参考文献

- [1] N. S. Altman. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, 46(3):175–185, 1992.
- [2] M. Bramanti, C.D. Pagani, and S. Salsa. *Matematica. Calcolo infinitesimale e algebra lineare*. Zanichelli, 2004.
- [3] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.
- [4] J. Demmel and W. Kahan. Accurate singular values of bidiagonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):873–912, 1990.
- [5] R. A. FISHER. The use of multiple measurements in taxonomic problems. *Annals of Eugenics*, 7(2):179–188, 1936.
- [6] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Johns Hopkins studies in the mathematical sciences. Johns Hopkins University Press, 1996.
- [7] D.C. Lay and E. Stade. *Linear Algebra and Its Applications*. Addison-Wesley Longman, Incorporated, 2005.
- [8] Makoto Sakai, Norihide Kitaoka, and Seiichi Nakagawa. Power linear discriminant analysis. In *2007 9th International Symposium on Signal Processing and Its Applications*, pages 1–4, Feb 2007.
- [9] Karl Pearson. On the criterion that a given system of deviations from the probable in the case of a correlated system of variables is such that it can be reasonably supposed to have arisen from random sampling. *Philosophical Magazine Series 1*, 50(302):11–28, 1900.
- [10] Joseph Lee Rodgers and W Alan Nicewander. Thirteen ways to look at the correlation coefficient. *The American Statistician*, 42(1):59–66, 1988.
- [11] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

第 15 章 推荐系统

15.1 概述

15.1.1 什么是推荐系统

新学期就要开始了，软件学院的同学们需要选修本学期的专业选修课。对于同学们来说，如何选到适合自己的课程是一个重要问题。一方面，同学们对这几年火爆的机器学习方向非常热情，另一方面，同学们也对图形学方向、移动开发方向感到好奇。对于学院来说，则希望每个方向的学生人数尽量均衡。一个可行的办法是学院给每位同学个性化的推荐。该如何给软件学院的同学们进行推荐呢？

我们每天都可以收到各种各样的推荐信息，无论是不同平台的热点新闻，还是电商网站上的推荐商品，亦或者是社交软件上的推荐好友，在这些看似熟悉平凡的场景中，都有着推荐系统的身影。依赖于不同的具体领域，推荐系统给消费者带来便利的同时，也为公司企业创造了更大的利润。

随着领域产业的发展以及移动终端的普及，网上信息量也随之大幅增长，用户能接受到的信息以及企业所获得的信息也大大增加，面对大量信息，用户无法有效从中获得真正对自己有用的信息，这即是信息过载 (Information Overload) 问题。目前，针对信息超载问题的解决方法主要是信息检索系统和个性化推荐系统。以搜索引擎为代表的信息检索系统，通过基于全文或其他内容的索引来获得相应的信息资源。但在以相同索引进行检索时，不同用户得到的结果是相同的。单纯的信息检索系统无法很好的满足不同用户的不同需求。而个性化推荐系统很好的解决了这一问题，利用已有信息，推荐系统提供了无效信息的过滤以及精准信息的推荐的服务。

推荐系统 (Recommender System) 作为一门独立学科发展近 20 年，但尚未由一个精确的定义。目前被广泛接受的定义是 Resnick 和 Varian[7] 提出的：“利用电子商务网站向客户提供商品信息和建议，帮助用户决定应该购买什么商品的系统”。

15.1.2 发展历史

推荐系统作为一个相对独立的研究方向，最早是来自明尼苏达大学 GroupLens 研究组推出的 GroupLens 系统 [6]，其中首次提出了基于协同过滤 (Collab-

orative Filtering) 的算法思想，该算法也是目前人们所说的基于用户的协同过滤推荐算法 (User-based Collaborative Filtering Algorithm)。

在后续的发展过程中，许多著名的算法逐渐成型，例如基于内容的推荐算法 (Content-based Algorithm)，基于物品的协同过滤推荐算法 (Item-based Collaborative Filtering Algorithm)，基于矩阵分解的协同过滤算法 (Matrix Factorization Algorithm) 等等。除此之外，基于深度学习以及知识图谱的推荐算法也在不断的发展，这些方法的补充和融合不断推动了推荐系统研究方向的丰富和发展。在第三节，我们会详细介绍推荐系统算法的不同分类。

15.1.3 推荐系统的描述方式

在开始正式的学习之前，我们先了解一下推荐系统涉及的几个概念，在讨论推荐问题时，用户 (User)、物品 (Item)、评价 (Review) 是 3 个基本的概念。

用户是推荐系统所针对推荐的对象，除了标识符外，通常会包含用户的一些相关信息，如用户的性别、年龄、所在城市、活跃城市等等。

物品是推荐给用户的对象，可以是商品，也可以是信息。同样的，除了标识符外，每个物品也有一些相关信息，例如种类、价格、关键词等等。

评价则是用户对物品的打分，表示了一定的信息。如在电影评价中，用户对于观看过的作品打分，分数的高低表示了用户对作品的喜爱或是讨厌；在电商平台上，用户对于商品的点击率，0 代表没有点击，1 代表点击查看过。

通常而言，推荐算法的输入来自以上的三种数据。作为输出，对于一个特定的用户，推荐算法给出的是一个推荐列表，该列表以不同的形式给出针对该用户可能感兴趣物品的优先排序信息。

在这里，我们给出一个最典型的形式化描述，这也是后续章节中算法所使用的描述。

稀疏矩阵 $\mathbf{R} \in \mathbb{R}^{n \times m}$ 表示 n 个用户对 m 件商品的评分矩阵。 r_{ij} 表示矩阵的第 i 个用户对第 j 个物品的打分，这是一个 0-10 的分值，0 代表没有打分，1 代表最不满意，10 代表最满意。

我们的工作即根据各种算法和已知信息，对没有打分的项进行预测，并根据预测分值的高低判断用户对物品的接受程度，从而进行推荐。

15.1.3.1 两个核心问题

对于一个推荐系统，它所要解决的问题有两个，分别是预测和推荐。预测所关心的主要问题是根据已有信息，推断每个用户对于没有打分的物品的可能打分。推荐索要解决的主要问题是，根据预测所得的结果将海量未打过分的物品排序，从而挑选出最终向用户推荐的物品。目前的算法主要针对解决预测问题，所以本章的重点也放在处理预测问题上。

随着推荐系统的广泛应用，如何对众多的推荐系统进行评价也就成为了学术界和工业界所关心的问题。总的来说，测评推荐系统的方法主要可以分为用户调研、在线测评和离线测评三种方式。离线测评通过划分训练集和测试集，根据推荐算法在测试集上的表现来衡量算法优劣以及调整参数，实施方便但指标结果不直观。由于推荐系统的效果依赖于多种因素，如用户界面、用户背景知识等在离线测试中很难反映的因素，在线测评即设计在线用户测试，通过 A/B 测试等方式，能够获得用户直观的满意度等指标。用户调研顾名思义，即通过问卷等方式询问用户喜欢哪种推荐结果，从而获得用户满意度等信息，但无法获得准确度等指标信息。

在实际投入使用中，用户对推荐系统的满意程度是最直观的指标，主要通过用户调研和在线测评的方式来获得。在用户调研中，通过调研问卷获得用户满意度和使用过程中的感受；在线测评中，可以通过用户点击率、停留时间、转化率等指标来衡量。我们主要介绍在离线测评中使用的一些指标，可以分为“准确度 (accuracy)”和“可用性 (Usefulness)”两个方面。首先是评分准确度，在使用评分体现推荐结果的系统中，预测评分和实际评分的关系是评价预测过程准确度的关键指标。RMSE(Root Mean Squared Error, 根均方差) 预测的是用户对每个商品感兴趣的程度。

$$\text{RMSE} = \sqrt{\frac{1}{|\mathbf{I}|} \sum_{(i,j) \in \mathbf{I}} (r_{ij} - \hat{r}_{ij})^2} \quad (15.1)$$

