绪论

什么是机器学习？

人可以通过经验学习，比方说“朝霞不出门，晚霞行千里”，就是通过经验得来的知识。获得知识后，即使在不同的地点，不同的时间，看到不同的霞，我们也能作出正确的判断。那么，机器是否也能学习并利用经验，从而对一些未出现过的情况，在不通过显式编程（人作出判断并告诉机器）的情况下也能作出正确的预测呢？答案是可以的，这就是**机器学习**。

对于机器来说，**经验**是通过**数据**传达的。机器学习的主要研究内容就是从数据中产生**模型**的算法，也即**学习算法**。Mitchell给出一个更为形式化的定义，假设我们用P来表示程序处理任务T时的性能，如果程序通过利用经验E提高了在任务T上的性能，则称该程序对E进行了学习。

在本书中，**模型**泛指所有从数据中学得的结果，在别的文献中，也有对**模型**和**模式**作出区分的，模型指学得的全局性结果（比如一棵决策树），模式指学得的局部性结果（比如一条规则）。

基本概念

要进行机器学习，首先要有数据，我们可以收集一组结构相同的记录，这组记录的集合就称为**数据集**。比如下面这个西瓜数据集：

编号	色泽	根蒂	敲声
001	青绿	蜷缩	浊响
002	乌黑	稍蜷	沉闷
003	浅白	硬挺	清脆

注：实际使用数据时往往需要先进行编码，即把文本改为数值（比如青绿=1，乌黑=2，等等），以便计算机进行处理。

接下来给出一些基本概念的定义：

示例(instance)，样本(sample)

> 数据集中的每条记录是对一个事件或对象（比如这里的西瓜）的描述，也称作示例或者样本。特别地，有时会把整个数据集称为一个样本，因为数据集可以看作是从样本空间中抽样所得。这时候就需要根据上下文信息来进行判断了。

属性(attribute)，特征(feature)，属性值(attribute value)

> 对象具备一些性质，并由此可以进行区分，这些性质就称为属性或者特征，比方说表格中的色泽、根蒂和敲声。不同对象在这些属性上会有不同的取值，这个取值就称为属性值。

属性空间(attribute space)，样本空间(sample space)，输入空间，特征向量(feature vector)

> 由属性张成的空间，比方说上面的表格中有3个属性，那就可以张成一个3维空间，每个样本都可以用空间中的一个点来表示，这个点对应于一个坐标向量，所以有时也把一个样本称为一个特征向量。

维数(dimensionality)

> 即数据集中每个样本拥有的特征数目。

学习(learning)，训练(training)，训练样本(training sample)，训练示例(training instance)

> 从数据中获的模型的过程。在这个过程中使用的数据称为训练数据，里面的每个样本称为一个训练样本，也称训练示例或训练例。训练样本的集合就是训练集。

模型(model)，学习器(learner)，假设(hypothesis)，真相(ground-truth)

> 模型有时也称为学习器，可以看作一组参数的有序集合，能够把属性空间映射到输出空间上。每一个模型对应于一个假设，也即数据存在的某种规律。真相指的是真正存在的规律，学习就是为了接近真相。

如果我们希望通过机器学习来实现预测(prediction)，那么只有样本是不够的，要让机器明白怎样的样本会产生怎样的结果，还需要为每个样本设置标记，标记有可能是离散值（分类任务），也可能是连续值（回归任务）。带标记的数据集如下：

编号	色泽	根蒂	敲声	标记
001	青绿	蜷缩	浊响	好瓜

编号	色泽	根蒂	敲声	标记
002	乌黑	稍蜷	沉闷	坏瓜
003	浅白	硬挺	清脆	坏瓜

标记(label)，样例(example)，标记空间(label space)，输出空间

> 标记指示的是对象的类别或者事件的结果，样本和标记组合起来就是样例。所有标记的集合称为标记空间，也称为输出空间。

分类(classification)，回归(regression)，聚类(clustering)

> 根据预测值的不同，可以把任务分为几种不同的类别。若预测的是离散值，比如“好瓜”，“坏瓜”，则该任务称为分类任务；若预测的是连续值，比如瓜的重量，则该任务称为回归任务；还有一种聚类任务，旨在基于某种度量将样本分为若干个簇(cluster)，使得同一簇内尽量相似，不同簇间尽量相异。聚类任务不需要对样本进行标记。
>
> 特别地，只涉及两种类别的分类任务称为二分类任务(binary classification)，通常称一个类为正类(positive class)，另一个类为反类或者负类(negative class)。涉及到多个类别的分类任务就称为多分类(multi-class classification)任务。

测试(testing)，测试样本(testing sample)，测试示例(testing instance)

> 训练完成后，使用模型预测新样本的标记这个过程称为测试。测试中使用到的样本称为测试样本，也称测试示例或测试例。

监督学习(supervised learning)，无监督学习(unsupervised learning)

> 根据训练样本是否标记可以把任务分为两大类，需要标记的是监督学习，不需要标记的是无监督学习。前者的代表是回归和分类，后者的代表是聚类。

泛化(generalization)能力

> 让机器进行学习的目标并不仅仅是为了让模型能在训练数据上有好的表现，我们更希望模型在新的样本上也能有良好的表现。模型适用于新样本的能力就称为泛化能力，泛化能力好的模型能够更好地适用于整个样本空间。

独立同分布(independent and identically distributed)

> 一般来说，训练数据只占训练空间很少的一部分，当我们希望这一小部分的采样能够很好地反映整个样本空间的情况，从而令学得的模型具有良好的泛化能力。通常假设样本空间中的所有样本都服从于一个未知的分布(distribution)，并且训练样本都是从该分布上独立采样所得，这就称为独立同分布。训练样本越多，越能反映该分布的特性，从而能学得泛化能力更强的模型。

假设空间

类似由样本构成的样本空间和由标记构成的标记空间，所有的假设共同构成了假设空间。学习的过程就可以看作是在假设空间中搜索最能**匹配(fit)**训练集的假设。

假设空间的规模有多大呢？举个例子，样本空间维度是3，也即每个样本由3个特征来表示。这三个属性可能的取值分别为3，2，2。那么假设空间的规模就是 `4 × 3 × 3 + 1 = 37`，为什么呢？因为除了可能的取值之外，每个属性还有一种选择就是使用通配符*号表示，即无论取何值都可以。所以一个有3种可能取值的属性在排列组合时有4种情形。最后的加一表示的是∅**假设**，它对应的是一种无论属性取何值都不可能达到目的的状况。比方说，前面的假设都是假设怎样的瓜会是好瓜，而∅假设则对应于好瓜根本不存在。

有时候会出现多个假设都能匹配训练集的情形，这些假设的集合就称为**版本空间(version space)**。版本空间是假设空间的子空间。

归纳偏好

对于学习算法来说，要产生一个模型就意味着要从版本空间中挑选出一个假设。但是版本空间中的假设都和训练集一致，无法直接分辨出哪一个更好，这时候**归纳偏好**，简称**偏好**就起作用了。

注意区分**偏好**和**特征选择**，前者是基于某种领域知识而产生的，后者则是基于对训练样本的分析进行的。

偏好指的是，在多个假设等效时，学习算法会认为某一种假设更优，并选择这种假设来建立最终的模型。如果一个学习算法没有偏好，比方说每次随机地从版本空间中选择一个假设，则它所给出的结果是时好时坏，自相矛盾的。所以任何一个有效的学习算法都应当有自己的归纳偏好。

怎样确定归纳偏好呢？一个常用的原则是**奥卡姆剃刀(Occam's razor)**，用一句话概述就是：**若多个假设与观察一致，则选最简单的那个**。注意，奥卡姆剃刀并不是唯一的准则，并且如何诠释“最简单”也是待考虑的。

给定基于某种归纳偏好的算法产生的模型A和基于另一种归纳偏好的算法产生的模型B，有时我们会注意到，A和B在不同的样本集上的表现各有好坏，有时候A的效果更好，有时候B的效果更好。甚至一个随机产生的模型都有可能在某个样本集上表现得优于我们精心设计的算法所产生的模型。怎样去定位这个问题呢？

书中使用**NFL定理(No Free Lunch Theorem)**来解答了这个问题，有一个关键点就是**总误差与学习算法无关**，证明在书上有，这里主要解析一下书中想要表达的思路。当我们考虑样本空间的所有可能分布，并认为它们都以相同的概率出现或同等重要时，无论使用什么模型，造成的总误差都是相同的，与学习算法无关！

但是！现实中并不是这样的，我们只考虑样本空间服从同一种分布的情形。打个比方，模型A是精心设计的算法产生的，模型B则简单地设定为把任何样本预测为负类。那么对于按照样本分布抽样所得的测试集，模型A效果会比B好；但是如果只抽取负类样本作为测试集，则模型B优于A。显然这种情况下模型B没有任何意义，因为我们根本就不care全是负类这种分布！

所以说，若考虑所有潜在的问题，则所有算法都一样好。**要谈论算法的优劣，必须结合具体问题！**学习算法的归纳偏好和问题是否相配，往往起到决定性的作用。

其它

关于发展历程和应用状况，属于阅读材料，所以这里不作详细的笔记了。这个小节再补充一点其他内容。

归纳（induciotn）和**演绎（deduction）**是科学推理的两大基本手段，前者是从特殊到一般的**泛化（generalization）**，后者是从一般到特殊的**特化（specialization）**。举个例子：

演绎

```
大前提：人都会死
小前提：苏格拉底是人
结论：   苏格拉底会死
```

演绎是一个层层递进的过程，条件A指出一个可能，条件B指出另一种可能，互相交叉，构成大小前提，一直往下，从而推出新的可能。数学公理系统中，新的定理就是通过一组公理的演绎得出的。

归纳

```
事件1：苏格拉底死了
事件2：柏拉图死了
事件3：阿基米德死了
结论：  人终有一死
```

归纳是从一组平等的事物中发现普遍规律/联系的过程。机器学习中从样例学习就属于归纳推理。

注：对于提到的人名，我很抱歉T.T

归纳学习

归纳学习有广义和狭义之分。广义上，只要是从样例中学习就都属于归纳学习；狭义上，不仅要从样例中学习，还要求学得**概念（concept）**，因此也称为**概念学习**。概念学习技术目前应用和研究都比较少，因为要学得泛化性能好且语义明确的概念太难了。现在的技术大多都是产生**黑箱模型**，难以明白学得的是什么，只知道它确实有用。

简要带过以下归纳学习（广义）的几大主流技术，最早期的主流是**符号主义学习**，例如决策树，能直接模拟人类对概念进行判定的流程，有良好的解释性很强大的表示能力。但是表示能力太强也直接导致了假设空间太大，复杂度极高的缺陷。问题规模较大时难以进行有效的学习。

接下来发展的另一主流技术是连接主义学习，例如神经网络。与符号主义学习能产生明确的概念不同，连接主义产生的是“黑箱”模型。连接主义学习涉及到大量的参数设置，而**参数设置缺乏理论指导，只能依靠手动调参**，参数的设置是差之毫厘谬以千里，参数的一点点差别体现在结果上可能是极其巨大的。因此，试错性大大地限制了连接主义学习的发展。

再往后，统计学习技术就闪亮登场了，例如支持向量机。它与连接主义学习有密切的联系，但没有连接主义学习那么大的局限性。

而现在最火的深度学习技术，其实就属于连接主义学习技术，可以简单地理解为多层神经网络。深度学习的模型复杂度非常高，以至于使用者只要下功夫调参，性能往往就会很好。为什么深度学习会突然变得热门呢？有两大支持，一是**数据支持**，深度学习模型参数极多，如果数据不够，很容易会过拟合，但这是一个大数据时代，海量的数据允许我们构造出精准的模型；二是**计算支持**，随着计算机技术的发展，现代计算机具备了足够的计算海量数据的能力。

习题

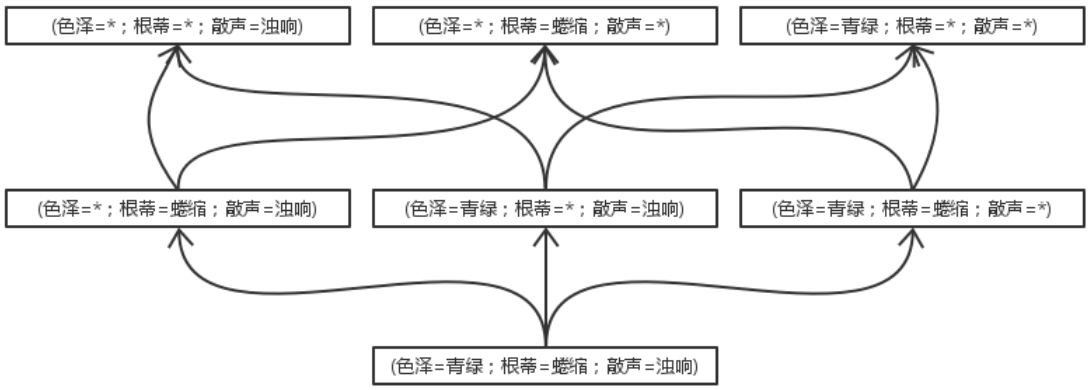
1.1

> 问：表1.1中若只包含编号为1和4的两个样例，试给出相应的版本空间。

此时训练集如下：

编号	色泽	根蒂	敲声	好瓜
1	青绿	蜷缩	浊响	是
4	乌黑	稍蜷	沉闷	否

由于好瓜的3个属性值与坏瓜都不同。所以能与训练集一致的假设就有很多可能了，从最具体的入手，往上逐层抽象可得：



1.2

> 问：与使用单个合取式来进行假设表示相比，使用“析合范式”将使得假设空间具有更强的表示能力 。例如：

$$\begin{aligned} \text{好瓜} \quad \leftrightarrow \quad & \big((\text{色泽}=*) \wedge (\text{根蒂}=\text{蜷缩}) \wedge (\text{敲声}=*)\big) \\ & \vee \big((\text{色泽}=\text{乌黑}) \wedge (\text{根蒂}=*) \wedge (\text{敲声}=\text{沉闷})\big) \end{aligned}$$

> 会把 “$(\text{色泽}=*) \wedge (\text{根蒂}=\text{蜷缩}) \wedge (\text{敲声}=*)$” 以及 “$(\text{色泽}=\text{乌黑}) \wedge (\text{根蒂}=*) \wedge (\text{敲声}=\text{沉闷})$” 都分类为好瓜。若使用最多包含 k 个合取式的析合范式来表达表1.1西瓜分类问题的假设空间，试估算有多少种可能的假设。

首先要理解题意，“使用最多包含 k 个合取式的析合范式来表达表1.1西瓜分类问题的假设空间” 这句话表达的意思是**每个假设可以由最少1个最多k个合取范式的析取来表示**。

表1.1：

编号	色泽	根蒂	敲声	好瓜
1	青绿	蜷缩	浊响	是
2	乌黑	蜷缩	浊响	是
3	青绿	硬挺	清脆	否
4	乌黑	稍蜷	沉闷	否

这里色泽有2种取值，根蒂有3种取值，敲声有3种取值。因为每个属性还可以用通配符表示取任何值都行，所以实际上这三个属性分别有3，4，4种选择。因此，在只考虑单个合取式的情况下，有 $3 \times 4 \times 4 = 48$ 种假设（因为训练集中有存在正例，所以 $\varnothing$ 假设不需考虑）。

现在我们考虑题目的条件，这实际上是一个组合问题。我们可以从48个基本假设中任意去1个到k个组合为新的假设：

使用1个合取式：$\binom{48}{1} = 48$ 种假设；

使用2个合取式：$\binom{48}{2} = \frac{48*47}{2*1} = 1128$ 种假设；

...