其中， \mathbf{I} 表示 $[R]_{n \times m}$ 中非零项的集合。MAE(Mean Absolute Error, 平均绝对误差) 相比 RMSE，对大误差的敏感度较低。

$$\text{MAE} = \sqrt{\frac{1}{|\mathbf{I}|} \sum_{(i,j) \in \mathbf{I}} |r_{ij} - \hat{r}_{ij}|} \quad (15.2)$$

有时候，我们并不是很关心预测评分的误差，而是关心预测评分和实际评分的趋势是否相同。一些表现好的物品可能绝对评分低，但是相对而言还是处于优先推荐的列表中。评价相关性的强弱，Pearson 相关系数是常见的指标：

$$r_j = \frac{\sum_j (\hat{r}_j - \bar{r})(r_j - \bar{r})}{\sqrt{\sum_j (\hat{r}_j - \bar{r})^2 \sum_j (r_j - \bar{r})^2}} \quad (15.3)$$

其中， \hat{r}_j 和 r_j 表示影片的预测评分和实际评分， \bar{r} 表示所有物品的平均分。

在推荐过程中，我们关心 TopN 列表的排序准确度，可以使用平均准确度指标 (Mean Average Precision, MAP)。在推荐列表长度 x 下的准确度 AP 为：

$$AP@x = \sum_{i=1}^x (prediction_i \times (change in recall)_i) \quad (15.4)$$

其中, $prediction_i$ 表示前 i 个结果的正确率, 如果第 i 个推荐结果正确, $(changeinrecall)_i$ 取值为 $\frac{1}{x}$, 否则为 0。求出 AP 后, MAP 即将所有用户的 AP 求平均。除了 MAP 外, 也可以使用召回率和准确率指标来衡量 TopN 推荐的准确性。在可用性方面, 覆盖率 (Coverage) 可以反映推荐的结果能否很好的覆盖所有物品, 是不是所有的物品都有被推荐的机会。

在本节中, 我们首先给出了推荐系统的概念, 并简要介绍推荐系统的发展历史, 推荐系统的基本描述方式和两个核心问题, 以及在推荐系统中的评价指标。在本章后续部分, 我们会在15.2节简要介绍用户画像, 我们将在15.3节介绍几种常见并且经典的推荐算法, 并尝试用这些算法来解决本章开头的选课推荐问题; 在15.4节我们讨论推荐系统在应用中的冷启动和稀疏性问题; 15.5节我们给出目前的一些前沿研究方向。

15.2 用户画像

在推荐系统中, 构建用户画像是一种常常被用来辅助提升推荐准确性的方法。用户画像是用来勾勒用户的具体特征, 如背景、用户特征、性格标签、用户场景等的方法, 将用户行为数据中提炼出用户的信息全貌, 从而解决将数据转化为商业价值的问题。在实际的应用中, 通常分为定性的用户画像分析和定量的用户画像分析。

15.2.1 定性分析用户画像

在定性用户画像分析中, 标签是一种常见而有效的手段。一方面, 标签提供了较细粒度上的用户特点分类, 能够很快的根据用户数据将用户分类。如对于学生, 有专业方向标签 (移动开发、计算机视觉), 有编程语言标签 (C++、Java、Go)。同时, 标签还具有较好的可解释性。由于标签本身就具有语义信息, 用标签作为分类可以获得较好的解释性。除了标签, 还可以基于知识进行用户画像分析。其主要思想是通过对用户对象的特征进行形式化的描述, 并通过验证、推理和解释等手段, 从而获得知识推荐系统。例如在电影推荐中, 已知用户具有“喜爱徐克导演的电影”这一特征, 而“徐克和周星驰都是香港导演”, 通过知识之间的联系, 从而推理用户也许也喜欢周星驰导演的作品。在这一方面, 目前有基于知识图谱 (Knowledge Graph) 的推荐系统的相关研究。

15.2.2 定量分析用户画像

相比定性分析, 定量分析相对较为简单。在定量用户画像分析中, 主要关注的是用户画像的粒度。如果对于用户画像的分析非常细化, 用户的刻画就越精密, 对于一定范围的推荐问题也就越准确。但同时, 随着粒度的细化, 对用户数据的

要求就越高，获取数据的成本也越高，同时模型的泛化性就相对越差。定量用户画像的粒度确定，往往需要根据业务要求和实际情况进行取舍。

15.3 算法的分类

传统的推荐算法主要分为基于内容的推荐算法 (Content-based Algorithm) 和基于协同过滤的推荐算法 (Collaborative Filtering, CF)，基于协同过滤的推荐算法又可以分为基于用户的推荐 (User-based Recommendation)、基于物品的推荐 (Item-based Recommendation)、基于模型的推荐 (Model-based Recommendation)。

一般来说，我们把推荐问题简化为“估计用户对其从未使用过的物品的评分”的问题。这个评分也就是我们为用户推荐物品的指标，有了这个评分，我们就可以将评分最高的 K 个物品推荐给用户了。我们先定义一组变量，假设 C 是用户集， I 是需要被推荐的物品集（用户从未使用过的物品集）， u 是效用函数，用来衡量物品 i 对用户 c 的推荐价值（用户 u 对物品 i 的评分估计），那么对于每个用户 $c \in C$ ，我们都希望找到使得 $u(c, i)$ 最大的 K 个物品 $i' \in I$ ，将这 K 个物品推荐给用户。

15.3.1 基于内容的推荐

15.3.1.1 传统的 CB 推荐方法

假设甲同学很喜欢 C++ 课程，那么推荐系统就很有可能将 Java 课程推荐给他，因为从内容上看，Java 课程和 C++ 课程非常相似，它们都属于编程语言的课程。这就是最基础的基于内容的推荐方法的思路。基于内容 (Content-based, 又称 CB) 的推荐旨在推荐与用户喜欢的物品相似的物品，其来源可追溯到认知科学、近似理论、信息检索、预测理论、管理科学和市场中的客户选择模型等相关研究 [8]。其中物品之间的相似性是根据物品的内容（比如课程的类型、授课老师）来确定。而我们要寻找一种方式来描述用户喜欢的物品和用户从未评过分的物品，计算出二者之间的匹配度，才可以进行推荐。推荐算法将分为以下三步来完成这个过程：

- 为每个物品 (Item) 构建一个物品的属性资料 (Item Profile)
- 为每个用户 (User) 构建一个用户的喜好资料 (User Profile)
- 计算用户喜好资料与物品属性资料的相似度，相似度高意味着用户可能喜欢这个物品，相似度低往往代表着用户不喜欢这个物品

新学期就要开始了，甲同学需要登陆教务系统选择下学期的专业选修课。接下来我们就要一步一步使用 CB 的方法来为甲同学推荐 2 门课程。

构建属性资料



首先，我们要构建课程的属性资料。物品的属性资料，是指被推荐的物品的详细属性，用特征向量来表示。如课程的特征向量可以是 <课程类型，授课老师，平均给分> 这三个属性对应的权重矩阵。

表 15.1: 课程及其对应属性

	课程类型	授课老师	平均给分
数据库	算法类	杨老师	86
算法设计	算法类	李老师	84
软件开发实训	开发类	许老师	88
机器学习	算法类	杨老师	86
IOS 开发	开发类	许老师	87
安卓开发	开发类	许老师	89
人工智能	算法类	杨老师	90
智能人机交互	产品设计类	李老师	82

我们假设 Item Profile 中包含课程类型，授课老师，平均给分这三个属性。这样一来，数据库的 Item Profile 就可以这样表示：[算法类，杨老师，85]，然而这种自然描述的 Item Profile 不能直接为代码所用，我们需要的是计算机能处理的表达方式。所以还要把自然描述的 Item Profiles 映射成程序能够读懂的数据结构，这里就需要进行一个映射——将自然语言描述的 Item Profile 转换成权重矩阵。对于课程类别和授课老师这种离散型特征，我们用 0 或者 1 表示所有的课程类别和授课老师比较合适（感兴趣的读者可深入阅读独热编码的相关文献）；对于平均给分这种实值，我们可以不做任何转化直接使用，也可以进行一定的预处理。我们将数据库这门课的详细特征表示为 1×8 的矩阵 F：[算法类，开发类，产品设计类，杨老师，李老师，许老师，王老师，平均给分]，第 0 个元素代表课程属于算法类、第 1 个元素代表课程属于开发类，以此类推，并初始化权重矩阵 w_{db} 为 $\mathbf{0}$ 。映射过程如下：

- 离散型特征

离散型特征映射的方法很直观：数据库课程属于算法类课程，那么数据库的行向量 w_{db} 中， $w_{db,0}$ 置为 1，以此表示数据库为算法类课程；此课程授课老师为李老师，将 $w_{db,4}$ 置为 1，以此表示数据库的授课老师是李老师。这样我们就得到了离散型特征映射后的权重矩阵表示：[1,0,0,0,1,0,0,0]

- 连续型特征

对于平均给分这种实值，我们可以不做任何转化直接使用，也可以进行一定的预处理。在这里，平均给分的数值上比布尔值大很多，为了不让它在推荐中太占主导地位，可以采用 min-max 标准化方法对它进行归一化。

min-max 标准化是对原始数据的线性变换，使结果落到 [0,1] 区间，转换函数为

$$x^* = \frac{x - \min}{\max - \min}, \quad (15.5)$$

式中 \max 为样本数据的最大值, \min 为样本数据的最小值。在我们的给分数据中, \max 为 88 分, \min 为 84 分, 那么数据库的平均给分这项的权重计算结果, 即 $w_{db,7}$, 就为 0.5。现在, 我们数据库的属性资料, 也就是权重矩阵 w_{db} 可以完整的表示为 [1, 0, 0, 0, 1, 0, 0, 0.5] 了。用同样的方法我们得到了所有课程的 Item Profile。

表 15.2: 所有课程的 Item Profile

权重矩阵	
数据库	$w_{db}[1, 0, 0, 0, 1, 0, 0, 0.5]$
算法设计	$w_{ad}[1, 0, 0, 0, 1, 0, 0, 0.25]$
软件开发实训	$w_{sd}[0, 1, 0, 0, 0, 1, 0.75]$
机器学习	$w_{ml}[1, 0, 0, 1, 0, 0, 0, 0.5]$
ios 开发	$w_{ios}[0, 1, 0, 0, 0, 1, 0, 0.625]$
C++ 编程	$w_{cp}[0, 1, 0, 0, 0, 1, 0, 0.875]$
人工智能	$w_{ai}[1, 0, 0, 1, 0, 0, 0, 1]$
智能人机交互	$w_{hi}[0, 0, 1, 0, 1, 0, 0, 0]$

构建用户资料

到目前为止, 我们已经为所有的 Item 进行建模了, 模型就是 Item Profile, 也就是那个 1×8 维的权重矩阵。但是这还不够, 我们还需要为用户进行建模, 所谓的对用户建模, 就是构造 User Profiles, 而这个 User Profiles 就相当于用户的偏好。用户的喜好资料是根据用户以往对某些物品的评分和那些物品的属性资料得到的, 它代表了用户对物品的不同属性的喜好程度, 这里我们假设 w_c 为用户 c 对物品的各个特征的喜好程度的权重矩阵。

我们有一个评分表

表 15.3: 甲同学与乙同学对课程的评分

	数据库	算法设计	C++ 编程
甲	8.5	9	9.5
乙	7	8	

表的含义是甲同学对数据库、算法设计、C++ 编程课程的评分分别为 8 分, 9 分, 9.5 分, 乙同学对算法设计、C++ 编程的评分分别为 7 分, 8 分。利用这一信息, 我们就可以开始为甲同学构建 User Profile 了, 方法如下:

步骤 1 算出甲同学所有打分的平均分, 在这个例子中甲同学给出的平均分 $Avg = (8+9+9.5) / 3 = 9$

步骤 2 对于离散型特征, 利用公式:

$$\frac{\sum(r_i - Avg)}{n} \quad (15.6)$$

算出甲同学对各个特征的喜爱程度。计算甲同学对算法类课程的喜爱程度时， r_i 是所有属于算法类课程的、而且是甲同学评过分的课程，Avg 就是 1 中算出来的平均分，n 就是所有所有属于算法类课程的、而且是甲同学评过分的课程的数量。在这个例子中公式应该等于 $((8.5 - 9) + (9 - 9)) / 2 = -0.25$ ，也就是说，甲同学对算法类的喜爱程度可以用-0.25 这个数值来反应

步骤 3 对于连续型特征，我们采用公式计算：

$$w_{c,n} = \frac{\sum r_i * w_i}{\sum r_i}。 \quad (15.7)$$

其中 $w_{c,n}$ 表示用户 c 对连续型特征 n 的喜爱程度的权重值， w_i 为物品 i 的特征 n 的权重。甲同学对平均给分这一课程特征的喜爱程度可以表示为：

$$w_{甲,8} = \frac{8.5 \times 0.5 + 9 \times 0.25 + 9.5 \times 0.875}{8.5 + 9 + 9.5} = 0.55。$$

步骤 4 User Profiles 和 Item profiles 的维度是相同的，都等于物品内容的特征数量，与 Item Profiles 的矩阵不同的是，User Profiles 中矩阵的元素不再是 0,1 或是平均分的数值，而是由 2 和 3 中计算得来的对每个属性的喜爱程度，回想一下，在 2.2 中我们已经做出了假设：矩阵的第 0 个元素代表算法类课程，所以这里的第 0 个元素是-0.25，表示甲同学对算法类课程的喜爱程度是-0.25。同理，可以算出甲同学对课程的其他属性的喜爱程度，所以最终甲同学的用户资料矩阵 w_c 可以表示为 [-0.25, 0.5, 0, -0.5, 0, 0.5, 0, 0.55]。

预测与推荐

利用余弦相似度的公式 (1.8) 计算 User Profile 和 Item Profile 的距离，来衡量给定的 User(c) 喜好和给定的 Item(i) 之间的相似度。余弦相似度的值越大说明 c 越有可能喜欢 i。

$$u(c, i) = \cos(\vec{w}_c, \vec{w}_i) = \frac{\vec{w}_c \cdot \vec{w}_i}{\|\vec{w}_c\|^2 \times \|\vec{w}_i\|^2} = \frac{\sum_{s=1}^K (w_{s,c} w_{s,i})}{\sqrt{\sum_{s=1}^K w_{s,c}^2} \sqrt{\sum_{s=1}^K w_{i,s}^2}}。 \quad (15.8)$$

我们接下来可以遍历整个课程列表，计算甲同学喜好与每个课程的相似度，以机器学习课程为例，前面我们得到了机器学习课程的 Item Profile(w_{ml}) 为 [1,0,0,1,0,0,0,0.5]，和甲同学的 User Profile 余弦相似度为

$$sim_{甲,ml} = \frac{-0.25 \times 1 + \dots + 0.55 \times 0.5}{\sqrt{(-0.25)^2 + 0.5^2 + (-0.5)^2 + 0.5^2 + 0.55^2} \sqrt{1^2 + 1^2 + 0.5^2}} = -0.27。$$