使用k个合取式：$\binom{48}{k} = \frac{48*47*...*(48-k+1)}{k!}$ 种假设；

把以上求得的各情形下的假设个数进行求和就得到问题的答案了。

1.3

> 问：若数据包含噪声，则假设空间中有可能不存在与所有训练样本都一致的假设。在此情形下，试设计一种归纳偏好用于假设选择。

归纳偏好是在无法断定哪一个假设更好的情况下使用的。既然问题是存在噪声，那么如果能知道噪声的分布（例如高斯噪声），就可以将这些性能相同的假设对应的误差减去由噪声引起的部分，此时再使用奥卡姆剃刀原则或者多释原则来进行假设选择就好了。更常见的做法是引入**正则化（regularization）**项，在建立模型时避免拟合噪声。

1.4*

> 问：本章1.4节在论述“没有免费的午餐”定理时，默认使用了“分类错误率”作为性能度量来对分类器进行评估。若换用其他性能度量 ℓ，则式(1.1)将改为

$$E_{ote}(\mathfrak{L}_a|X,f)=\sum_h\sum_{x\in\chi-X}P(x)\ell(h(x),f(x))P(h|X,\mathfrak{L})$$

> 试证明“没有免费的午餐定理”仍成立。

回顾NFL的证明可以发现，关键是要证明，在考虑所有可能的目标函数（对应所有可能的问题，或者说样本空间的所有分布情况）时，模型的性能变得与学习算法无关。

也即证明 $\sum_f \ell(h(x),f(x))$ 最终可以化简为与常数形式。

当我们考虑所有可能的**f**，并且**f**均匀分布时，任务就失去了优化目标，无论使用哪一种算法，所得模型的平均性能会变得相同。

如果想看更严谨的推导可以看Wolpert, D.H., Macready, W.G当初写的论文"No Free Lunch Theorems for Optimization"。

事实上我对这个定理存有疑问，**NFL定理是否适用于所有性能度量？**。比方说，推荐系统中使用到的覆盖率，可以简单理解为生成的推荐结果包含的商品数与训练集中出现的商品数的比率。对于这样一个性能度量，倾向于推荐更多商品的算法会明显比其他算法更优。并且，即使考虑所有可能的**f**，我们依然对这些算法存在偏好。

1.5

> 问：试述机器学习能在互联网搜索的哪些环节起什么作用。

这题比较开放，机器学习能起到的作用有很多，例如利用机器学习知识进行数据挖掘，从应用的角度来说，可以构建推荐系统、赛事预测、语音识别等等。

模型评估与选择

误差

在分类任务中，通常把错分的样本数占样本总数的比例称为**错误率（error rate）**。比如m个样本有a个预测错了，错误率就是 `a/m`；与错误率相对的有**精度（accuracy）**，或者说正确率，数值上等于1-错误率。

更一般地，通常会把模型输出和真实值之间的差异称为**误差（error）**。在训练集上的误差称为**训练误差（training error）**或者**经验误差（empirical error）**。而在新样本上的误差则称为**泛化误差（generalization error）**。我们希望模型的泛化误差尽可能小，但现实是，我们无法知道新样本是怎样的，所以只能尽可能地利用训练数据来最小化经验误差。

但是否经验误差小，泛化误差就一定小呢？这不是一定的，如果模型相比训练数据来说过于复杂，那就很有可能把训练数据本身的一些特点当作整个样本空间的特点，从而使得在训练数据上有很小的经验误差，但一旦面对新样本就会有很大误差，这种情况叫做**过拟合（overfitting）**。相对的是**欠拟合（underfitting）**。

欠拟合很容易避免，只要适当地增加模型复杂度（比方说增加神经网络的层数）就好。但**过拟合是无法彻底避免的**，只能缓解（减少模型复杂度/增加训练数据），这也是机器学习发展中的一个关键阻碍。

在现实任务中，要处理一个问题，我们往往有多种算法可以选择，即使是同一个算法也需要进行参数的选择，这就是机器学习中的**模型选择（model selection）**问题。既然泛化误差无法使用，而经验误差又存在着过拟合问题，不适合作为标准，那么我们应该如何进行模型选择呢？针对这个问题，后面的三个小节会给出回答。

这里先简单归纳一下，书中将模型选择问题拆解为（1）评估方法；（2）性能度量；（3）比较检验；三个子问题。可以这样理解：

- **评估方法**：用什么数据做评估？如何获得这些数据？
- **性能度量**：评估时如何衡量模型的好坏？有哪些评价标准？
- **比较检验**：如何比较模型的性能？注意不是简单地比大小！在机器学习中性能比较是相当复杂的。

评估方法

前面已经提到了不能把经验误差用作模型评估，否则会存在过拟合的嫌疑。那么很自然地，我们就会想到是否有一种方法能近似泛化误差呢？答案是有的，就是使用**测试集（testing set）**进行评估，利用**测试误差（testing error）**来近似泛化误差。

测试集和训练集一样，从样本空间中独立同分布采样而得，并且应尽可能与训练集互斥，也即用于训练的样本不应再出现在测试集中，否则就会高估模型的性能。为什么呢？举个例子，老师布置了2道题做课后作业，如果考试还是出这2两题，只能证明大家记住了这2道题；只有出不一样的题，才能看出大家是否真的掌握了知识，具备了举一反三的能力。

注意！！测试数据更多地是指模型在实际使用中遇到的数据，为了和模型评估中使用的测试集进行区分，一般会把模型评估用的测试集叫做**验证集（validation set）**。举个例子，在Kaggle或者天池上参加比赛，我们一般会拿到一份带标记的原始数据集和一份不带标记的测试数据集。我们需要选用一种评估方法来把原始数据集划分成训练集和验证集，然后进行训练，并按照模型在验证集上的性能表现来进行选择。最后挑出最好的模型对测试集的样本进行预测，并提交预测结果。下文将介绍几种常用的评估方法。

留出法

直接将数据集划分为两个互斥集合，注意保持数据分布的一致性（比如比例相似）。保留类别比例的采样方式又叫**分层采样（stratified sampling）**。举个例子，原始数据集有100个样本，假设训练集占70个，验证集占30个。若训练集中正例反例各35个，也即比例为 `1:1` ，那么验证集中就应该正例反例个15个，同样保持 `1:1` 的比例。当然，这个比例最好还是遵循原始数据集中数据的分布规律。

单独一次留出法的结果往往不可靠，一般是进行多次随机划分，然后取各次评估的平均值作为评估结果。

留出法最大的缺点就是要进行划分，当训练集占的比例较大时，模型可以更准确地刻画原始数据集的特征，但是因为验证集较小，评估的结果往往不稳定也不准确；当训练集占的比例较小时，训练出的模型又不能充分学习到原始数据集的特征，评估结果可信度不高。这个问题没有完美的解决方案，一般取数据集2/3~4/5的样本作为训练集，余下的作为验证集。

交叉验证

又称为**k折交叉验证（k-fold cross validation）**，将数据集划分为k个互斥子集。每次使用k-1个子集的并集作为训练集，余下的一个子集作为验证集，这就构成了k组训练/验证集，从而可以进行k次训练和验证。最终取k次验证的均值作为评估结果。 常用的k值包括5，10，20。

类似于留出法，因为存在多种划分k个子集的方式，为了减少因不同的样本划分而引入的差别，需要进行多次k折交叉验证。例如10次10折交叉验证，指的是进行了总计100次训练和100次评估。

特别地，令k=数据集样本数的交叉验证称为**留一法（Leave-One-Out，简称LOO）**，即有多少样本就进行多少次训练/验证，并且每次只留下一个样本做验证。这样做的好处是不需要担心随即样本划分带来的误差，因为这样的划分是唯一的。一般来说，留一法的评估结果被认为是比较准确的。但是！当数据集较大时，使用留一法需要训练的模型太多了！这种计算开销是难以忍受的！

自助法

在留出法和交叉验证法中，我们都需要对数据集进行划分，从而使得训练所用的数据集比源数据集小，引入了一些因规模不同而造成的偏差，有没有办法避免规模不同造成的影响呢？

自助法（bootstrapping）正是我们需要的答案，以**自助采样（bootstrap sampling）**为基础，对包含m个样本的源数据集进行有放回的m次采样以获得同等规模的训练集。在这m次采样中都不被抽到的概率大约为0.368，也即源数据集中有大约1/3的样本是训练集中没有的。因此，我们可以采用这部分样本作为验证集，所得的结果称为**包外估计（out-of-bag estimate）**。

注意，自助法适用于数据集小，难以划分训练/验证集的情况。因为自助法能产生多个不同训练集，所以对集成学习也大有好处。但是！**自助法改变了数据集的分布**，也因此引入了一些额外的误差。因此，数据量足的时候还是留出法和交叉验证法用得多一些。

调参和最终模型

调参（parameter tuning）一般先选定一个范围和变化步长，比如(0,1]，步长0.2，这样就有五个参数候选值。然后进行评估，选出最好的一个。这样选出的未必是全局最优的参数，但为了在开销和性能之间折中，只能这么做，毕竟我们无法试尽参数的所有取值。而且多个参数组合的情况是指数上升的，比方说有3个参数，每个参数评估5种取值，就需要测试多达 5^3 种情形。

特别注意，训练/验证这个过程是为了让我们**确定学习算法和算法的参数**，确定了这些之后，我们需要再利用整个源数据集进行训练，这次训练所得的模型才是最终模型，也即提交给用户，进行测试的模型。

性能度量

性能度量（performance measure）指的是用于衡量模型泛化能力的评价标准。使用不同的性能度量往往导致不同的评判结果。比方说搭建推荐系统，两个模型中一个精度高，一个覆盖度高，如果我们想让更多的商品得到推荐可以就会选后一个模型。所以说，模型的好坏是相对的，取决于我们采用什么性能度量，而**采用什么性能度量则应取决于我们的任务需求**。

这个小节主要介绍分类任务中常用的性能度量。

错误率和精度

在本章的开头已经提及到了，不再累述，这两个性能度量可写作更一般的形式，基于数据分布和概率密度函数进行定义。

查准率，查全率，F1

假设我们正常处理一个二分类问题，按照模型预测值和真实值可以把测试样本划分为四种情形：**真正例（true positive）**，**假正例（false positive）**，**真反例（true negative）**，**假反例（false negative）**。可以把结果表示为下图这个矩阵——**混淆矩阵(confusion matrix)**。

真实情况	预测结果	
	正例	反例
正例	TP（真正例）	FN（假反例）
反例	FP（假正例）	TN（真反例）

查准率，又称准确率（precision），用于衡量模型避免错误的能力，分母是模型预测的正例数目。

$$precision = \frac{TP}{TP + FP}$$

查全率，又称召回率（recall），用于衡量模型避免缺漏的能力，分母是测试样本真正包含的正例数目。

一般来说，这两者是矛盾的，提高其中一者则另一者必然会有所降低。

$$recall = \frac{TP}{TP + FN}$$

F1，是查准率和查全率的调和平均，用于综合考虑这两个性能度量。

$$\frac{1}{F1} = \frac{1}{2} \times (\frac{1}{precision} + \frac{1}{recall}) \Rightarrow F1 = \frac{2 \times precision \times recall}{presion + recall}$$

有时候我们对查准率，查全率的需求是不同的。比方说广告推荐，要尽量避免打扰用户，因此查准率更重要；而逃犯检索，因为漏检的危害很大，所以查全率更重要。这时就需要使用F_β了。

F_β，是查准率和查全率的加权调和平均，用于综合考虑这两个性能度量，并采用不同的权重。

$$\frac{1}{F_\beta} = \frac{1}{1 + \beta^2} \times (\frac{1}{precision} + \frac{\beta^2}{recall}) \Rightarrow F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{(\beta^2 \times presion) + recall}$$

其中 $\beta > 0$ 度量了查全率对查准率的相对重要性，等于1时F_β退化为F1，小于1时查准率更重要，大于1时查全率更重要。

书中还介绍了如何对多次训练/测试产生的多个混淆矩阵进行评估，包括宏方法（先分别计算性能度量，再计算均值）和微方法（先对混淆矩阵各元素计算均值，再基于均值计算性能度量）两种途径。

ROC与AUC

很多时候，使用模型对测试样本进行预测得到的是一个实值或者概率（比如神经网络），需要进一步设置**阈值（threshold）**，然后把预测值和阈值进行比较才能获得最终预测的标记。

我们可以按照预测值对所有测试样本进行排序，最可能是正例的排前面，最不能是正例的排后面。这样分类时就像是在这个序列中以某个**截断点（cut point）**把样本分成两部分。我们需要**根据任务需求来设置截断点**。比如广告推荐更重视查准率，可能就会把截断点设置得更靠前。

因此！**排序本身的质量很能体现出一个模型的泛化性能**，ROC曲线就是一个用来衡量排序质量的工具。

ROC，全称受试者工作特征（Receiver Operating Characteristic）。怎样画ROC曲线呢？先定义两个重要的计算量：**真正例率（True Positive Rate，简称TPR）**和**假正例率（False Positive Rate，简称FPR）**。

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{TP + FN}$$

TPR其实就等于召回率。在绘制ROC曲线时，纵轴为TPR，横轴为FPR。首先按预测值对样本进行排序，然后按序逐个把样本预测为正例，并计算此时的TPR和FPR，然后在图上画出该点，并与前一个点连线。如下图：

有两个值得注意的特例：

- 经过 (0,1) 点的曲线，这代表所有正例都在反例之前出现（否则会先出现假正例从而无法经过 (0,1) 点），这是一个**理想模型**，我们可以设置一个阈值，完美地分割开正例和反例。
- 对角线，这对应于**随机猜测**模型，可以理解为真正例和假正例轮换出现，即每预测对一次接下来就预测错一次，可以看作是随机猜测的结果。

若一个模型的ROC曲线完全包住了另一个模型的ROC曲线，我们就认为这个模型更优。但是如果两条曲线发生交叉，要怎么判断呢？比较合理的判据是**AUC（Area Under ROC Curve）**，即ROC曲线下的面积。

$$AUC = \frac{1}{2}\sum_{i=1}^{m-1}(x_{i+1}-x_i)\cdot(y_i+y_{i+1})$$

补充一点，ROC曲线上的面积等于**排序损失（loss）**。也即有：

$$AUC = 1-\ell_{rank}$$

代价敏感错误率与代价曲线

现实任务中，有时会遇到不同类型错误造成后果不同的状况。比如医生误诊，把患者诊断为健康人的影响远大于把健康人诊断为患者，因为可能因为这次误诊丧失了最佳治疗时机。为了权衡不同类型错误带来的不同损失，可以为这些错误类型赋以**非均等代价（unequal cost）**。

还是举二分类为类，可以**根据任务的领域知识**来设定一个**代价矩阵（cost matrix）**：

真实类别	预测类别	
	第0类	第1类
第0类	0	$cost_{01}$
第1类	$cost_{10}$	0

预测值与真实值相等时，自然错误代价为0。但把第0类错预测为第1类和把第1类错预测为第0类这两种错误的代价是不同的。注意，**重要的不是代价在数值上的大小，而是它们的比值**。比方说 $\frac{cost_{01}}{cost_{10}}>1$， 这就说明把第0类错预测为第1类的代价更高。