用同样的计算，得到所有课程和甲同学喜好的相似度，选择相似度最高的前 k 个课程，推荐给甲同学，大功告成！

15.3.1.2 基于线性回归的 CB 推荐方法

这种最基础的基于内容的推荐是利用用户的喜好和物品的特征的相似性为指标来做推荐。而基于内容的推荐系统中还有一种基于线性规划的推荐方法。

还记得前面我们学习过的线性回归算法吗？在那里，我们以高年级学生们在***课程的成绩构建特征向量，作为模型输入 x ，高年级学生们机器学习的成绩作为模型输出 y 训练模型，来预测低年级学生们在机器学习课程中可能获得的分数。那在我们课程推荐的任务里，可不可以将不同课程的 profiles 作为模型输入 x ，甲同学对他们的评分作为模型输出 y 训练模型，来预测甲同学对一门新的课程的评分，以此来做推荐呢？其实这就是使用线性规划的基于内容的推荐方法的思路。

甲同学评过的每个课程都是一个训练样本，其中数据库这门课的特征向量 $x_1 = [1, 0, 0, 0, 1, 0, 0, 0.5]$, $y_1 = 8.5$ ，算法设计这门课的特征向量 $x_2 = [1, 0, 0, 0, 0, 1, 0, 0, 0.25]$, $y_2 = 9$ ，使用所有的课程构成的训练集 $\{(x_1, y_1), (x_2, y_2) \dots (x_n, y_n)\}$ 训练模型，用梯度下降求出 θ ，假设我们通过线性规划求出了甲同学的参数 θ 为 $[4, 5.5, 3.5, 6, 4.5, 5.75, 4, -2]$ ，甲同学对课程的评分可用 $x_{ml}\theta$ 计算出，为 9 分。对于其他课程也一样，最后我们计算出所有课程的预测评分，将得分最高的前 K 个课程推荐给甲同学即可。

接下来我们思考几个问题，

- 如果两门课程的特征向量相同，那我们在算法中这两个课程呢？
- 甲同学的确很喜欢算法类的课程，但是他以前从未上过产品设计的课程，所以我们没办法得到他对于产品设计类的课程的喜爱程度，给甲同学推荐的课程就只能是和他评过分的课程相似的课程了，这似乎不利于他探索新的知识领域？
- 如果我们是给大一的同学推荐课程，我们并没有任何同学们的课程评分，我们又怎么能够通过 CB 的方法进行推荐呢？

这其实就对应着 UB 的算法的内容分析有限、过于专门化、用户冷启动的问题。

15.3.2 基于协同过滤的推荐系统

15.3.2.1 基于历史的协同过滤

基于用户的协同过滤

甲同学和他的师兄乙同学平日里经常在一起讨论技术问题，兴趣方向很一致，他们都喜欢大一的时候上过的线性代数和 C++ 编程的课程。师兄告诉甲同学，他大三选的机器学习课程非常有趣，他学起来也比较轻松，甲同学心里跃跃欲试，想去体验一下这门课程。

上面这个情景呢，就有点基于用户的协同过滤（User-based Collaboration Filter，



又称 UBCF) 推荐的意思了。基于用户，也就是根据邻居用户的偏好信息产生对目标用户的推荐。它基于这样一个假设：如果一些用户对某一类项目的打分比较接近，则他们对其他类项目的打分也比较接近。甲同学和师兄对线性代数和 C++ 课程的评价比较接近，则他们对机器学习的课程评价也很可能接近，在师兄对机器学习课程评价很高的情况下，把这门课推荐给甲同学就非常合情合理了。

UBCF 推荐算法采用统计计算方式搜索目标用户的相似用户，并根据相似用户对项目的打分来预测目标用户对指定项目的评分，最后选择相似度较高的前若干个相似用户的评分作为推荐结果，反馈给用户。也就是说，这里的效用函数 $u(c, i)$ 是基于 $u(c_s, i)$ 计算得到的，其中 $c_s \in C$ ，是用户 c 的相似用户。UBCF 推荐算法的核心就是通过相似性度量方法计算出最近邻居集合，并将最近邻居的评分结果作为推荐预测结果返回给用户。

我们现在有这样一个评分表，行表示同学，列表示课程，表的内容是同学对课程的评分。

表 15.4: 各同学对课程的评分

	线性代数	C++	数学分析	工程制图	机器学习
2016 级甲	8	8.5	8.5	6	
2015 级乙	8	9	9	6.5	8.5
2015 级丙	7	6	6	9	7.5
2015 级丁	7		8	8	
2015 级戊	9.5	8	8.5	7	9

从上表评分我们可以直观的看到，甲同学和乙同学的课程评分相似度很高，因此，甲同学对机器学习这门课程的预测评分应该可以由乙同学对机器学习的打分来得到。在真实的预测中，推荐系统只对前若干个邻居进行搜索，并根据这些邻居的评分为目标用户预测指定项目的评分。由上面的例子不难知道，UBCF 推荐算法的步骤可以分为用户相似性度量、最近邻居查询和预测评分。

对于甲同学，我们先找到甲同学评过分的课程集 I= 线性代数，C++，数学分析，工程制图 再看哪些同学在课程集 I 上评过分，定义出用户集乙，丙，戊 定义出用户集之后，我们需要计算甲同学的偏好和用户集中的用户的偏好的相似度。计算用户相似度方法很多种，下面我们列出常用的三种，分别是：余弦相似性、相关相似性以及修正的余弦相似性。

余弦相似性 (Cosine)：用户一项目评分矩阵可以看作是 n 维空间上的向量，对于没有评分的项目将评分值设为 0，余弦相似性度量方法是通过计算向量间的余弦夹角来度量用户间相似性的。设向量 i 和 j 分别表示用户 i 和用户 j 在 n 维空间上的评分，则用基于用户的协同过滤的推荐算法中用户 i 和用户 j 之间的相似性为

$$sim(i, j) = \cos(\vec{i}, \vec{j}) = \frac{\vec{i} \cdot \vec{j}}{\|\vec{i}\| * \|\vec{j}\|}。 \quad (15.9)$$

修正的余弦相似性 (Adjusted Cosine): 余弦相似度未考虑到用户评分尺度问题, 如在评分区间 [1 — 10] 的情况下, 对甲同学来说评分 8 以上就是自己喜欢的, 而对于同学乙, 评分 9 以上才是自己喜欢的。通过减去用户对项的平均评分, 修正的余弦相似性度量方法改善了以上问题。用户 i 和用户 j 之间的相似性为

$$sim(i, j) = \frac{\sum_{c \in I_{i,j}} (r_{ic} - \bar{r}_i)(r_{jc} - \bar{r}_j)}{\sqrt{\sum_{c \in I_i} (r_{ic} - \bar{r}_u)^2} \sqrt{\sum_{c \in I_j} (r_{jc} - \bar{r}_u)^2}}. \quad (15.10)$$

$I_{i,j}$ 表示用户 i 和用户 j 共同评分过的项集合, I_i 和 I_j 分别表示用户 i 和用户 j 评分过的物品集合, r_{ic} 和 r_{jc} 分别表示用户 i 和用户 j 给物品 c 的评分, \bar{r}_i 和 \bar{r}_j 分别表示用户 i 和用户 j 对物品的平均给分。

相关相似性 (Correlation)。此方法是采用皮尔森 (Pearson) 相关系数来进行度量。设 I_{ij} 表示用户 i 和用户 j 共同评分过的项目集合, 则用户 i 和用户 j 之间相似性为

$$sim(i, j) = \frac{\sum_{c \in I_{ij}} (r_{ic} - \bar{r}_i)(r_{jc} - \bar{r}_j)}{\sqrt{\sum_{c \in I_{ij}} (r_{ic} - \bar{r}_u)^2} \sqrt{\sum_{c \in I_{ij}} (r_{jc} - \bar{r}_u)^2}}. \quad (15.11)$$