使用了非均等代价之后，我们在使用性能度量时自然也需要作出相应的改变，比方说**代价敏感（cost-sensitive）**版本的错误率：

$$E(f;D;cost)=\frac{1}{m}\Big(\sum_{x_i\in D^+}\mathbb{I}(f(x_i)\neq y_i)\times cost_{01}+\sum_{x_i\in D^-}\mathbb{I}(f(x_i)\neq y_i)\times cost_{10}\Big)$$

由于ROC曲线不能反应使用非均等代价之后的期望总体代价，所以改用**代价曲线（cost curve）**来取替。

代价曲线图的纵轴为归一化代价（将代价映射到 [0,1] 区间），横轴为正例概率代价。画法类似于ROC曲线，它是将ROC曲线的每一个点转为图中的一条线。依次计算出ROC曲线每个点对应的FPR和FNR，然后把点 (0,FPR) 和点 (0,FNR) 连线。最终所得的图中，所有线的下界所围成的面积就是该模型的期望总体代价。

比较检验

看起来似乎有了获取测试集*的评估方法和用于比较模型的性能度量之后，就能够通过不同模型在测试集上的性能表现来判断优劣了。但是！事实上，在机器学习中，模型比较并不是这样简单的比大小，而是要考虑更多。

注：指验证集，但无论是书中还是论文中，都使用测试集较多，明白两者的区别就可以了。

在模型比较中，主要有以下三个重要考虑：

1. 测试集上的性能只是泛化性能的近似，未必相同；
2. 测试集的选择对测试性能有很大影响，即使规模一致，但测试样例不同，结果也不同；
3. 一些机器学习算法有随机性，即便算法参数相同，在同一测试集上跑多次，结果也可能不同；

那么应该如何有效地进行模型比较呢？答案是采用**假设检验（hypothesis test）**。基于假设检验的结果，我们可以推断出，若在测试集上观察到模型A优于B，则是否A的泛化性能在统计意义上也优于B，以及做这个结论的把握有多大。

本小节首先介绍最基本的二项检验和t检验，然后再深入介绍其他几种比较检验方法。默认以错误率作为性能度量。

几个基础概念：

- **置信度**：表示有多大的把握认为假设是正确的。
- **显著度**：也称“显著性水平”，表示假设出错的概率。显著度越大，假设被拒绝的可能性越大。
- **自由度**：不被限制的样本数，也可以理解为能自由取值的样本数，记为 v 或 df。

单个模型、单个数据集上的泛化性能检验

> 我们有多大把握相信对一个模型泛化性能的假设？

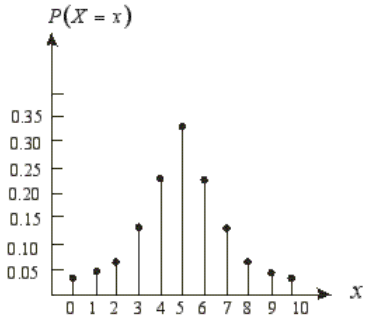
二项检验

在进行比较检验前，完成了一次模型预测，已知测试错误率为 $\hat{\epsilon}$。

一个泛化错误率为 ϵ 的模型在 m 个样本上预测错 m' 个样本的概率为：

$$P(\hat{\epsilon};\epsilon)=\binom{m}{m'}\epsilon^{m'}(1-\epsilon)^{m-m'}$$

这个概率符合二项分布:



又因为已知测试错误率为 $\hat{\epsilon}$，也即知道了该模型在 m 个样本上实际预测错 了$\hat{\epsilon}\times m$ 个样本。代入公式，对 ϵ 求偏导会发现，给定这些条件时，$\epsilon=\hat{\epsilon}$ **的概率是最大的**。

使用**二项检验（binomial test）**，假设泛化错误率 $\epsilon\leq\epsilon_0$，并且设定置信度为 $1-\alpha$。则可以这样定义错误率的阈值 $\bar{\epsilon}$：

$$\bar{\epsilon}=\max\epsilon\qquad s.t.\qquad\sum_{i=\epsilon_0\times m+1}^{m}\binom{m}{i}\epsilon^{i}(1-\epsilon)^{m-i}<\alpha$$

其中 $s.t.$ 表示左式在右边条件满足时成立。右式计算的是发生不符合假设的事件的总概率，如果我们要有 $1-\alpha$ 的把握认为假设成立，那么发生不符合假设的事件的总概率就必须低过 α。

在满足右式的所有 ϵ 中，选择最大的作为阈值 $\bar{\epsilon}$。如果在测试集中观测到的测试错误率 $\hat{\epsilon}$ 是小于阈值 $\bar{\epsilon}$的， 我们就能以$1-\alpha$ 的把握认为假设成立，即该模型的泛化误差 $\epsilon\leq\epsilon_0$。

t检验

二项检验只用于检验某一次测试的性能度量，但实际任务中我们会进行多次的训练/测试，得到多个测试错误率，比方说进行了k次测试，得到 $\hat{\epsilon}_1,\hat{\epsilon}_2,\ldots,\hat{\epsilon}_k$。这次就会用到**t检验(t-test)**。

定义这 k 次测试的平均错误率 μ 和方差 σ^2：

$$\mu=\frac{1}{k}\sum_{i=1}^{k}\hat{\epsilon}_i$$

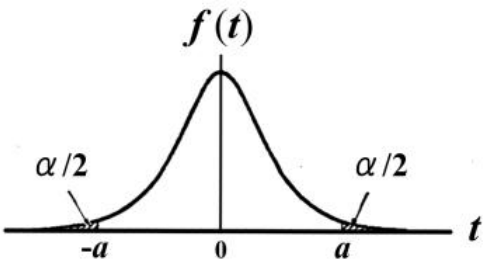
$$\sigma^2=\frac{1}{k-1}\sum_{i=1}^{k}(\hat{\epsilon}_i-\mu)^2$$

注意！这里使用的是**无偏估计的样本方差**，分母是 $k-1$，因为当均值确定，并且已知 $k-1$ 个样本的值时，第 k 个样本的值是可以算出来的，也可以说是**受限的**。

假设泛化错误率 $\epsilon=\epsilon_0$，并且设定显著度为 α。计算统计量t：

$$t=\frac{\sqrt{k}(\mu-\epsilon_0)}{\sigma}$$

该统计量服从自由度 $v=k-1$ 的t分布，如下图：



自由度越大，约接近于正态分布，自由度为无穷大时变为标准正态分布（$\mu=0$，$\sigma=1$）。

如果计算出的t统计量落在临界值范围 $[t_{-a/2},t_{a/2}]$ 之内（注：临界值由自由度 k 和显著度 α 决定，通过查表得出），我们就能以 $1-\alpha$ 的把握认为假设成立，即该模型的泛化误差 $\epsilon=\epsilon_0$。

两个模型/算法、单个数据集上的泛化性能检验

> 我们有多大把握相信两个模型的泛化性能无显著差别？

交叉验证t检验

对两个模型A和B，各使用k折交叉验证分别得到k个测试错误率，即 $\hat{\epsilon}_1^A,\hat{\epsilon}_2^A,\dots,\hat{\epsilon}_k^A$ 和 $\hat{\epsilon}_1^B,\hat{\epsilon}_2^B,\dots,\hat{\epsilon}_k^B$。使用**k折交叉验证成对t检验**（**paired t-tests**）来进行比较检验。

对于这两组k个测试错误率，计算两组之间的每一对的差，即 $\triangle_i=\hat{\epsilon}_k^A-\hat{\epsilon}_k^B$，从而得到k个 $\triangle$。我们可以计算 $\triangle$ 的均值 μ 和方差 σ^2，定义统计量t：

$$t=|\frac{\sqrt{k}\mu}{\sigma}|$$

可以看到，和前面的t检验相比，这里的分子没有被减项，其实是省略了。因为我们假设两个模型的泛化错误率相同，实际上是假设 $|\epsilon^A-\epsilon^B|=0$，这个 0 被省略了。

类似地，这个统计量服从自由度 $v=k-1$ 的t分布。我们设定好显著度 α，查表获取临界值范围，如果计算出的t统计量落在在范围内，就能以 $1-\alpha$ 的把握认为假设成立，即两个模型的泛化性能无显著差别，否则认为平均测试错误率较低的模型更胜一筹。

McNemar检验

对于一个二分类问题，如果使用留出法，我们不仅可以获得两个算法A和B各自的测试错误率，或能够获得它们分类结果的差别（都预测正确、都预测错误、一个预测正确一个预测错误），构成一张**列联表（contingency table）**：

算法B	算法A	
	分类正确	分类错误
分类正确	e_{00}	e_{01}
分类错误	e_{10}	e_{11}

假设两个算法的泛化性能无显著区别，则 e_{01} 应该等于 e_{10}，变量 $|e_{01}-e_{10}|$ 应服从均值为 1，方差为 $e_{01}+e_{10}$ 的正态分布，可以计算统计量 χ^2：

$$\chi^2=\frac{(|e_{01}-e_{10}|-1)^2}{e_{01}+e_{10}}$$

该变量服从自由度为 $v=1$ 的 χ^2 分布（卡方分布），类似t检验，设定好显著度 α，按照自由度和显著度查表获得临界值。若计算所得的统计量 χ^2 小于临界值，则能以 $1-\alpha$ 的把握认为假设成立，即两个算法的泛化性能无显著差别，否则认为平均测试错误率较低的算法更胜一筹。

注：这里 v 为1是因为只有2个算法

多个模型/算法、多个数据集上的泛化性能检验

> 我们有多大把握相信多个模型的泛化性能皆无显著差别？若有，接下来怎样做？

在一组数据集上进行多个算法的比较，情况就变得较复杂了，一种做法是使用前面的方法分开两两比较；另一种更直接的做法是使用基于算法排序的Friedman检验。

Friedman检验

假设有 $N=4$ 个数据集，$k=3$ 种算法，可以使用一种评估方法，获得各个算法在各个数据集上的测试结果，然后按照性能度量由好到坏进行排序，序值为1，2，3。若并列，则取序值的平均值。然后对各个算法在各数据集上的序值求平均得到平均序值，如：

数据集	算法A	算法B	算法C
D1	1	2	3
D2	1	2.5	2.5
D3	1	2	3
D4	1	2	3
平均序值	1	2.125	2.875

令 r_i 表示第 i 个算法的平均序值，则 r_i 服从均值为 $\frac{k+1}{2}$，方差为 $\frac{(k^2)-1}{12}$ 的正态分布。可以计算统计量 χ^2：

$$\chi^2=\frac{12N}{k(k+1)}(\sum_{i=1}^{k}r_i^2-\frac{k(k+1)^2}{4})$$

在 k 和 N 都较大时(通常要求 $k>30$)，该变量服从自由度为 $v=k-1$ 的 χ^2 分布（卡方分布）。

以上这种检验方式也称为**原始Friedman检验**，被认为过于保守，现在通常用统计量 F 代替：

$$F=\frac{(N-1)\chi^2}{N(k-1)-\chi^2}$$

该变量服从于自由度为 $v=k-1$ 或 $v=(k-1)(N-1)$ 的 F 分布。

和前面的检验方式有所区别，F检验是根据设定的显著度 α 和**算法个数** k 以及 **数据集个数**N 这三者来查表的，如果计算出的统计量 F 小于查表所得的临界值，则假设成立，能以$1-\alpha$ 的把握认为认为这 k 个算法的泛化性能无显著区别。

但如果这个假设被拒绝了呢？这时就需要进行**后续检验（post-hoc test）**，常用的有 **Nemenyi后续检验**。

Nemenyi后续检验

定义**平均序值差别的临界值域**为：

$$CD=q_\alpha\sqrt{\frac{k(k+1)}{6N}}$$

其中 q_α是由 显著度 α 和**算法个数** k 确定的，通过查表获取。若两个算法的平均序值之差不超过 CD，则能以$1-\alpha$ 的把握认为这两个算法的泛化性能无显著区别，否则认为平均序值较小的更胜一筹。

Nemenyi后续检验还可以通过Friedman检验图更直观地体现出来，横轴为性能度量，纵轴为算法，每个算法用一段水平线段表示，线段中心点为该算法的平均序值，线段长度为 CD。若两个算法的线段投影到x轴上有重叠部分，则可以认为这两个算法的泛化性能无显著区别。

偏差与方差

除了估计算法的泛化性能，我们往往还希望知道**为什么有这样的性能**？这时一个有用的工具就是**偏差-方差分解（bias-variance decomposition）**。

知乎上面有两个问题都有不错的答案，不妨先看看。[1] 机器学习中的Bias(偏差)，Error(误差)，和Variance(方差)有什么区别和联系？；[2] 偏差和方差有什么区别？。

对学习算法的期望繁华错误率进行拆解，最终会发现能拆解为三个项（需要推导）：

$$E(f;D)=\mathbb{E}_D[(f(x;D)-\bar{f}(x))^2]+(\bar{f}(x)-y)^2+\mathbb{E}_D[(y_D-y)^2]$$

依次对应于**方差（variance）**、**偏差（bias）**、**噪声（noise）**：

$$E(f;D)=var(x)+bias^2(x)+\epsilon^2$$

这三者的含义是这样的：

- 方差：使用同规模的不同训练集进行训练时带来的性能变化，刻画**数据扰动带来的影响**；
- 偏差：学习算法的期望预测与真实结果的偏离程度，刻画**算法本身的拟合能力**；
- 噪声：当前任务上任何算法所能达到的期望泛化误差的下界（即不可能有算法取得更小的误差），刻画**问题本身的难度**；

也即是说，泛化性能是有学习算法的拟合能力，数据的充分性以及问题本身的难度共同决定的。给定一个任务，噪声是固定的，我们需要做得就是尽量降低偏差和方差。

但是这两者其实是有冲突的，这称为**偏差-方差窘境（bias-variance dilemma）**。给定一个任务，我们可以控制算法的训练程度（如决策树的层数）。在训练程度较低时，拟合能力较差，因此训练数据的扰动不会让性能有显著变化，此时偏差主导泛化错误率；在训练程度较高时，拟合能力很强，以至于训练数据自身的一些特性都会被拟合，从而产生过拟合问题，训练数据的轻微扰动都会令模型产生很大的变化，此时方差主导泛化错误率。

注意，将泛化性能完美地分解为方差、偏差、噪声这三项仅在**基于均方误差的回归任务**中得以推导出，分类任务由于损失函数的跳变性导致难以从理论上推导出分解形式，但已经有很多方法可以通过实验进行估计了。

习题

2.1

> 问：数据集包含1000个样本，其中500个正例，500个反例，将其划分为包含70%样本的训练集和30%样本的测试集用于留出法评估，试估算共有多少种划分方式。

按照题意，训练集包含700个样本，测试集包含300个样本。并且为了保证训练集和测试集的类别比例一致，使用分层采样，训练集包含正例反例各350个；测试集包含正例反例个150个。

有多少种划分方式是一个排列组合问题，等于从500个正例中挑选出300个的所有可能组合乘上从500个正例中挑选出300个的所有可能组合：

$$\binom{500}{350} \times \binom{500}{350} = \binom{500}{150} \times \binom{500}{150}$$