我们在这里采用皮尔森相关系数来度量甲同学和乙同学的相似度:

$$\hat{r}_{\text{甲}} = (8 + 8.5 + 8.5 + 6) / 4 = 7.75,$$

$$\hat{r}_{\text{乙}} = (8 + 9 + 9 + 6.5 + 8.5) / 5 = 8.2,$$

$$sim(\text{甲}, \text{乙}) = \frac{0.25 \times (-0.2) + \dots + (-1.75) \times (-1.7)}{\sqrt{(0.25^2 + \dots + (-1.75)^2) \times ((-0.2)^2 + \dots + (-1.7)^2)}} = 0.975.$$

同理, 计算出甲同学和用户集里用户的相似度分别为 -0.987, -0.278, 0.664, 我们这里选取和甲同学相似度最高的前两位, 即两个最近邻居乙和丙。预测评分时, UBCF 算法将用户 c 的所有相似用户的评分用聚集函数计算进行预测, 即

$$r_{c,i} = \operatorname{argmax}_{c' \in \hat{C}} r_{c',i}, \quad (15.12)$$

\hat{C} 代表了和用户 c 最相似的 K 个用户集。常用的聚集函数有:

$$r_{c,i} = \frac{1}{N} \sum_{c' \in \hat{C}} r_{c',i}; \quad (15.13)$$

$$r_{c,i} = \lambda \sum_{c' \in \hat{C}} sim(c, c') \times r_{c',i}; \quad (15.14)$$

$$r_{c,i} = \bar{r}_c + \lambda \sum_{c' \in \hat{C}} sim(c, c') \times (r_{c',i} - \bar{r}_{c'}). \quad (15.15)$$

λ 是正则化因子，通常情况下

$$\lambda = \frac{1}{\sum_{c' \in \hat{C}} |sim(c, c')|}。 \quad (15.16)$$

那么预测甲同学机器学习的评分时，可以用乙同学和丙同学对机器学习的评分来作为数据计算。计算公式如下：

$$r_{c,i} = \bar{r}_c + \frac{\sum_{c' \in K} sim(c, c')(r_{c',i} - \bar{r}_{c'})}{\sum_{c' \in K} |sim(c, c')|}。 \quad (15.17)$$

其中 K 是我们选取最近邻居的集合， $sim(c, c')$ 表示用户相似度， $|sim(c, c')|$ 表示用户相似度的绝对值， \bar{r}_c 表示用户 c 的平均给分， \bar{r}'_c 表示用户 c' 的平均给分。

$$r_{甲,ml} = 7.75 + \frac{0.975 \times (8.5 - 8.2) + 0.664 \times (9 - 8.4)}{0.975 + 0.664} = 8.17。$$

同理我们得到甲同学对其他课程的评分估计为：从表中我们可以得到排名前

表 15.5：甲同学对其他课程的评分估计

课程	预测评分
机器学习	8.17
IOS 开发	8
人工智能	8.55
智能人机交互	8.15

二的两个课程分别是机器学习和人工智能，我们可以放心的把这两门课推荐给甲同学了！**基于物品的协同过滤**

接下来我们学习基于物品的协同过滤。在日常的学习生活中，我们也常常会有这样的发现：喜欢软件体系结构课程的同学，往往对软件需求分析、软件测试这一类的课程也比较感兴趣；喜欢图形学的同学，则往往对数字媒体、计算机视觉的相关课程感兴趣。基于物品的协同过滤 (Item-based Collaboration Filter, IBCF) 也是基于这样一个思想：给用户推荐和他们之前喜欢的物品类似的物品，而评价两个物品 A, B 是否相似，则根据是否喜欢物品 A 的人也喜欢物品 B 。基于物品的协同过滤算法主要分为两步：计算物品之间的相似度，根据物品的相似度和用户的历史行为给用户进行推荐。下面我们介绍一种简单的 IBCF 算法——Slope one[4]。Slope one 算法是 Daniel Lemire 在 2005 年提出的一个 IBCF 算法，首先计算任意两个物品 i, j 之间的差值 $R(ij)$

$$R(ij) = \frac{\sum_{u \in I_i \cap I_j} (r_{ui} - r_{uj})}{|I_{i,j}|}。 \quad (15.18)$$

其中， I_i 表示所有给物品 i 评分的用户， $|I_j|$ 表示所有给物品 j 评分的用户数量， $I_{i,j}$

表示所有给物品 i, j 评分的用户, r_{ui} 表示用户 u 给物品 i 的评分。接着, 根据用户历史行为和差值, 预测用户 u 对未评分的物品 j 的评分 p_{uj}

$$p_{uj} = \frac{\sum_{i \in N(u)} |I_i \cap I_j| (r_{ui} - R(ij))}{\sum_{i \in N(u)} |I_{i,j}|}, \quad (15.19)$$

其中, $N(u)$ 表示用户 u 评过分的物品。

在15.4的例子中, 对于 5 门课程, 我们可以用公式15.18计算任意两门课程之间的差值, 如15.6所示

表 15.6: 课程的差值

	线性代数	C++	数学分析	工程制图	机器学习
线性代数		1	0.9	1.8	0.5
C++	1		0.125	2.25	1
数学分析	0.9	0.125		1.9	0.833
工程制图	1.8	2.25	1.9		1.833
机器学习	0.5	1	0.833	1.833	

得到两门课程之间的差值后, 我们就可以由15.19预测每位同学对未评分的课程的评分, 如表15.7

表 15.7: IBCF 预测结果

	线性代数	C++	数学分析	工程制图	机器学习
2016 级甲	8	8.5	8.5	6	6.583
2015 级乙	8	9	9	6.5	8.5
2015 级丙	7	6	6	9	7.5
2015 级丁	7	6.542	8	8	6.611
2015 级戊	9.5	8	8.5	7	9

这么我们就可以由基于物品的协同过滤算法得到预测评分从而进行推荐了。

15.3.2.2 基于模型的推荐

上述的基于记忆的协同过滤算法虽然思想清晰、实现简单, 但由于需要对所有用户或物品进行相似度计算, 在真实应用中, 随着用户规模和物品规模的上升, 评分矩阵的行列数会随之增大, 同时由于用户只能对有限的物品评分, 矩阵内部非常稀疏, 系统的运算开销非常大。基于模型的推荐是通过将推荐问题建模成一个分类问题或是回归问题, 利用将评分矩阵视为输入数据, 从而利用机器学习算法来构建预测模型, 从而进行推荐。本节, 我们基于一个例子来介绍基于模型的推荐中, 一个经典的算法模型——基于矩阵分解 (Matrix Factorization) 的推荐。

奇异值分解 (SVD)

奇异值分解是 Golub[3] 提出的一种正交矩阵分解法，在统计学和信号处理领域有很好的表现。后来 SVD 被用于分析文档的潜在语义因子 (latent semantic indexing)[2]。之后很快被引入推荐系统领域。

SVD 的基本原理是将给定的矩阵 $\mathbf{M} \in \mathbb{R}^{n \times m}$ 分解成三个矩阵的乘积形式，即

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^T. \quad (15.20)$$

公式15.20中，矩阵 $\mathbf{U} \in \mathbb{R}^{n \times n}$ 和 $\mathbf{V} \in \mathbb{R}^{m \times m}$ 是两个酉矩阵；矩阵 $\Sigma \in \mathbb{R}^{n \times m}$ 是对角矩阵，其对角线上的值为 \mathbf{M} 的奇异值。