2.2

> 问：数据集包含100个样本，其中正反例各一半，假定学习算法所产生的模型是将新样本预测为训练样本数较多的类别（训练样本数相同时进行随机猜测），试给出用10折交叉验证法和留一法分别对错误率进行评估所得的结果。

（1）10折交叉验证法：通过分层采样获得10个互斥子集，每个子集包含10个样本，正反例各5个。每次取其中9个子集做训练，1个子集做测试。因为在训练集中两个类别数目相当（都为 $9*5=45$ 个），所以只能进行随机猜测，错误率为50%。

（2）留一法：每次取一个样本做测试，若取出的样本为正例，那么剩下的训练集中有50个反例，49个正例，因此预测结果为反例，反之亦然。故错误率为100%。

2.3

> 问：若学习器A的F1值比学习器B高，试析A的BEP值是否也比B高。

模型A的F1值高于模型B的F1值也即：

$$F1_A > F1_B \Leftrightarrow \frac{2 \times precision_A \times recall_A}{presion_A + recall_A} > \frac{2 \times precision_B \times recall_B}{presion_B + recall_B}$$

BEP是查准率precision与查全率recall相等时的取值，令式中的 $precision_A = recall_A = BEQ_A$，$precision_B = recall_B = BEQ_B$，因此有：

$$\frac{2 \times BEQ_A^2}{2 \times BEQ_A} > \frac{2 \times BEQ_B^2}{2 \times BEQ_B} \Rightarrow BEQ_A > BEQ_B$$

得证，若学习器A的F1值比学习器B高，则A的BEP值也比B高。

2.4

> 问：试述真正例率（TPR）、假正例率（FPR）与查准率（P）、查全率（R）之间的联系。

- 真正例率是真正例占真实正例的比例；
- 假正例率是假正例占真实反例的比例；
- 查准率是真正例占预测正例的比例；
- 查全率是真正例占真实正例的比例；

其中，真正例指模型预测为正例并且预测正确的样例，假正例指模型预测为正例并且预测错误的样例。前缀真实表明在数据集中的真实标记，前缀预测表明模型预测出的标记。

特别地，查全率与真正例率是相等的。

2.5

> 问：试证明 $AUC = 1 - \ell_{rank}$

先回顾一下AUC的计算方法：

$$AUC = \frac{1}{2}\sum_{i=1}^{m-1}(x_{i+1} - x_i) \cdot (y_i + y_{i+1})$$

计算方法类似于积分，把ROC曲线下的面积分解为m（样本个数）个小矩形，然后对这些小矩阵的面积求和。每个矩形的宽等于相邻的两个点横坐标的差值，每个矩形的高等于相邻两个点纵坐标之和的 $\frac{1}{2}$。上式把常数 $\frac{1}{2}$ 提到求和项外了。

再看看排序损失（指按算法预测值排序时带来的损失）：

$$\ell_{rank} = \frac{1}{m^+ \cdot m^-}\sum_{x^+ \in D^+}\sum_{x^- \in D^-}(\mathbb{I}(f(x^+) < f(x^-)) + \frac{1}{2}\mathbb{I}(f(x^+) = f(x^-)))$$

其中 m^+ 和 m^- 分别表示测试集中的正例集和反例集。

这题目前还没有头绪，AUC曲线上的面积是：

$$\begin{aligned}&\frac{1}{2}\sum_{i=1}^{m-1}(x_{i+1} - x_i) \cdot ((1 - y_i) + (1 - y_{i+1}))\\ &= -\frac{1}{2}\sum_{i=1}^{m-1}(x_{i+1} - x_i) \cdot (y_i + y_{i+1} - 2)\end{aligned}$$

怎么证明这个表达式和 ℓ_{rank} 相等呢？

2.6

> 问：试述错误率与ROC曲线之间的关系

错误率 ϵ：

$$\epsilon = \frac{FP + FN}{m}$$

其中 m 为样本总数， FP 为假正例个数， FN 为假反例个数。因为给定测试集，正例反例的数目是固定的。当我们以ROC曲线上的一个点作为阈值划分预测的正例和反例时，该点的真正例率（y轴）越高，那么假反例就会越少；该点的假正例率（x轴）越低，那么假正例就越少，从而使得错误率越低。归纳起来就是，在ROC曲线上距离左上角 (0,1) 点越近的点错误率越低。

2.7

> 问：试证明任意一条ROC曲线都有一条代价曲线与之对应，反之亦然。

回顾代价曲线的作法，求ROC曲线上每一点对应的FPR和FNR，在代价平面连接点 (0,FPR) 和点 (1，FNR)，代价曲线由所有连线的下界确定。由于ROC曲线是连续的，故必然有一条对应的代价曲线。 反过来，对代价曲线上的任何一点作切线，都可以得到对应的FPR与FNR，从而计算出ROC曲线上对应点的坐标。因此代价曲线也必然有一条对应的ROC曲线。

2.8

> 问：$Min - Max$ 规范化和 $z - score$ 规范化是两种常用的规范化方法。令 x 和 x' 分别表示变量在规范化前后的取值，相应的，令 x_{min} 和 x_{max} 表示规范化前的最小值和最大值， x'_{min} 和 x'_{max} 表示规范化后的最小值和最大值，$\bar{x}$ 和 σ_x 分别表示规范化前的均值和标准差。试析二者的优缺点。

$Min - Max$ 规范化：

$$x' = x'_{min} + \frac{x - x_{min}}{x_{max} - x_{min}} \times (x'_{max} - x'_{min})$$

$z - score$ 规范化：

$$x' = \frac{x - \bar{x}}{\sigma_x}$$

$Min - Max$ 规范化比较简单，缺点在于当有新数据输入时，可能导致max和min的变化，需要重新定义。

$z - score$ 规范化能把数据的原分布转换为标准正态分布，即均值为0，标准差为1，对于某些任务可能会有帮助。比如分类时，规范化后取值靠近-1的认为是负类，取值靠近1的认为是正类。注意！它并非是把数据映射到 [-1,1] 区间，会有超出的部分（回想正态分布曲线两段的延伸段），实际处理时一般把超出部分都修改为-1和1以满足映射到区间内的要求。这种规范化的缺点就是改变了数据的分布，这可能会引入某种误差。

2.9

> 问：试述卡方检验过程。

根据概率论与数理统计中的内容（交大版本，P239）。卡方检验适用于方差的检验。步骤如下：

1）分均值已知与均值未知两种情况，求得卡方检验统计量；

2）根据备选假设以及α，求得所选假设对应的拒绝域（临界值区间）；

3）根据1）中求得的卡方统计量与2）中求得的拒绝域，判断假设成立与否；

2.10*

> 问：试述原始Friedman检验和F检验的区别。

暂时没有思路。

线性模型

给定一个包含d个属性的实例 $\mathbf{x}=(x_1;x_2;\ldots;x_d)$，**线性模型（linear model）**的原理是学得一个可以通过属性的线性组合来进行预测的函数，也即：

$$f(\mathbf{x})=w_1x_1+w_2x_2+\ldots+w_dx_x+b$$

一般写作向量形式：$f(\mathbf{x})=\mathbf{w}^T\mathbf{x}+b$。其中权重向量 $\mathbf{w}$ 和偏置项 b 就是我们需要学习的参数。

线性模型有良好的可解释性，每个属性对应的权重可以理解为它对预测的重要性。并且建模较为简单，许多功能更为强大的非线性模型都是在线性模型的基础上引入层级结构或高维映射得到的。

这一章的内容大致如下：

- **线性回归**：如何把离散属性连续化？怎样用最小二乘法进行参数估计？多元线性回归如何求解？广义线性模型是怎样的？。
- **对数几率回归**：分类任务和线性回归是如何关联起来的？从概率的角度来看，如何用极大似然法进行参数估计并获取最优解？
- **线性判别分析**：二分类任务如何求得LDA模型的参数？如何推广到多分类任务？
- **多分类学习**：如何把多分类任务拆分为二分类任务？有哪些拆分策略？是如何进行建模和预测的？
- **类别不平衡问题**：再缩放思想以及三种解决类别不平衡问题的主要方法。

线性回归

离散属性连续化

由于不同模型对数据的要求不一样，在建模之前，我们需要对数据做相应的处理。一般的线性回归模型要求属性的数据类型为连续值，故需要对离散属性进行连续化。

具体分两种情况：

1. 属性值之间有序：也即属性值有明确的大小关系，比方说把三值属性 “高度” 的取值 {高，中，低} 转换（编码）为 {1.0，0.5，0.0}；
2. 属性值之间无序：若该属性有 k 个属性值，则把它转换为 k 维向量（把1个属性扩展为k个属性），比方说把无序离散属性 “商品” 的取值 {牙膏，牙刷，毛巾} 转换为 (0,0,1)，(0,1,0)，(1,0,0)。 这种做法在自然语言处理和推荐系统实现中很常见，属性 “单词” 和 “商品” 都是无序离散变量，在建模前往往需要把这样的变量转换为哑变量，**否则会引入不恰当的序关系，从而影响后续处理**（比如距离的计算）。

补充：对应于离散属性连续化，自然也有**连续属性离散化**。比方说决策树建模就需要将连续属性离散化。此外，在作图观察数据分布特征时，往往也需要对连续属性进行离散化处理（比方说画直方图）。

最小二乘法

回归任务最常用的性能度量是**均方误差（mean squared error, MSE）**。首先介绍**单变量线性回归**，试想我们要在二维平面上拟合一条曲线，则每个样例（即每个点）只包含一个实值属性（x值）和一个实值输出标记（y值），此时均方误差可定义为：

$$E(f;D)=\frac{1}{m}\sum_{i=1}^{m}(y_i-f(x_i))^2$$
$$=\frac{1}{m}\sum_{i=1}^{m}(y_i-wx_i-b)^2$$

有时我们会把这样描述模型总误差的式子称为**损失函数**或者**目标函数**（当该式是优化目标的时候）。这个函数的自变量是模型的参数 w 和 b。由于给定训练集时，样本数 m 是一个确定值，也即常数，所以可以把 $\frac{1}{m}$ 这一项拿走。

最小二乘法（least square method）就是基于均方误差最小化来进行模型求解的一种方法，寻找可使损失函数值最小的参数 w 和 b 的过程称为最小二乘**参数估计（parameter estimation）**。

通过对损失函数分别求参数 w 和 b 的偏导，并且令导数为0，可以得到这两个参数的**闭式（closed-form）解**（也即**解析解**）：

$$w=\frac{\sum_{i=1}^{m}y_i(x_i-\bar{x})}{\sum_{i=1}^{m}x_i^2-\frac{1}{m}\left(\sum_{i=1}^{m}x_i\right)^2}$$

$$b=\frac{1}{m}\sum_{i=1}^{m}(y_i-wx_i)$$

在实际任务中，只要我们把自变量（x, y, m）的值代入就可以求出数值解了。

为什么可以这样求解呢？因为损失函数是一个**凸函数**（记住是向下凸，类似U型曲线），导数为0表示该函数曲线最低的一点，此时对应的参数值就是能使均方误差最小的参数值。特别地，**要判断一个函数是否凸函数，可以求其二阶导数**，若二阶导数在区间上非负则称其为凸函数，若在区间上恒大于零则称其为**严格凸函数**。

多元线性回归

前面是直线拟合，样例只有一个属性。对于样例包含多个属性的情况，我们就要用到**多元线性回归（multivariate linear regression）**（又称作多变量线性回归）了。

令 $\hat{\mathbf{w}}=(\mathbf{w};b)$。把数据集表示为 $m\times(d+1)$ 大小的矩阵，每一行对应一个样例，前 d 列是样例的 d 个属性，**最后一列恒置为1**，对应偏置项。把样例的实值标记也写作向量形式，记作 $\mathbf{y}$。则此时损失函数为：

$$E_{\hat{\mathbf{w}}}=(\mathbf{y}-X\hat{\mathbf{w}})^T(\mathbf{y}-X\hat{\mathbf{w}})$$

同样使用最小二乘法进行参数估计，首先对 $\hat{\mathbf{w}}$ 求导：

$$\frac{\partial E_{\hat{\mathbf{w}}}}{\partial\hat{\mathbf{w}}}=2X^T(X\hat{\mathbf{w}}-\mathbf{y})$$

令该式值为0可得到 $\hat{\mathbf{w}}$ 的闭式解：

$$\hat{\mathbf{w}}*=(X^TX)^{-1}X^T\mathbf{y}$$

这就要求 X^TX 必须是可逆矩阵，也即必须是**满秩矩阵（full-rank matrix）**，这是线性代数方面的知识，书中并未展开讨论。但是！**现实任务中 X^TX 往往不是满秩的**，很多时候 X 的列数很多，甚至超出行数（例如推荐系统，商品数是远远超出用户数的），此时 X^TX 显然不满秩，会解出多个 $\hat{\mathbf{w}}$。这些解都能使得均方误差最小化，这时就需要由学习算法的**归纳偏好**决定了，常见的做法是引入**正则化（regularization）**项。

广义线性模型

除了直接让模型预测值逼近实值标记 y，我们还可以让它逼近 y 的衍生物，这就是**广义线性模型（generalized linear model）**的思想，也即：

$$y=g^{-1}(\mathbf{w^Tx}+b)$$

其中 $g(\cdot)$ 称为**联系函数（link function）**，要求单调可微。使用广义线性模型我们可以实现强大的**非线性函数映射**功能。比方说**对数线性回归（log-linear regression）**，令 $g(\cdot)=ln(\cdot)$，此时模型预测值对应的是**实值标记在指数尺度上的变化**：

$$\ln y=\mathbf{w^Tx}+b$$

对数几率回归（逻辑回归）

前面说的是线性模型在回归学习方面的应用，这节开始就是讨论分类学习了。

线性模型的输出是一个实值，而分类任务的标记是离散值，怎么把这两者联系起来呢？其实广义线性模型已经给了我们答案，我们要做的就是**找到一个单调可微的联系函数**，把两者联系起来。

对于一个二分类任务，比较理想的联系函数是**单位阶跃函数（unit-step function）**：

$$y=\begin{cases}0 & \text{z}<0;\\ 0.5 & \text{z}=0;\\ 1 & \text{z}>0,\end{cases}$$

但是单位阶跃函数不连续，所以不能直接用作联系函数。这时思路转换为**如何近似单位阶跃函数**呢？**对数几率函数（logistic function）**正是我们所需要的（注意这里的 y 依然是实值）：

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

对数几率函数有时也称为对率函数，是一种**Sigmoid函数**（即形似S的函数）。将它作为 $g^{-}(\cdot)$ 代入广义线性模型可得：

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

该式可以改写为：

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

其中，$\frac{y}{1-y}$ 称作**几率（odds）**，我们可以把 y 理解为该样本是正例的概率，把 $1-y$ 理解为该样本是反例的概率，而几率表示的就是**该样本作为正例的相对可能性**。若几率大于1，则表明该样本更可能是正例。对几率取对数就得到**对数几率（log odds，也称为logit）**。几率大于1时，对数几率是正数。

由此可以看出，对数几率回归的实质使用线性回归模型的预测值逼近分类任务真实标记的对数几率。它有几个优点：

1. 直接对分类的概率建模，无需实现假设数据分布，从而避免了假设分布不准确带来的问题；
2. 不仅可预测出类别，还能得到该预测的概率，这对一些利用概率辅助决策的任务很有用；
3. 对数几率函数是任意阶可导的凸函数，有许多数值优化算法都可以求出最优解。