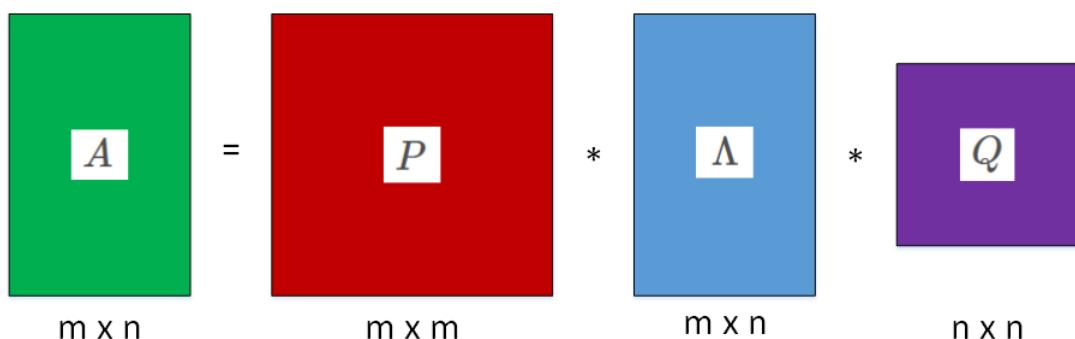


图 15.1: SVD

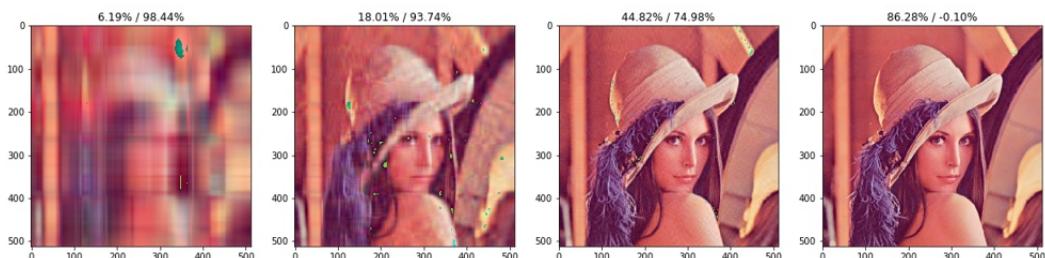


图 15.2: 不同特征值对图片清晰度的影响

矩阵 Σ 中的奇异值由大到小排列，代表矩阵 \mathbf{M} 中的重要特征。在大部分情况下，矩阵 Σ 中前若干个奇异值即可近似表示矩阵。

正如图15.2所示，往往只需要很少的奇异值，就可以表示原矩阵的主要特征。由此我们很自然的联想到，对于评分矩阵，我们可以通过 SVD 将其分解成表示用户特征的矩阵 \mathbf{P} 和表示物品特性的矩阵 \mathbf{Q} ，然后通过奇异值实现降维。

矩阵分解

在大部分应用场景下，推荐系统的评分矩阵 $\mathbf{R} \in \mathbb{R}^{n \times m}$ 是稀疏的，从 SVD 中，我们想到：评分矩阵 \mathbf{R} 的主要特征只需要较小的维度即可表示，将评分矩阵分解成两个规模更小的特征矩阵 $\mathbf{P} \in \mathbb{R}^{n \times k}$, $\mathbf{Q} \in \mathbb{R}^{k \times m}$ 的乘积，从而减少计算量。矩阵分解 [11] 的基本形式如下：

$$\mathbf{R} \approx \mathbf{P} \times \mathbf{Q} = \hat{\mathbf{R}}. \quad (15.21)$$

其中, $\mathbf{p}_i^T = (p_{i1}, p_{i2}, \dots, p_{ik})^T, i = 1, 2, \dots, n$ 表示 \mathbf{P} 的行, $\mathbf{q}_j = (q_{1j}, q_{2j}, \dots, q_{kj}), j = 1, 2, \dots, m$ 表示 \mathbf{Q} 的列。当 k 远小于 n, m 时, 求解问题的规模也大大减小, 我们可以尝试通过机器学习的方法来求解两个特征矩阵 \mathbf{P}, \mathbf{Q} 。设想这样一个问题, 在获得了过去三届学生课程成绩的数据后, 学校希望根据以往学生的选课数据以及最终的课程成绩数据, 给即将升入三年级选择专业方向的应届生提供更适合的推荐课程, 以此缓解专业方向选课不平衡的问题。回顾第一节给出的描述, 令 $\mathbf{R} \in \mathbb{R}^{n \times m}$ 表示 n 个学生在 m 门课程的成绩评分矩阵 r_{ij} 表示矩阵的第 i 个学生在第 j 门课程的成绩评分, 这是一个 0-5 的分值, 0 代表没有选修过, 1 代表评分最低, 5 代表评分最高。我们采用 RMSE 作为评价指标, 同时机器学习的损失函数定义如下

$$\hat{r}_{ij} = \langle \mathbf{p}_i, \mathbf{q}_j^T \rangle; \quad (15.22)$$

$$\mathcal{L} = \text{RMSE}^2 = \frac{1}{|\mathbf{I}|} \sum_{(i,j) \in \mathbf{I}} (r_{ij} - \hat{r}_{ij})^2. \quad (15.23)$$

因此, 我们将求解推荐预测问题, 转化成了求解两个特征矩阵的问题, 而又可以进一步定义成一个最小化问题,

$$(\mathbf{P}, \mathbf{Q}) = \arg \min_{(\mathbf{P}, \mathbf{Q})} \mathcal{L}. \quad (15.24)$$

尽管问题规模已经减小, 但是我们仍需要处理 $(n + m) \times k$ 个参数。由于推荐系统中评分矩阵一般是稀疏的, 因此集合 \mathbf{I} 是较小的, 如果直接使用 15.24 作为我们的损失函数, 容易导致过拟合。为了避免这一情况, 我们可以考虑前面章节中介绍的正则化技术。

$$\mathcal{L} = \frac{1}{|\mathbf{I}|} \sum_{(i,j) \in \mathbf{I}} (r_{ij} - \hat{r}_{ij})^2 + \lambda \left(\sum_i n_{p_i} \|\mathbf{p}_i\|^2 + \sum_j n_{q_j} \|\mathbf{q}_j\|^2 \right). \quad (15.25)$$

公式 15.25 为加入正则化处理的损失函数, 其中 n_{p_i} 表示用户 \mathbf{p}_i 所评分的物品数量, n_{q_j} 表示物品 \mathbf{q}_j 的用户评分数。令 I_i 表示用户 \mathbf{p}_i 所评分的物品集合, 则 n_{p_i} 等于 $|I_i|$ 。令 I_j 表示物品 \mathbf{q}_j 所被评分的用户集合, 则 n_{q_j} 等于 $|I_j|$ 。这个正则化满足吉洪诺夫正则化 (Tikhonov regularization)。

什么是吉洪诺夫正则化? 在线性回归章节中, 我们介绍了线性模型 $\mathbf{Ax} = \mathbf{y}$ 在损失函数为 $\|\mathbf{Ax} - \mathbf{y}\|^2$ 的闭式解 $\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{y}$, 模型存在闭式解的条件是 \mathbf{A} 是一个满秩矩阵。在 \mathbf{A} 不满足满秩的条件下, 问题可能无解、存在多个解或解是不稳定的, 这种问题被称为不适定问题 (Ill-posed Problem)。这种情况下, 往往需要对问题加以限制, 使得其变为一个适定问题, 从而求得近似解。对于无解的情况, 通过增加条件找到近似解; 对于存在多个解的情况, 通过增加限制缩小解空间; 对于解不稳定, 通过加以限制使得保证连续依赖性。吉洪诺夫正则化定义损