最大似然估计

有了预测函数之后，我们需要关心的就是怎样求取模型参数了。这里介绍一种与最小二乘法异曲同工的办法，叫做**极大似然法（maximum likelihood method）**。我在另一个项目中有这方面比较详细的讲解，欢迎前往项目主页交流学习。

前面说道可以把 y 理解为一个样本是正例的概率，把 $1-y$ 理解为一个样本是反例的概率。而所谓极大似然，就是最大化预测事件发生的概率，也即**最大化所有样本的预测概率之积**。令 $p(c=1|\mathbf{x})$ 和 $p(c=0|\mathbf{x})$ 分别代表 y 和 $1-y$。（注：书中写的是 $y=1$ 和 $y=0$，这里为了和前面的 y 区别开来，我用了 c 来代表标记）。简单变换一下公式，可以得到：

$$p(c=1|\mathbf{x})=\frac{e^{(}\mathbf{w^T}\mathbf{x}+b)}{1+e^{\mathbf{w^T}\mathbf{x}+b}}$$

$$p(c=0|\mathbf{x})=\frac{1}{1+e^{\mathbf{w^T}\mathbf{x}+b}}$$

但是！由于预测概率都是小于1的，如果直接对所有样本的预测概率求积，所得的数会非常非常小，当样例数较多时，会超出精度限制。所以，一般来说会对概率去对数，得到**对数似然（log-likelihood）**，此时**求所有样本的预测概率之积就变成了求所有样本的对数似然之和**。对率回归模型的目标就是最大化对数似然，对应的似然函数是：

$$\ell(\mathbf{w},b)=\sum_{i=1}^{m}\ln p(c_i|\mathbf{x_i};\mathbf{w};b)$$

$$=\sum_{i=1}^{m}\ln(c_i p_1(\hat{\mathbf{x_i}};\beta)+(1-c_i)p_0(\hat{\mathbf{x_i}};\beta))$$

可以理解为若标记为正例，则加上预测为正例的概率，否则加上预测为反例的概率。其中 $\beta=(\mathbf{w};b)$。

对该式求导，令导数为0可以求出参数的最优解。特别地，我们会发现似然函数的导数和损失函数是等价的，所以说**最大似然解等价于最小二乘解**。最大化似然函数等价于最小化损失函数：

$$E(\beta)=\sum_{i=1}^{m}(-y_i\beta^T\hat{x_i}+\ln(1+e^{\beta^T\hat{\mathbf{x_i}}}))$$

这是一个关于 β 的高阶可导连续凸函数，可以用最小二乘求（要求矩阵的逆，计算开销较大），也可以用数值优化算法如**梯度下降法（gradient descent method）**、**牛顿法（Newton method）**等逐步迭代来求最优解（可能陷入局部最优解）。

线性判别分析（LDA）

二分类

线性判别分析（**Linear Discriminant Analysis**，简称**LDA**），同样是利用线性模型，LDA提供一种不同的思路。在LDA中，我们不再是拟合数据分布的曲线，而是**将所有的数据点投影到一条直线上，使得同类点的投影尽可能近，不同类点的投影尽可能远**。二分类LDA最早有Fisher提出，因此也称为**Fisher判别分析**。

具体来说，投影值 $y = \mathbf{w}^T\mathbf{x}$，我们不再用 y 逼近样例的真实标记，而是希望同类样例的投影值尽可能相近，异类样例的投影值尽可能远离。如何实现呢？首先，同类样例的投影值尽可能相近意味着**同类样例投影值的协方差应尽可能小**；然后，异类样例的投影值尽可能远离意味着**异类样例投影值的中心应尽可能大**。合起来，就等价于最大化：

$$
\begin{aligned}
J &= \frac{\|\mathbf{w}^T\mu_0 - \mathbf{w}^T\mu_1\|_2^2}{\mathbf{w}^T\Sigma_0\mathbf{w} + \mathbf{w}^T\Sigma_1\mathbf{w}} \\
&= \frac{\mathbf{w}^T(\mu_0 - \mu_1)(\mu_0 - \mu_1)^T\mathbf{w}}{\mathbf{w}^T(\Sigma_0 + \Sigma_1)\mathbf{w}}
\end{aligned}
$$

其中，分子的 μ_i 表示第i类样例的**均值向量**（即表示为向量形式后对各维求均值所得的向量）。分子表示的是两类样例的均值向量投影点（也即类中心）之差的 ℓ_2 范数的平方，这个值越大越好。 分母中的 Σ_i 表示第i类样例的**协方差矩阵**。分母表示两类样例投影后的协方差之和，这个值越小越好。

定义**类内散度矩阵**（**within-class scatter matrix**）：

$$
\begin{aligned}
S_w &= \sigma_0 + \sigma_1 \\
&= \sum_{x \in X_0}(\mathbf{x} - \mu_0)(\mathbf{x} - \mu_0)^T + \sum_{x \in X_1}(\mathbf{x} - \mu_1)(\mathbf{x} - \mu_1)^T
\end{aligned}
$$

定义**类间散度矩阵**（**between-class scatter matrix**）：

$$
S_b = (\mu_0 - \mu_1)(\mu_0 - \mu_1)^T
$$

这两个矩阵的规模都是 $d \times d$，其中 d 是样例的维度（属性数目）。于是可以重写目标函数为：

$$
J = \frac{\mathbf{w}^T S_b \mathbf{w}}{\mathbf{w}^T S_w \mathbf{w}}
$$

也即 S_b 和 S_w 的**广义瑞利熵**（**generalized Rayleigh quotient**）。

可以注意到，分子和分母中 w 都是二次项，因此，**最优解与 w 的大小无关，只与方向有关**。

令分母为1，用拉格朗日乘子法把约束转换为方程，再稍加变换我们便可以得出：

$$
\mathbf{w} = S_w^{-1}(\mu_0 - \mu_1)
$$

但一般不直接对矩阵 S_w 求逆，而是采用**奇异值分解**的方式。

多分类

多分类LDA与二分类不同在于，学习的是一个规模为 $d \times d'$ 的投影矩阵 $\mathbf{W}$，而不是规模为 $d \times 1$ 的投影向量 $\mathbf{w}$。这个投影矩阵把样本投影到 d' 维空间（或者说 d' 维超平面）上，由于 d' 通常远小于样例原来的属性数目 d，且投影过程用到了类别信息（标记值），所以LDA也常常被视为一种**监督降维技术**。（注：d' 最大可取为类别数-1）

多分类学习

有些二分类学习方法（如LDA）可以直接推广到多分类，但现实中我们更多地是**基于一些策略，把多分类任务分解为多个二分类任务**，利用二分类模型来解决问题。有三种最经典的拆分策略，分别是一对一，一对其余，和多对多。

一对一

一对一（**One vs. One，简称OvO**）的意思是把所有类别两两配对。假设样例有N个类别，OvO会产生 $\frac{N(N-1)}{2}$ 个子任务，**每个子任务只使用两个类别的样例**，并产生一个对应的二分类模型。测试时，新样本输入到这些模型，产生 $\frac{N(N-1)}{2}$ 个分类结果，最终预测的标记由投票产生，也即把被预测得最多的类别作为该样本的类别。

一对其余

一对其余（**One vs. Rest，简称OvR**）在有的文献中也称为一对所有（**One vs. All，简称OvA**），但这种说法并不严谨。因为OvR产生 N 个子任务，**每个任务都使用完整数据集**，把一个类的样例当作正例，其他类的样例当作反例，所以应该是一对其余而非一对所有。OvR产生 N 个二分类模型，测试时，新样本输入到这些模型，产生 N 个分类结果，若只有一个模型预测为正例，则对应的类别就是该样本的类别；若有多个模型预测为正例，则选择置信度最大的类别（参考模型评估与选择中的**比较检验**）。

OvO和OvR各有优劣：OvO需要训练的分类器较多，因此**OvO的存储开销和测试时间开销通常比OvR更大**；OvR训练时要用到所有样例，因此**OvR的训练时间开销通常比OvO更大**。测试性能取决于具体的数据分布，大多情况下这两个拆分策略都差不多。

多对多

多对多（**Many vs. Many，简称MvM**）是每次将多个类作为正例，其他的多个类作为反例。OvO和OvR都是MvM的特例。书中介绍的是一种比较常用的MvM技术——**纠错输出码（Error Correcting Outputs Codes，简称ECOC）**。

MvM的正反例划分不是任意的，必须有特殊的构造，否则组合起来时可能就无法定位到预测为哪一类了。ECOC的工作过程分两步：

- 编码：对应于训练。假设有N个类别，计划做M次划分，每次划分把一部分类别划为正类，一部分类别划分为反类，最终训练出M个模型。而每个类别在M次划分中，被划为正类则记作+1，被划为负类则记作-1，于是可以表示为一个M维的编码。
- 解码：对应于预测。把新样本输入M个模型，所得的M个预测结果组成一个预测编码。把这个预测编码和各个类别的编码进行比较，跟哪个类别的编码距离最近就预测为哪个类别。

类别划分由**编码矩阵（coding matrix）**指定，编码矩阵有多重形式，常见的有二元码（正类为+1，负类为-1）和三元码（多出了**停用类**，用0表示，因为有停用类的存在，**训练时可以不必使用全部类别的样例**）。举个三元码的例子：

	f1 ↓	**f2** ↓	**f3** ↓	**f4** ↓	**f5** ↓	海明距离 ↓	欧氏距离 ↓
$C_1 \rightarrow$	-1	+1	-1	+1	+1	4	4
$C_2 \rightarrow$	+1	-1	-1	+1	-1	2	2
$C_3 \rightarrow$	-1	+1	+1	-1	+1	5	$2\sqrt{5}$
$C_4 \rightarrow$	-1	-1	+1	+1	-1	3	$\sqrt{10}$
测试样本→	-1	-1	+1	-1	+1	-	-

这里一共有4个类别，对应每一行。计划做5次划分，得到f1至f5共五个二分类器，对应每一列。可以看到每一个类别有一个5位的编码表示。测试时，把新样本输入到5个模型，得到预测编码。然后计算这个预测编码和每个类别编码的距离。这里举了海明距离（不同的位数的数目）和欧氏距离作为例子。可以看到测试样本与类别2的距离最近，因此预测该样本为类别2。

特别地，为什么称这种方法为**纠错**输出码呢？因为**ECOC编码对分类器的错误有一定的容忍和修正能力**。即使预测时某个分类器预测成了错误的编码，在解码时仍然有机会产生正确的最终结果。具体来说，对同一个学习任务，**编码越长，纠错能力越强**。但是相应地也需要训练更多分类器，增大了计算和存储的开销。

对同等长度的编码来说，理论上**任意两个类别之间的编码距离越远，纠错能力越强**。但实际任务中我们一般不需要获取最优编码，一方面非最优编码已经能产生不错的效果；另一方面，**即使获得理论上最优的编码，实际性能也不一定最好**。因为机器学习还涉及其他一些方面，在划分多个类时产生的新问题难度往往也不同，有可能理论最优的编码产生的类别子集难以区分，问题难度更大，从而使得性能下降。

类别不平衡问题

类别不平衡（class-imbalance）问题非常普遍，比方说推荐系统中用户购买的商品（通常视作正例）和用户未购买的商品（通常视作反例）比例是极为悬殊的。如果直接用类别不平衡问题很严重的数据集进行训练，所得模型会严重偏向所占比例较大的类别。**本节默认正类样例较少，负类样例较多**。这里主要介绍三种做法：

欠采样

欠采样（undersampling）针对的是负类，也即移取训练集的部分反例，使得正类和负类的样例数目相当。由于丢掉了大量反例，所以**时间开销也大大减少**。但是带来一个问题就是，**随机丢弃反例可能会丢失一些重要信息**。书中提到一种解决方法是利用**集成学习机制**，将反例划分为多个集合，用于训练不同的模型，从而使得**对每个模型来说都进行了欠采样，但全局上并无丢失重要信息**。

过采样

过采样（oversampling）针对的是正类，也即增加训练集的正例，使得正类和负类的样例数目相当。过采样的**时间开销会增大很多**，因为需要引入很多正例。注意！过采样**不能简单地通过重复正例来增加正例的比例**，这样会引起严重的过拟合问题。一种较为常见的做法是对已有正例进行**插值**来产生新的正例。

阈值移动

阈值移动（threshold-moving）利用的是**再缩放**思想。回想前面对数几率回归中，几率 $\frac{y}{1-y}$ 表示正例的相对可能性，我们默认以1作为阈值，其实是假设了样本的真实分布为正例反例各一半。但这可能不是真相，假设我们有一个存在类别不平衡问题的训练集，正例数目为 m^+，反例数目为 m^-，可以重定义：

$$\frac{y'}{1-y'} = \frac{y}{1-y} \times \frac{m^-}{m^+}$$

这就是**再缩放（rescaling）**。当几率大于 $\frac{m^+}{m^-}$ 时就预测为正例。但必须注意，这种思想是**基于观测几率近似真实几率这一假设**的，现实任务中这一点未必成立。

习题

3.1

> 问：试析在什么情形下，预测函数 $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ 中不必考虑偏置项 b。

Quora上就有这个问题，而且解释得也不错。试想一下，拟合曲线时，如果不考虑偏置项，则只能拟合一条过原点的曲线。在多元线性回归中也同理，如果不考虑偏置项，那么拟合的超平面就只能过原点，但现实中数据点的分布并不是这样的。使用一个不依赖于属性的的偏置项能够让权重向量所描述的超平面更好地拟合数据点的分布。如果输出值的期望（均值）为0就不需要考虑偏置项了，或者说偏置项此时就等于0。

3.2

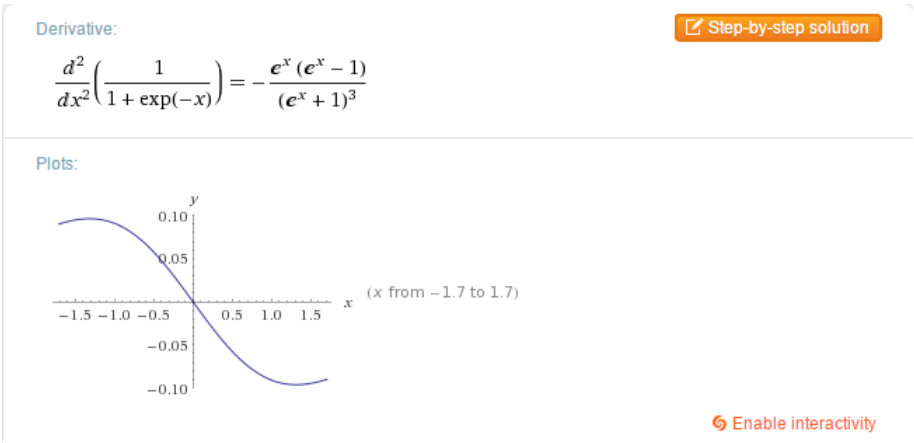
> 问：试证明，对于参数w，对率回归（logistics回归）的目标函数（式1）是非凸的，但其对数似然函数（式2）是凸的。

式1：$y = \frac{1}{1+e^{-(\mathbf{w^T}\mathbf{x}+b)}}$

式2：$E(\beta) = \sum_{i=1}^{m}(-y_i\beta^T\hat{x}_i + \ln(1+e^{\beta^T\hat{\mathbf{x}_i}}))$

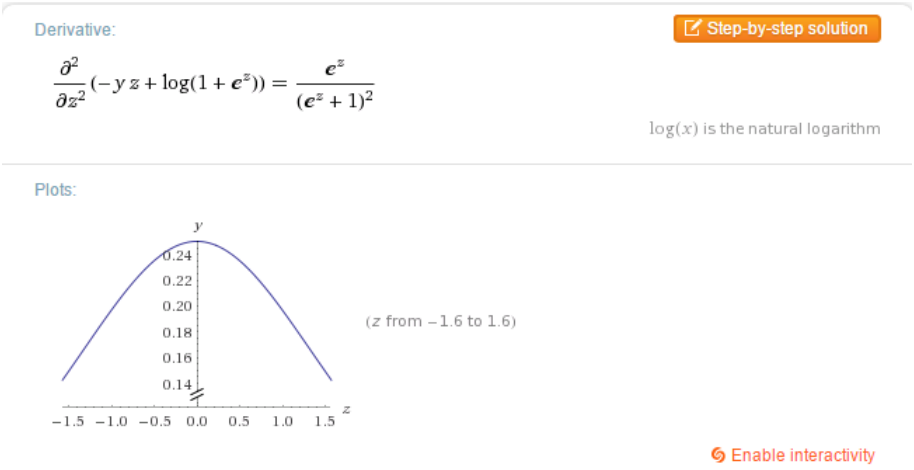
笔记中已经提到了，检验一个函数是否凸函数，可以看其二阶导数是否在区间上恒大于0.

目标函数（也即sigmoid函数）：



显然，sigmoid的二阶导数在自变量大于0处取值小于0，所以它不是凸函数。

对数似然函数：



可以看到这个函数在区间上恒大于0（两边无限延伸），符合凸函数的要求，但我不太明白的是为什么作者把这个函数仍然称为对数似然函数，对数几率回归目标是最大化对数似然，最小化损失，私以为把这个函数称为损失函数更合适。

3.3

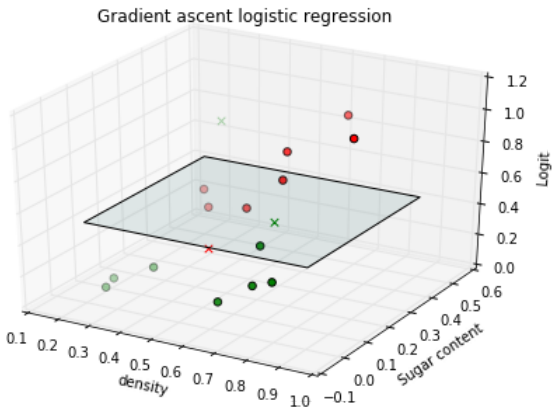
> 问：编程实现对率回归，并给出西瓜数据集3.0α上的结果

西瓜数据集3.0α：

编号	密度	含糖率	好瓜
1	0.697	0.460	是
2	0.774	0.376	是
3	0.634	0.264	是
4	0.608	0.318	是
5	0.556	0.215	是
6	0.403	0.237	是
7	0.481	0.149	是
8	0.437	0.211	是
9	0.666	0.091	否
10	0.243	0.0267	否
11	0.245	0.057	否
12	0.343	0.099	否
13	0.639	0.161	否
14	0.657	0.198	否
15	0.36	0.37	否
16	0.593	0.042	否
17	0.719	0.103	否

把这个数据集转换为csv表格，并且注意要把标记转换为**0**、**1**，注意不是**-1**、**+1**！！逻辑回归的二分类标记必须是**0**、**1**，对应于**sigmoid**函数的值域。

代码实现放在了code文件下的exercise3.3.ipynb，不过Github似乎不支持ipynb的在线预览，可以下载下来用ipython notebook打开。结果如下：



用了梯度上升法来更新权值，步长0.05，最大迭代次数2000次。上图中红色为好瓜，绿色为坏瓜，圆形标记表示预测正确，叉号标记表示预测错误。可以看到有一个好瓜被预测为坏瓜，有两个坏瓜被预测为好瓜。事实上，在200次迭代后，已经基本定型了，权值并没有太大的变化。

3.4

> 问：选择两个UCI数据集，比较10折交叉验证法和留一法所估计出的对率回归的错误率。

在UCI数据集上可以进行下载。最近时间不多，暂且挖个坑。

3.5

> 问：编程实现线性判别分析，并给出西瓜数据集3.0α上的结果

最近时间不多，暂且挖个坑。

3.6

> 问：LDA仅在线性可分数据上能获得理想结果，试设计一个改进方法，使其能较好地用于非线性可分数据

可以利用核方法，把非线性可分数据的分布转换为线性可分，书上137页有介绍：KLDA（核线性判别分析方法）。

3.7

> 问：令码长为9，类别数为4，试给出海明距离意义下理论最优的EOOC二元码并证明之。

码长为9，即要产生9个分类器，因此需要9种划分方面。类别数为4，因此1V3有4种分法，2V2有6种分法，3V1同样有4种分法。书上说理论上任意两个类别之间的距离越远，纠错能力就越强。这句话可以理解为各个类别之间的距离之和越大约好。对于1个2V2分类器，4个类别的海明距离累积为4；对于3V1与1V3分类器，海明距离均为3，所以可以认为2V2的效果更好。故最优EOOC二元码由6个2V2分类器和3个3v1或1v3分类器构成。

3.8*

> 问：EOOC编码能起到理想纠错作用的重要条件是：在每一位编码上出错的概率相当且独立。试析多分类任务经ECOC编码后产生的二类分类器满足该条件的可能性及由此产生的影响。

在每一位编码上出错的概率即指在第i个分类器上的错误率，假设每个分类器选择相同的模型与最优的参数。那么满足概率相当并且独立应该需要每个分类器的正负例比例相当，并且每个分类器划分的正负例在空间中的相对分布情况应当相近。一般情况下一般很难满足这样的条件，肯定会有错误率较高的分类器。错误率较高的分类器在较少时影响不大，但当高错误率分类器占到多数时，就会拖低整体的错误率。所以我认为在某些极端情况下，增加码长可能会降低正确率

3.9

> 问：使用OvR和MvM将多分类任务分解为二分类任务求解时，试述为何无需专门针对类别不平衡性进行处理。

因为OvR或者MvM在输出结果阶段，是对各个二分类器的结果进行汇总，汇总的这个过程就会消除不平衡带来的影响（因为总和总是1）

3.10*

> 问：试推导出多分类代价敏感学习（仅考虑基于类别的误分类代价）使用“再缩放“能获得理论最优解的条件。

最近时间不多，暂且挖个坑。

决策树

由于决策树的内容我之前有做过一个比较详细的PPT分享，所以这一章的笔记暂时不打算花太大精力，主要是理清最重要的定义和思路，更详细的之后有时间会考虑补上。

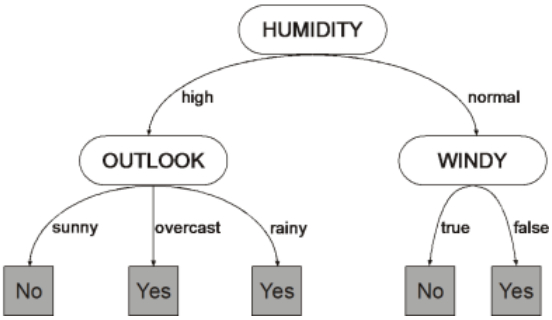
这一章的内容大致如下：

- **基本流程**：决策树是如何决策的？决策树学习的目的是什么？如何生成一颗决策树？
- **划分选择**：怎样选择最优划分属性？有哪些判断指标？具体是怎样运作的？
- **剪枝处理**：为什么要剪枝？如何判断剪枝后决策树模型的泛化性能是否提升？预剪枝和后剪枝是怎样工作的？有什么优缺点？
- **连续与缺失值**：如何把连续属性离散化？如何基于离散化后的属性进行划分？和离散属性有何不同？如何在属性值缺失的情况下选择最优划分属性？给定划分属性，如何划分缺失该属性值的样本？
- **多变量决策树**：决策树模型的分类边界的特点是怎样的？多变量决策数是如何定义的？又是如何工作的？

基本流程

决策树（decision tree）是一种模仿人类决策的学习方法。举个例子，比方说买电脑，我们首先看看外观帅不帅气，然后再看看性能怎么样，还得看看价格如何，最终**经过一系列的判断做出**是否购买电脑的**决策**。

一棵决策树可以分成三个部分：叶节点，非叶节点，分支。**叶节点**对应**决策结果**，也即分类任务中的类别标记；**非叶节点**（包括根节点）对应一个**判定问题**（某属性=？）；**分支**对应父节点**判定问题的不同答案**（可能的属性值），可能连向一个非叶节点的子节点，也可能连向叶节点。



决策就是从根节点开始走到叶节点的过程。每经过一个节点的判定，数据集就按照答案（属性值）划分为若干子集，**在子节点做判定时只需要考虑对应的数据子集就可以了**。

决策树学习的目的是为了**产生一棵泛化能力强，即处理未见示例能力强的决策树**。

决策树生成是一个**递归过程**：

生成算法：

1. 传入训练集和属性集
2. 生成一个新节点
3. 若此时数据集中所有样本都属于同一类，则把新节点设置为该类的叶节点，然后**返回**[1]。
4. 若此时属性集为空，或者数据集中所有样本在属性集余下的所有属性上取值都相同，无法进一步划分，则把新节点设置为叶节点，类标记为数据集中样本数最多的类，然后**返回**[2]
5. 从属性集中选择一个最优划分属性
    - 为该属性的每个属性值生成一个分支，并按属性值划分出子数据集
    - 若分支对应的子数据集为空，无法进一步划分，则直接把子节点设置为叶节点，类标记为父节点数据集中样本数最多的类，然后**返回**[3]
    - 将子数据集和去掉了划分属性的子属性集作为算法的传入参数，继续生成该分支的子决策树。

稍微注意以下，3处返回中的第2处和第3处设置叶节点的类标记原理有所不同。第2处将类标记设置为当前节点对应为数据集中样本数最多的类，这是利用当前节点的**后验分布**；第3处将类标记设置为为父节点数据集中样本数最多的类，这是把父节点的样本分布作为当前节点的**先验分布**。

划分选择

在决策树模型中，我们不断进行判定的初衷是希望**划分后需要考虑的可能更少**，准确地说，是希望所得子节点的**纯度（purity）**更高（也可以说是混乱程度更低）。

信息熵（information entropy）是一种衡量样本集纯度的常用指标：

$$Ent(D) = -\sum_{k=1}^{|\mathcal{Y}|} p_k log_2 p_k$$

一定要记得最前面的负号！！！其中 $|\mathcal{Y}|$ 为类别集合，p_k 为该类样本占样本总数的比例。

信息熵越大，表示样本集的混乱程度越高，纯度越低。

信息增益

信息增益（information gain）是**ID3算法**采用的选择准则，定义如下：

$$Gain(D,a) = Ent(D) - \sum_{v=1}^{V} \frac{|D^v|}{|D|} Ent(D^v)$$

它描述的是按某种属性划分后纯度的提升，**信息增益越大，代表用属性 a 进行划分所获得的纯度提升越大**。其中 V 表示属性 a 的属性值集合，D^v 表示属性值为 v 的数据子集。求和项也称为**条件熵**，我们可以理解为它是先求出每个数据子集的信息熵，然后按每个数据子集占原数据集的比例来赋予权重，比例越大，对提升纯度的帮助就越大。

多个属性都取得最大的信息增益时，任选一个即可。

信息增益又称为**互信息（Mutual information）**。

- 一个连续变量X的不确定性，用方差Var(X)来度量
- 一个离散变量X的不确定性，用熵H(X)来度量
- 两个连续变量X和Y的相关度，用协方差或相关系数来度量
- 两个离散变量X和Y的相关度，用互信息I(X;Y)来度量(直观地，X和Y的相关度越高，X对分类的作用就越大)

（信息）增益率

增益率（gain ratio）是**C4.5算法**采用的选择准则，定义如下：

$$Gain\_ratio(D,a)=\frac{Gain(D,a)}{IV(a)}$$

其中，

$$IV(a)=-\sum_{v=1}^{V}\frac{|D^v|}{|D|}log_2\frac{|D^v|}{|D|}$$

一定要记得最前面的负号！！！IV称为属性的**固有值（intrinsic value）**，它的定义和信息熵是类似的，信息熵衡量的是样本集在类别上的混乱程度，而**固有值衡量的是样本集在某个属性上的混乱程度。固有值越大，则该属性混乱程度越高，可能的取值越多。**

之所以要定义增益率是为了**避免模型过份偏好用取值多的属性作划分**。这是使用信息增益作准则非常容易陷入的误区，比方说每个样本都有一个“编号”属性，这个属性的条件熵肯定是最小的，但如果选择了该属性作为根节点，那么构建出的决策树就没有任何意义了，因为这个模型根本不具备泛化性能。

注意了，**C4.5并非直接选择增益率最高的属性**，它使用了一个**启发式**：先从属性集中找到信息增益高于平均水平的属性作为候选，然后再比较这些候选属性的增益率，从中选择增益率最高的。

基尼指数

基尼指数（Gini index）是**CART算法**采用的选择准则，定义如下：

基尼值：

$$\begin{aligned}Gini(D)&=\sum_{k=1}^{|\mathcal{Y}|}\sum_{k'\neq k}p_kp_{k'}\\&=1-\sum_{k=1}^{|\mathcal{Y}|}p_k^2\end{aligned}$$

基尼指数：

$$Gini\_index(D,a)=\sum_{v=1}^{V}\frac{|D^v|}{|D|}Gini(D^v)$$

基尼值是另一种衡量样本集纯度的指标。反映的是**从一个数据集中随机抽取两个样本，其类别标志不同的概率**。

基尼值越小，样本集的纯度越高。

由基尼值引伸开来的就是基尼指数这种准则了，**基尼指数越小，表示使用属性 a 划分后纯度的提升越大**。

剪枝处理

剪枝（pruning）是决策树学习算法应对**过拟合**的主要手段。因为决策树模型太强大了，很可能把训练集学得太好以致于把训练集本身的特性也给学习了（特别是属性数多于样本数的情况），所以去除掉一些分支是有必要的。

怎么判断剪枝有没有用呢？具体来说就是**判断剪枝后模型的泛化性能有没有提升**？这就涉及到第二章模型评估与选择的内容了。不过这里不用做比较检验，我们需要做的首先是**选定一种评估方法划分训练集和测试集**，然后**选定一种性能度量**用来衡量剪枝前后的模型在测试集上的效果。

预剪枝

预剪枝（prepruning）是在决策树生成的过程中，对每个节点在划分前先进行估计，若当前节点的划分不能带来决策树泛化性能提升（比方说，划分后在测试集上错得更多了 / **划分前后在测试集上效果相同**），就停止划分并将当前节点标记为叶节点。

后剪枝

后剪枝（postpruning）是先从训练集生成一颗完整的决策树，然后自底向上地逐个考察非叶节点，若将该节点对应的子树替换为叶节点能带来决策树泛化性能的提升，则将该子树替换为叶节点。实际任务中，即使没有提升，**只要不是性能下降，一般也会剪枝**，因为根据奥卡姆剃刀准则，简单的模型更好。