失函数为 $\|\mathbf{Ax} - \mathbf{y}\|^2 + \|\Gamma\mathbf{x}\|^2$, 其中 Γ 表示吉洪诺夫矩阵, 通常取单位矩阵的倍数 $\alpha\mathbf{I}$ 。这样一来, 可以求得闭式解为 $\mathbf{x} = (\mathbf{A}^T\mathbf{A} + \Gamma^T\Gamma)^{-1}\mathbf{A}^T\mathbf{y}$, 从代数上解释, 吉洪诺夫正则化在 \mathbf{A} 不是满秩的条件下, 通过加上 $\Gamma^T\Gamma$ 使得括号内的矩阵为满秩矩阵。

观察损失函数, 我们可以发现, 它与前面章节提到的损失函数有些不同, 包含有两个要求解的特征矩阵 \mathbf{P}, \mathbf{Q} , 同时这两个矩阵耦合在一起, 并不好求解。因此在这里, 我们介绍一种新的方法——交替最小二乘法 (Alternating Least Square, ALS) 来进行求解这一类稀疏矩阵的矩阵分解问题。

交替最小二乘法

交替最小二乘法的基本流程如下:

- 步骤 1 对特征矩阵 \mathbf{Q} 进行初始化。
- 步骤 2 固定 \mathbf{Q} , 通过梯度下降法对损失函数进行求解, 从而求解特征矩阵 \mathbf{P} 。
- 步骤 3 固定 \mathbf{P} , 通过梯度下降法对损失函数进行求解, 从而求解特征矩阵 \mathbf{Q} 。
- 步骤 4 重复 Step2, Step3 直到满足终止条件。

现在我们介绍一下, 在 \mathbf{Q} 固定的情况下, 如何求解特征矩阵 \mathbf{P} 。对于 \mathbf{p}_i 来说, 求解的过程可以看成一个在已知用户评分过的物品信息的前提下, 线性最小二乘求解的过程, 其目的等效于求得损失函数的最小值。如果我们对 \mathbf{p}_i 的每一项求偏导

$$\begin{aligned} \frac{1}{2} \frac{\partial f}{\partial p_{ik}} &= 0, \quad \forall k; \\ \Rightarrow \sum_{j \in I_i} (\mathbf{p}_i \mathbf{q}_j^T - r_{ij}) q_{kj} + \lambda n_{p_i} p_{ik} &= 0, \quad \forall k; \\ \Rightarrow \sum_{j \in I_i} q_{kj} \mathbf{q}_j \mathbf{p}_i^T + \lambda n_{p_i} p_{ik} &= \sum_{j \in I_i} q_{kj} r_{ij}, \quad \forall k. \end{aligned} \tag{15.26}$$

可以看到, 对于 \mathbf{p}_i 而言, 其 k 项均可以对应15.26的关系。我们自然地想到, 是否可以通过矩阵运算, 一次性将这 k 项都求解出来呢? 我们定义 $\mathbf{Q}_{I_i} \in \mathbb{R}^{k \times n_{p_i}}$ 是 \mathbf{Q} 的一个子矩阵, \mathbf{Q}_{I_i} 的每一列都是 I_i 项在 \mathbf{Q} 的对应一列。定义 $\mathbf{R}(i, I_i) \in \mathbb{R}^{1 \times n_{p_i}}$, 其每一项都是 I_i 项在 \mathbf{R} 的第 i 行的对应项。定义 $\mathbf{E} \in \mathbb{R}^{n_{p_i} \times n_{p_i}}$ 是一个单位矩阵。我们不难推出

$$\begin{aligned} (\mathbf{Q}_{I_i} \mathbf{Q}_{I_i}^T + \lambda n_{p_i} \mathbf{E}) \mathbf{p}_i^T &= \mathbf{Q}_{I_i} \mathbf{R}^T(i, I_i), \quad \forall i; \\ \Rightarrow \mathbf{p}_i &= (\mathbf{A}_i^{-1} \mathbf{V}_i)^T. \end{aligned} \tag{15.27}$$

其中, $\mathbf{A}_i = \mathbf{Q}_{I_i} \mathbf{Q}_{I_i}^T + \lambda n_{p_i} \mathbf{E}$, $\mathbf{V}_i = \mathbf{Q}_{I_i} \mathbf{R}^T(i, I_i)$ 。由此, 我们可以推出整个特征矩阵 \mathbf{P} 。同理, 我们可以推导出固定 \mathbf{P} 后, 求解特征矩阵 \mathbf{Q} 的过程。定义 $\mathbf{P}_{I_j} \in \mathbb{R}^{n_{q_j} \times k}$ 是 \mathbf{P} 的一个子矩阵, \mathbf{P}_{I_j} 的每一行都是 I_j 项在 \mathbf{P} 的对应一行。定义 $\mathbf{R}(I_j, j) \in \mathbb{R}^{n_{q_j} \times 1}$,

其每一项都是 I_j 项在 \mathbf{R} 的第 j 列的对应项。定义 $\mathbf{E} \in \mathbb{R}^{n_{q_j} \times n_{q_j}}$ 是一个单位矩阵。我们不难推出

$$\begin{aligned} & (\mathbf{P}_{I_j}^T \mathbf{P}_{I_j} + \lambda n_{q_j} \mathbf{E}) \mathbf{q}_j^T = \mathbf{P}_{I_j} \mathbf{R}(I_j, j), \quad \forall j; \\ & \Rightarrow \mathbf{q}_j = \left(\mathbf{A}_j^{-1} \mathbf{V}_j \right)^T. \end{aligned} \quad (15.28)$$

其中， $\mathbf{A}_j = \mathbf{P}_{I_j}^T \mathbf{P}_{I_j} + \lambda n_{p_j} \mathbf{E}, \mathbf{V}_j = \mathbf{P}_{I_j} \mathbf{R}(I_j, j)$ 。由此，我们可以推出整个特征矩阵 \mathbf{Q} 。将上述两个过程带入 ALS 算法的对应步骤，经过迭代即可实现一个简单的推荐模型的构建。

通过 ALS 算法，我们对表15.4计算得出 2016 级甲同学的机器学习预测值，从而可以根据值的高低对推荐课程进行排序，从而进行后续的推荐！

15.4 实际应用

在上面的小节中，我们介绍了不同类别的推荐算法，并用每种算法给出了课程选择的推荐。但在其中，我们也遗留了一个待回答的问题：新用户加入时，应该如何进行推荐？这就是冷启动问题。在本节中，我们会介绍一些实际应用场景下的推荐系统需要考虑的问题。

当我们在驾驶汽车时，通常在车正式开启之前需要有热车阶段，这个过程就是冷启动过程。在推荐系统中，冷启动问题 (Cold start) 是指当新用户加入、新物品加入或者推荐系统刚刚运行，缺少足够的数据时，如何进行推荐的问题。冷启动对于一个推荐系统是非常重要的，因为新用户最初使用 APP 阶段也是这个用户最可能卸载 APP 的时候。我们将介绍推荐物品冷启动和推荐用户冷启动这两种情况问题。

当新的商品加入系统，还没有足够的用户评分数据时，我们说这是一个推荐物品冷启动问题。由于商品在加入系统时是可以得到其对应的一些基本属性的，所以一般可以先对物品的属性进行挖掘。在获得物品的属性信息后，可以通过聚类算法对商品进行分类。例如教务系统中新加入一门网络金融的课程，通过挖掘出这门课程“互联网相关”、“金融相关”、“授课老师也教过《经济学原理》”等信息，可以通过聚类，将其推荐给具有“选修金融专业”属性的学生。

当新的用户加入后，缺少历史数据时进行的推荐问题，我们说这是一个推荐用户冷启动问题。最简单的方式是不需要用户操作，由推荐系统来决定推荐的内容，往往是通过热门排行榜进行推荐。Cremonesi[1] 在 2010 年的论文中提到热门排行推荐在一些场景下能够有较好的表现。除了热门推荐，另外一种思路想办法采集用户的信息，这种思路下有许多具体的实现方式。例如通过社交账号进行登录，可以在用户授权的情况下导入用户的社交网络数据；有的系统则在初始化时，让用户进行简单的兴趣测试，从而获得大概的用户画像等等。

15.5 前沿方向

基于内容的推荐方法利用用户已选择的项目来寻找其他类似属性的项目进行推荐，但是这种方法需要有效的特征提取，传统的浅层模型依赖于人工设计特征，其有效性及可扩展性非常有限，制约了基于内容的推荐方法的性能。协同过滤是目前应用最为广泛的推荐算法但是同时也遭遇到了严重的数据稀疏（一个用户评分过的项目仅仅占总项目数量的极少部分），采用浅层模型无法学习到用户和项目的深层次特征。

近年来，深度学习在图像处理、自然语言理解和语音识别等领域取得了突破性进展，已经成为人工智能的一个热潮，为推荐系统的研究带来了新的机遇。深度学习能够直接从内容中提取特征，表征能力强；容易对噪声数据进行处理，抗噪能量强；可以使用 RNN 循环神经网络对动态或者序列数据进行建模；可以更加准确的学习 user 和 item 的特征；深度学习便于对负责数据进行统一处理。接下来这一部分我们列举几个深度学习在推荐系统研究中的应用，让大家对深度学习在推荐系统中的应用有一个初步了解。

15.5.1 基于 AE 的协同过滤

文章 [10] 提出的 Bayesian SDAE 模型以 Item 的 Side information 为输入来学习 Item 的隐向量。该模型假设 SDAE 中的参数满足高斯分布，同时假设 User 的隐向量也满足高斯分布，进而利用概率矩阵分解来拟合原始评分矩阵。该模型通过最大后验估计 (MAP) 得到其要优化的目标函数，进而利用梯度下降学习模型参数，从而得到 User 与 Item 对应的隐向量矩阵。

15.5.2 基于 RNN 的新闻推荐

Okura 等人 [5] 采用循环神经网络的方法研究了新闻推荐问题。首先为了抓住文章的语义信息，采用降噪自编码器 (DAE) 从新闻中提取文章的隐表示；然后为了学习用户的偏好，采用 RNN 从用户的历史行为列表中学习用户的隐表示；最后，为了利用用户与新闻之间的关联，基于新闻和用户的隐表示采用点乘的方式为用户产生新闻推荐列表。

15.5.3 基于 Transformer 的电商推荐

近日，阿里巴巴搜索推荐事业部发布了一项新研究，首次使用强大的 Transformer 模型捕获用户行为序列的序列信号，供电子商务场景的推荐系统使用。该模型 (BST) 已经部署在淘宝线上，实验结果表明，与两个基准线对比，在线点击率 (CTR) 均有显著提高。模型框架如图15.4。

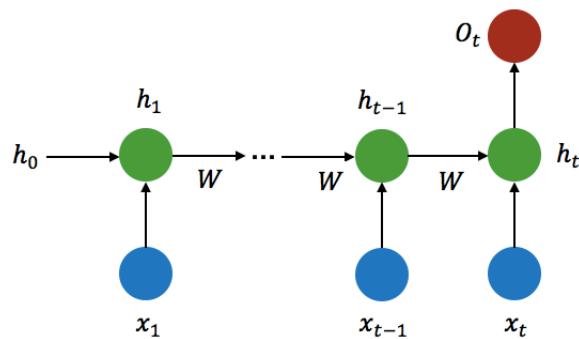


图 15.3: 循环神经网络 RNN 结构图

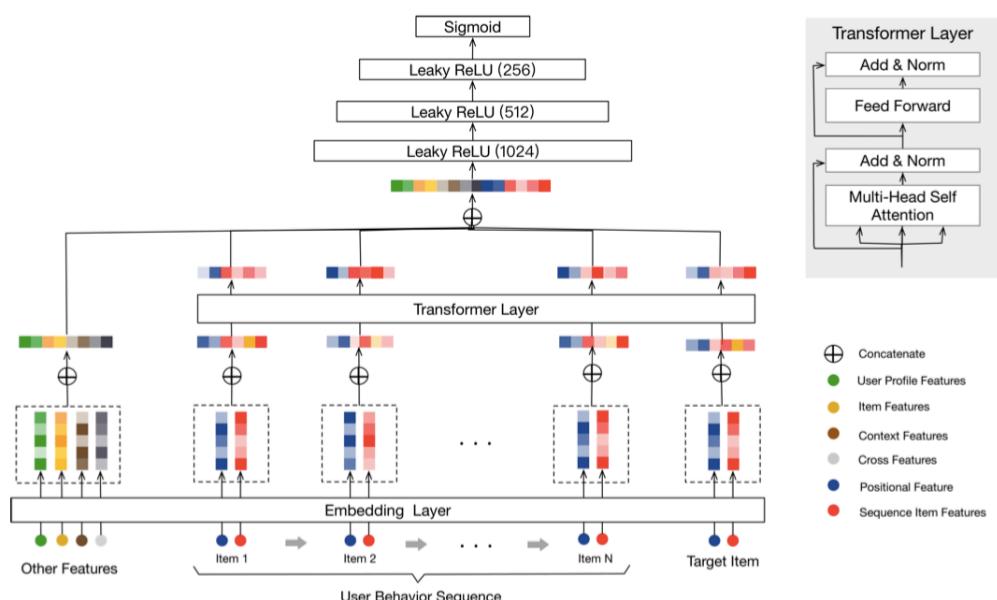


图 15.4: BST 的网络架构

参考文献

- [1] Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pages 39–46. ACM, 2010.
- [2] Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391–407, 1990.
- [3] Gene H Golub and Christian Reinsch. Singular value decomposition and least squares solutions. In *Linear Algebra*, pages 134–151. Springer, 1971.
- [4] Daniel Lemire and Anna Maclachlan. Slope one predictors for online rating-based collaborative filtering. In *Proceedings of the 2005 SIAM International Conference on Data Mining*, pages 471–475. SIAM, 2005.
- [5] Shumpei Okura, Yukihiko Tagami, Shingo Ono, and Akira Tajima. Embedding-based news recommendation for millions of users. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1933–1942. ACM, 2017.
- [6] Iacovou Neophytos Suchak Mitesh Bergstrom Peter Riedl Resnick, Paul and John. GroupLens: an open architecture for collaborative filtering of netnews. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, pages 175–186. ACM, 1994.
- [7] Paul Resnick and Hal R Varian. Recommender systems. *Communications of the ACM*, 40(3):56–59, 1997.
- [8] Alexander Tuzhilin. Towards the next generation of recommender systems. In *Proceedings of the 1st International Conference on E-Business Intelligence (ICEBI2010)*,. Atlantis Press, 2010.
- [9] Aaron Van den Oord, Sander Dieleman, and Benjamin Schrauwen. Deep content-based music recommendation. In *Advances in neural information processing systems*, pages 2643–2651, 2013.
- [10] Hao Wang, Naiyan Wang, and Dit Yan Yeung. Collaborative deep learning for recommender systems. 2014.
- [11] Yunhong Zhou, Dennis Wilkinson, Robert Schreiber, and Rong Pan. Large-scale parallel collaborative filtering for the netflix prize. In *International conference on algorithmic applications in management*, pages 337–348. Springer, 2008.
- [12] 王国霞, 刘贺平, et al. 个性化推荐系统综述. *计算机工程与应用*, 48(7), 2012.