特别地，只有一层划分（即只有根节点一个非叶节点）的决策树称为**决策树桩（decision stump）**。

优缺点

预剪枝是一种贪心策略，因为它在决策树生成时就杜绝了很多分支展开的机会，所以不但**降低了过拟合的风险**，同时也**显著减少了模型的训练时间开销和测试时间开销**。但是！这种贪心策略有可能导致**欠拟合**，因为有可能当前划分不能提升模型的泛化性能，但其展开的后续划分却会显著提升泛化性能。在预剪枝中这种可能被杜绝了。

后剪枝是种比较保守的策略，**欠拟合的风险很小，泛化性能往往优于预剪枝的决策树**。但是由于后剪枝是在生成了完整决策树后，自底向上对所有非叶节点进行考察，所以**训练时间开销要比未剪枝决策树和预剪枝决策树都大得多**。

连续与缺失值

连续值

前面线性模型已经谈到了离散属性连续化，而决策树模型需要的则是**连续属性离散化**，因为决策树**每次判定只能做有限次划分**。最简单的一种离散化策略是C4.5算法采用的**二分法（bi-partition）**。

给定一个包含连续属性 a 的数据集，并且 a 在数据集中有 n 个不同取值，我们先把属性 a 的 n 个属性值**从小到大进行排序**。**所谓“二分”是指将这些属性值分为两个类别**（比方说把身高这一属性分为高于170和低于170两个类别）。

这就产生了一个新问题，怎么找到合适的划分点（例如上面例子的170）呢？

在对连续属性值排序完之后，由于有 n 个不同取值，取每**两个取值的平均值作为划分点**的话，就有 $n-1$ 个候选划分点。我们需要做得就是按照准则（比方说用ID3算法的话就是信息增益）进行 $n-1$ 次判断。每次拿出一个候选划分点，把连续属性分为两类，转换为离散属性。然后基于这个基础计算准则，最终选出一个最优的属性值划分点。

注意！和离散属性不同，连续属性用于当前节点的划分后，其**后代节点依然可以使用该连续属性进一步划分**。比方说当前节点用身高低于170划分了，那么它的后代节点还可以用身高低于160来进一步划分。

缺失值

确实值在实际任务中是非常常见的，如果直接丢弃包含缺失值的样本会造成极大的浪费。具体来说缺失值的处理分以下两个部分：

- **如何在属性值缺失的情况下选择最优划分属性？**

假设数据集为 D，有缺失值的属性为 a，令 $\tilde{D}$ 表示 D 中没有缺失属性 a 的样本子集。我们只能基于 $\tilde{D}$ 来判断属性 a 的优劣。但是我们又希望包含缺失值的样本也能在建模过程体现出一定的影响了，因此要**重新定义准则**。在那之前，先定义几个新定义用到的变量：

$$\rho = \frac{\sum_{\mathbf{x}\in\tilde{D}} w_{\mathbf{x}}}{\sum_{\mathbf{x}\in D} w_{\mathbf{x}}}$$

$$\tilde{p_k} = \frac{\sum_{\mathbf{x}\in\tilde{D}_k} w_{\mathbf{x}}}{\sum_{\mathbf{x}\in\tilde{D}} w_{\mathbf{x}}}, \quad (1 \le k \le |\mathcal{Y}|)$$

$$\tilde{r_v} = \frac{\sum_{\mathbf{x}\in\tilde{D}^v} w_{\mathbf{x}}}{\sum_{\mathbf{x}\in\tilde{D}} w_{\mathbf{x}}}, \quad (1 \le v \le V)$$

ρ 表示**无缺失值样本所占的比例**;

$\tilde{p_k}$ 表示**无缺失值样本中第 k 类所占的比例**;

$\tilde{r_v}$ 表示**无缺失值样本中在属性 a 上取值 a^v 的样本所占的比例** ;

注意，这里的 $w_{\mathbf{x}}$ 表示样本的权值，它是**含缺失值样本参与建模**的一种方式。在根节点处初始时，所有样本 $\mathbf{x}$ 的权重都为1。

接下来重新定义信息熵和信息增益，推广到样本含缺失值的情况：

$$Ent(\tilde{D}) = -\sum_{k=1}^{|\mathcal{Y}|} \tilde{p_k} log_2 \tilde{p_k}$$

$$\begin{aligned} Gain(D,a) &= \rho \times Gain(\tilde{D},a) \\ &= \rho \times (Ent(\tilde{D}) - \sum_{v=1}^{V} \tilde{r_v} Ent(\tilde{D^v})) \end{aligned}$$

按照新的定义来计算包含缺失值的属性的信息增益，然后和其他属性的信息增益相比，选出最优的。

- **给定划分属性，如何划分缺失该属性值的样本？**

假设有一个包含缺失值的属性被计算出是最优划分属性，那么我们就要按该属性的不同取值划分数据集了。缺失该属性值的样本怎么划分呢？答案是**按概率划分**，这样的样本会被**同时划入所有子节点**，并且其**权重更新**为对应的 $\tilde{r_v} \dot{w}_{\mathbf{x}}$。

可以把无缺失值的决策树建模想象为各样本权值恒为1的情形，它们**只对自己所属的属性值子集作贡献**。而样本含缺失值时，它会**以不同的概率对所有属性值子集作贡献**。

多变量决策树

前面提到的决策树都是**单变量决策树（univariate decision tree）**， 即在每个节点处做判定时都只用到一个属性。它有一个特点，就是形成的**分类边界都是轴平行（axis-parallel）的**。

如果把属性都当作坐标空间中的坐标轴，由于我们建模时假设样本的各属性之间是没有关联的，所以**各坐标轴是相互垂直的**。而决策数每次只取一个确定的属性值来划分，就等同于**画一个垂直于该属性坐标轴的超平面**（只有两个属性时就是一条线），它**与其他坐标轴都是平行的**，这就是轴平行。最终由多个与坐标轴平行的超平面组成分类边界。

这样有一个弊端就是，如果真实分类边界特别复杂，就需要画出很多超平面（线），在预测时就需要继续大量的属性测试（遍历决策树）才能得到结果，**预测时间开销很大**。

多变量决策树（multivariate decision tree），顾名思义，它不再是选择单个最优划分属性作为节点，而是试图寻找一个**最优的多属性的线性组合**作为节点，它的每个非叶节点都是一个形如 $\sum_{i=1}^{d} w_i a_i = t$ 的线性分类器。多变量决策树的决策边界能够**斜着走**，甚至**绕曲线走**，从而用更少的分支更好地逼近复杂的真实边界。

习题

4.1

> 试证明对于不含冲突数据（即特征向量完全相同但标记不同）的训练集，必存在与训练集一致（即训练误差为0）的决策树

假设不存在与训练集一致的决策树，那么训练集训练得到的决策树至少有一个节点上存在无法划分的多个数据（若节点上没有冲突数据，那么总是能够将数据分开的）。这与前提-不含冲突数据 矛盾，因此必存在与训练集一致的决策树

4.2

> 试析使用“最小训练误差”作为决策树划分选择的缺陷。

若以最小训练误差作为决策树划分的依据，由于训练集和真是情况总是会存在一定偏差，这使得这样得到的决策树会存在过拟合的情况，对于未知的数据的泛化能力较差。因此最小训练误差不适合用来作为决策树划分的依据。

4.3

> 试编程实现基于信息熵进行划分选择的决策树算法，并为表4.3中数据生成一棵决策树

最近时间不多，暂且挖个坑。

4.4

> 试编程实现基于基尼指数进行划分选择的决策树算法，并为表4.2中数据生成预剪枝、后剪枝决策树，并与未剪枝决策树进行比较。

最近时间不多，暂且挖个坑。

4.5

> 试编程实现基于对率回归进行划分选择的决策树算法，并为表4.3中数据生成一棵决策树

最近时间不多，暂且挖个坑。

4.6

> 试选择4个UCI数据集，对上述3种算法所产生的未剪枝、预剪枝、后剪枝决策树进行实验比较，并进行适当的统计显著性检验。

最近时间不多，暂且挖个坑。

4.7

> 图4.2是一个递归算法，若面临巨量数据，则决策树的层数会很深，使用递归方法易导致“栈”溢出，试使用“队列”数据结构，以参数maxDepth控制数的最大深度，写出与图4.2等价、但不使用递归的决策树生成算法。

下面算法我没有尝试写对应的代码，若有朋友写过欢迎交流/指正。

以下代码为 队列+MaxDepth控制，即广度优先搜索。其实我觉得这里如果要用MaxDepth进行控制的话，应该选择堆栈而非队列，即应该用深度优先搜索。但下面还是给出队列的形式。若要改为深度优先搜索只需要将先进后出 改成 先进先出即可（即数据存取都在一端）。

```
————————————————————————————————————————
输入：训练集 D={(x1,y1),(x2,y2),...,(xm,ym)};
      属性集 A={a1,a2,...,ad}
      最大深度 MaxDepth
过程：函数TreeGenerate(D,A,MaxDepth)
1:生成节点root
2:if D中样本全部属于同一类别C then
3:     将root标记为C类叶节点;return
4:end if
5:if A=空集 or D中样本在A上取值相同 then
6:     将root标记为叶节点，其类别标记为D中样本数最多的类：return
7:end if
8:从A中选择最优划分属性a*;
9:将root标记为分支节点，属性为属性a*;
10:将root放入NodeQueue;
11:将D放入DataQueue;
12:将A\{a*}放入AQueue;
13:初始化深度depth=1;
14:将depth放入DepthQueue;
15:while NodeQueue 非空:
16:     取出NodeQueue队尾的节点rNode，其对应的属性是ra*;
17:     取出DataQueue队尾的数据集rD;     #此处r均指队尾rear
18:     取出AQueue队尾的属性集rA;
19:     取出DepthQueue队尾的元素rdepth;
20:     if rdepth==MaxDepth:
21:          将rNode标记为叶节点，类别标记为rD中样本最多的类;
22:          continue;     #跳过本次循环，即不再对这个节点做展开
23:     for ra*的每一个取值ra*v do:
24:          为rNode生成一个分支节点，令rDv表示rD在ra*上取值为ra*v的样本子集;
25:          if rDv为空 then:
26:               将分支节点标记为叶节点，其类别标记为rD中样本最多的类;
27:          else if rD中样本全部属于同一类别C then
28:               将分支节点标记为C类叶节点;
29:          else if rA=空集 or rD中样本在A上取值相同 then
30:               将分支节点标记为叶节点，其类别标记为rD中样本数最多的类;
31:          else:
32:               从rA中选择最优划分属性a*v;
33:               将分支节点的属性记为a*v;
34:               将分支节点放入NodeQueue的队头;
35:               将rDv放入DataQueue的队头;
36:               将rA\{a*v}放入AQueue的队头;
37:               将(rDepth+1)放入DepthQueue的队头;
38:          end if
39:     end for
40:end while
输入：以root为根节点的一棵决策树
————————————————————————————————————————
```

4.8

> 试将决策树生成的深度优先搜索过程修改为广度优先搜索，以参数MaxNode控制树的最大结点数，将题4.7中基于队列的决策树算法进行改写。对比题4.7中的算法，试分析哪种方式更易于控制决策树所需储存不超过内存。

在4.7中写的基于队列的算法本身就是广度优先搜索的。若要写成深度优先搜索的方式应该将队列换成堆栈即可。

以下对4.7中的代码进行改写，用队列+MaxNode控制

```
————————————————————————————————————————
输入：训练集 D={(x1,y1),(x2,y2),...,(xm,ym)};
      属性集 A={a1,a2,...,ad}
      最大节点数 MaxNode
过程：函数TreeGenerate(D,A,MaxNode)
1:生成节点root
2:if D中样本全部属于同一类别C then
3:     将root标记为C类叶节点;return
4:end if
5:if A=空集 or D中样本在A上取值相同 then
6:     将root标记为叶节点，其类别标记为D中样本数最多的类：return
7:end if
8:从A中选择最优划分属性a*;
9:将root标记为分支节点，属性为属性a*;
10:将root放入NodeQueue;
11:将D放入DataQueue;
12:将A\{a*}放入AQueue;
13:初始化节点数numNode=1;
```

```
14:while NodeQueue 非空:
15:     取出NodeQueue队尾的节点rNode，其对应的属性是ra*;
16:     取出DataQueue队尾的数据集rD;     #此处r均指队尾rear
17:     取出AQueue队尾的属性集rA;
18:     if numNode==MaxNode:
19:          将rNode标记为叶节点，类别标记为rD中样本最多的类;
20:          continue;     #跳过本次循环，即不再对这个节点做展开。对于下一个节点，由于条件任然成立，故任然不展开
21:     for ra*的每一个取值ra*v do:
22:          为rNode生成一个分支节点，令rDv表示rD在ra*上取值为ra*v的样本子集;
23:          if rDv为空 then:
24:               将分支节点标记为叶节点，其类别标记为rD中样本最多的类;
25:          else if rD中样本全部属于同一类别C then
26:               将分支节点标记为C类叶节点;
27:          else if rA=空集 or rD中样本在A上取值相同 then
28:               将分支节点标记为叶节点，其类别标记为rD中样本数最多的类;
29:          else:
30:               从rA中选择最优划分属性a*v;
31:               将分支节点的属性记为a*v;
32:               将分支节点放入NodeQueue的队头;
33:               将rDv放入DataQueue的队头;
34:               将rA\{a*v}放入AQueue的队头;
35:               将(rDepth+1)放入DepthQueue的队头;
36:          end if
37:          numNode+=1
38:     end for
39:end while
输入：以root为根节点的一棵决策树
```

讨论：4.7中与4.8中用队列的话均为广度优先搜索，我觉得是出题的时候的疏忽。。

应该是广度优先搜索（队列）+MaxNode控制 与 深度优先搜索（堆栈）+MaxDepth控制 两种方法之间的比较。

个人认为，广度优先搜索（队列）+MaxNode 的方法更容易控制决策树所需内存不溢出。因为最大节点数目是固定的。队列中储存的是当前深度未处理的节点以及当前深度以处理节点的下一级节点，其数目是可控的，总小于最大节点数。而深度优先搜索在堆栈中储存的是当前节点的兄弟节点、当前节点的父节点、当前节点的父节点的兄弟节点.....若一些分支节点的分支数很多，那么堆栈的深度就会比较深，虽然有MaxDepth控制最大深度，但还是可能出现栈溢出的情况。

4.9

> 试将4.4.2节对缺失值的处理机制推广到基尼指数的计算中去。

最近时间不多，暂且挖个坑。

4.10

> 从网上下载或自己编程实现任意一种多变量决策树算法，并观察其在西瓜数据集3.0上产生的结果。 答：此处要求实现一种多变量决策树算法。实际上4.3与4.4题就是多变量决策树算法。其在西瓜数据集3.0上产生的结果如下

最近时间不多，暂且挖个坑。

神经网络

在机器学习中，**神经网络（neural networks）**一般是指“神经网络学习”。所谓神经网络，目前用得最广泛的一个定义是“神经网络是由**具有适应性的简单单元**组成的广泛**并行互连的网络**，它的组织能够模拟生物神经系统**对真实世界物体所做出的反应**”。它是一种**黑箱模型**，解释性较差，但效果很好。目前已有一些工作尝试改善神经网络的可解释性。

这一章的内容大致如下：

- **神经元模型**：什么是神经元模型？它的构造是怎样的？激活函数是什么？如何组建神经网络？
- **感知机与多层网络**：感知机的构造是怎样的？感知机不能解决什么问题？多层网络是怎样的？多层前馈神经网络有什么特点？
- **误差逆传播算法**：BP算法如何调整参数？有什么值得注意的地方？标准BP算法和累积BP算法有什么区别？如何设置隐层神经元的个数？如何处理BP神经网络的过拟合问题？
- **全局最小与局部极小**：什么是全局最小？什么是局部极小？如何跳出局部极小？
- **其他常见神经网络**：有哪些常见的神经网络？它们各自有什么特点？什么是竞争型学习？可塑性、稳定性、增量学习、在线学习各指什么？
- **深度学习**：深度学习是如何提升模型容量的？如何训练深度神经网络？怎么理解深度学习？

神经元模型

神经元（**neuron**）模型是神经网络最基本的组成成分，不妨对比一下生物学中的神经元和机器学习中的神经元：

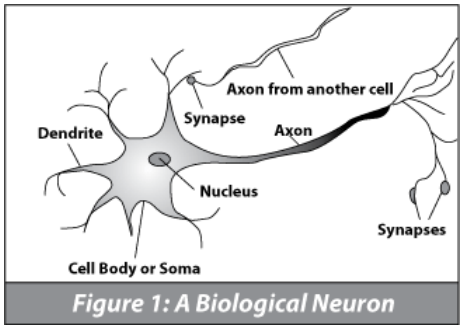
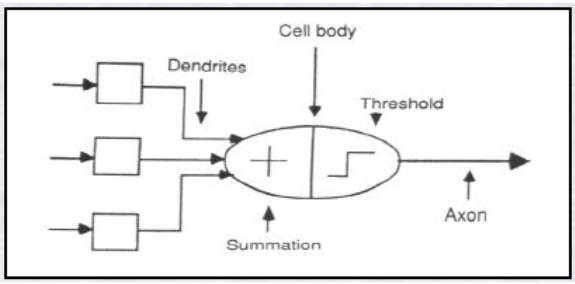


Figure 1: A Biological Neuron



在生物学中，每个神经元都有多个**树突（dendrite）**，一个**轴突（axon）**，以及一个**细胞体（cell body）**。当它**兴奋**时，就会向相连的神经元发送化学物质，从而改变它们内部的电位。如果一个神经元的电位超过了**阈值（threshold，也称bias）**，那它就会被**激活**，也即兴奋，继而向其他相连神经元发送化学物质。

在机器学习中，最常用的是右图的**M-P神经元模型**（亦称为**阈值逻辑单元（threshold logic unit）**）。树突对应于输入部分，每个神经元接收到若干个来自其他神经元的输入信号，这些信号通过带权重的**连接（connection）**传递给细胞体，这些权重又称为**连接权**。细胞体分为两部分，前一部分计算**总输入值（即输入信号的加权和，或者说累积电平）**；后一部分先计算总输入值与该神经元的阈值的差值，然后通过**激活函数（activation function）**的处理，从轴突输出给其它神经元。也即：

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

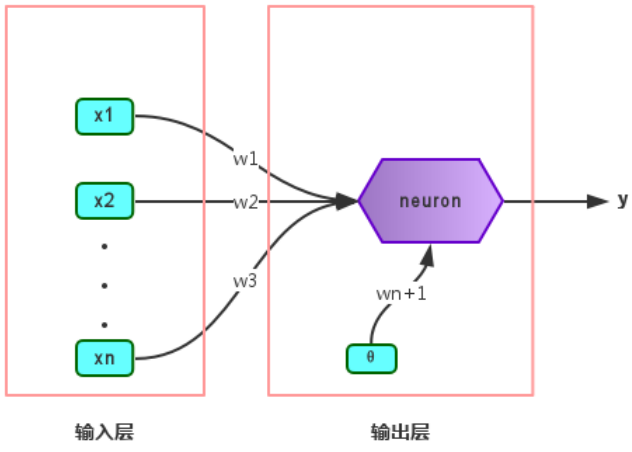
最理想的激活函数是阶跃函数，但它不连续。类似于线性分类，可以采用Sigmoid函数来近似。因为这类函数能把较大范围内变化的输入值挤压到 (0,1) 输出值范围内，所以也称为**挤压函数（squashing function）**。

将**多个神经元按一定的层次结构连接起来**，就得到了神经网络。它是一种包含许多参数的模型，比方说10个神经元两两连接，则有100个参数需要学习（每个神经元有9个连接权以及1个阈值）。若将每个神经元都看作一个函数，整个神经网络就是由这些函数相互嵌套而成。

感知机与多层网络

感知机

感知机（Perceptron）仅由两层神经元组成，如下图：



两层是指输入层和输出层，但**只有输出层是M-P神经元**，也即**只有一层功能神经元（functional neuron）**。输入层只负责把每一个样本的各个属性传递给输出层（**输入层的神经元数量等于样本的属性数目**），不进行函数处理。其实说白了这个模型跟逻辑回归是一样的，不过按我的理解就是感知机的输出层可以有多个神经元，产生多个输出。而且线性模型中偏置项是和属性一起加权求和的，但神经网络中则是求属性加权和和预测的差。

有时候阈值 θ 可以看作一个**输入固定为 -1.0 的哑结点（dummy node）**，连接权为 w_{n+1}。这样就可以把权重和阈值的学习统一为权重的学习了。更新权重的方式如下：

$$w_i \leftarrow w_i + \Delta w_i$$

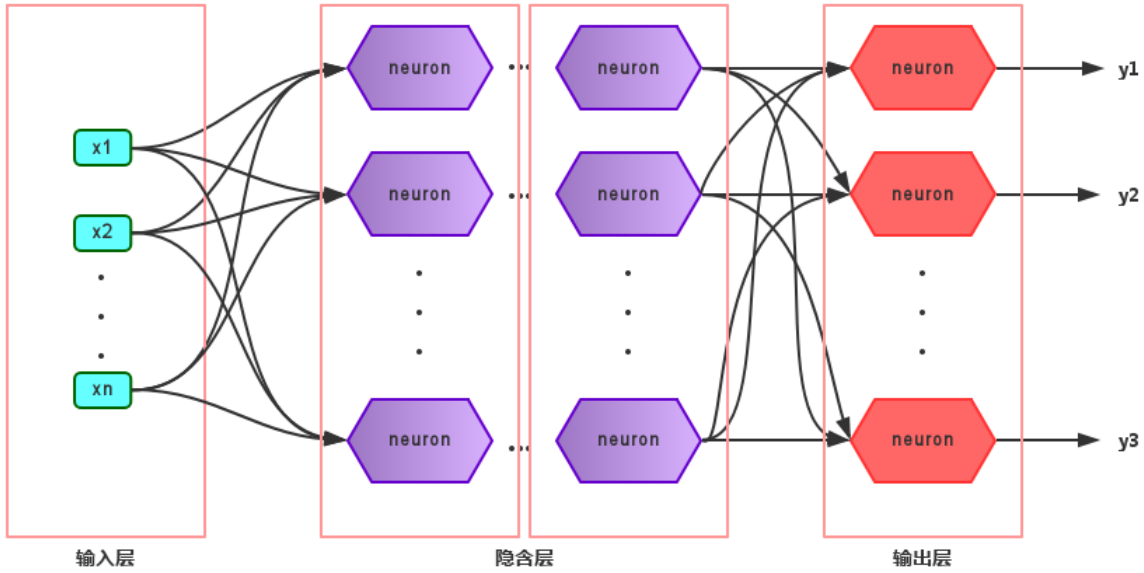
$$\Delta w_i = \eta(y - \hat{y})x_i$$

其中，η 称为**学习率（learning rate）**，取值范围是 (0,1)。感知机是**逐个数据点输入来更新**的。设定初始的权重后，逐个点输入，如果没有预测错就继续检验下一个点；如果预测错了就更新权重，然后重新开始逐个点检验，**直到所有点都预测正确了就停止更新**（所以这其实是一种最小化经验误差的方法）。

已经证明了，**若两类模式是线性可分（linearly separable）的，即存在一个线性超平面能将它们分开**。比如二维平面上可以用一条**直线**完全分隔开两个类别的点。由于感知机只有一层功能神经元，所以学习能力极其有限，**只能处理线性可分问题**。对于这类问题，感知机的学习过程必然**收敛（converge）**而求出适当的权向量；对于线性不可分问题，感知机的学习过程会发生**振荡（fluctuation）**，难以稳定下来。

多层网络

使用多层的功能神经元可以解决线性不可分问题，比方说两层（功能神经元）的感知机就可以解决异或问题（在二维平面中需要使用两条直线才能分隔开）。在输入层和输出层之间的层称为隐层或者**隐含层（hidden layer）**，隐含层的神经元也是功能神经元。



上图展示的最为常见的多层神经网络——**多层前馈神经网络（multi-layer feedforward neural networks）**，它有以下特点：

1. 每层神经元与下一层神经元**全互连**
2. 神经元之间**不存在同层连接**
3. 神经元之间**不存在跨层连接**

因为说两层网络时容易有歧义（只包含输入层输出层？还是包含两层功能神经元？），所以一般称包含一个隐含层的神经网络为**单隐层网络**。只要包含隐层，就可以称为**多层网络**。神经网络的学习其实就是调整各神经元之间的**连接权（connection weight）**以及各神经元的阈值，也即是说，**神经网络学到的东西都蕴含在连接权和阈值中**了。

误差逆传播算法

训练多层网络要比感知机复杂多了，感知机的学习方法是不够的。**误差逆传播算法（error BackPropagation，简称BP）**也称为**反向传播算法**，是最为成功的一种神经网络学习方法之一。一般而言，**BP神经网络**是指用BP算法训练的多层前馈神经网络，但BP算法也能训练其他类型的神经网络，如递归神经网络。

标准BP算法

假设要训练的是一个单隐层的前馈神经网络，BP算法使用**均方误差**作为性能度量，基于**梯度下降（gradient descent）**策略，以目标函数的负梯度方向对参数进行调整。

流程如下：

输入：训练集 $D = (\mathbf{x}_k, \mathbf{y}_k)_{k=1}^m$，学习率 η。

过程：
1：在 $(0, 1)$ 范围内随机初始化网络中所有连接权和阈值
2：**repeat**

3： **for all** $(\mathbf{x}_k, \mathbf{y}_k \in D)$ **do**
4： 根据当前参数计算出样本的输出 $\hat{\mathbf{y}}_k$
5： 计算输出层神经元的梯度项 g_j
6： 计算隐层神经元的梯度项 e_h
7： 更新连接权与阈值
8： **endfor**
9：**until** 达到停止条件

输出：连接权与阈值确定的多层前馈神经网络

所谓**逆传播**其实就是从输出层开始逐步往后更新，因为输出层的误差确定后就可以对输出层的连接权和阈值进行更新，并且可以推算出隐含层输出的“真实值”，从而计算出隐含层的“误差”，然后更新隐含层的连接权和阈值。BP就是这样一种**利用一层层倒推来最终更新整个神经网络**的方法，每一层的更新公式其实和感知机用的是类似的。

在学习过程中，学习率 η 控制着每一轮迭代的更新步长，**太大则容易振荡，太小则收敛速度太慢**。有时为了**精细调节**，输出层和隐含层更新参数时会**使用不同的学习率**。

累积BP算法

BP算法的目标是最小化训练集 D 上的**累积误差**：

$$E = \frac{1}{m}\sum_{k=1}^{m} E_k$$

而标准BP算法**每输入一个样例就进行一次更新**，所以它的参数更新非常频繁，而且不同样例可能会对更新起到**抵消**效果，从而使得模型需要更多次迭代才能到达累积误差的极小点。

标准BP算法和累积BP算法的区别类似于**随机梯度下降**和**标准梯度下降**的区别。

如果把更新方式变为**每输入一遍训练集进行一次更新**，就得到**累积BP算法**，更新公式需要重新推导一下。这样更新一次就称为一**轮（one round，亦称one epoch）**学习。

使用累积BP算法时参数更新的频率要低得多，但是！在很多人任务中，**累积误差在下降到一定程度后，进一步下降会非常缓慢**。此时标准BP算法能更快地获得较好的解（特别是训练集 D 很大的时候）。

隐层神经元个数

已证明**只需一个包含足够多神经元的隐层，多层前馈神经网络就能以任意精度逼近任意复杂度的连续函数**。那么该怎样设置隐层神经元的个数呢？**并不是越多越好**，因为可能会出现过拟合问题而得不偿失。

目前尚未有成熟的理论可以确定怎样设置隐层神经元的个数还有隐层的层数。实际应用中主要靠**试错法（trial-by-error）**来调整，也即是不断改变设置，试验模型给出的应答，然后选择结果最好的。

过拟合问题

鉴于BP神经网络强大的表达能力，很容易会遇到过拟合问题。主要有以下两种应对策略：

- **早停（early stopping）**：把数据集分为训练集和验证集，若训练集误差降低但测试集误差升高，就停止训练，并返回具有最小验证集误差的连接权和阈值。
- **正则化（regularization）**：在目标函数中添加一个用于**描述网络复杂度**的部分，比如连接权和阈值的平方和。这样训练时就会**偏好较小的连接权和阈值**，从而令输出更“光滑”。为什么可以用正则化来避免过拟合呢？可以看看知乎上的解答。带正则化项的目标函数如下：

$$E = \lambda \frac{1}{m}\sum_{k=1}^{m} E_k + (1-\lambda)\sum_i w_i^2$$

其中 λ 是用于对经验误差和网络复杂度折中的参数，常通过**交叉验证法**来估计。

全局最小与局部极小

学习模型的过程其实就是一个寻找最优参数的过程，在谈到最优时，一般会提到**局部极小（local minimum）**和**全局最小（global minimum）**。

- 局部极小解：参数空间中的某个点，其邻域点的误差函数值均不小于该点的误差函数值。
- 全局最小解：参数空间中的某个点，所有其他点的误差函数值均不小于该点的误差函数值。

这两个解对应的误差函数值分别成为**局部最小值**和**全局最小值**。

要成为局部极小，只要满足该点在参数空间中梯度为0就可以了。局部极小可以有多个，全局最小则只有一个。**全局最小一定也是局部极小，反之则不成立。**

因为用梯度下降搜索最优解可能会陷入非全局最小解的局部极小解，在现实任务中，人们会使用以下这些策略试图**跳出局部极小**：

- **以多组不同参数值初始化多个神经网络**：经过训练后，取误差最小的解作为最终参数。这种方法相当于从多个不同的初始点开始搜索，从而增加找到全局最小解的可能性。
- **模拟退火（simulated annealing）**技术：每次迭代都以一定的概率接收比当前解差的结果，从而有机会跳出局部极小（**当然也有可能跳出全局最小**），每次接受“次优解”的概率会随着时间推移逐渐减小，从而保证了算法的稳定。
- **随机梯度下降（stochastic gradient descent，简称SGD）**：在计算梯度时加入了随机因素，因此即便陷入局部极小点，梯度依然可能不为0，从而有机会跳出局部极小，继续搜索。

除了这些方法之外，**遗传算法**也常用于训练神经网络以逼近全局最小。当这些技术大多是**启发式**，没有理论的保障。也就是说，这些方法指示基于直观或者经验构造的，能**在可接受的时间和空间花费下给出待解决的组合优化问题的一个可行解**，但**不能估计可行解与最优解的偏离程度**。

其他常见神经网络

RBF网络

RBF（Radial Basis Function）网络是一种单隐层前馈神经网络，它**使用径向基函数作为隐层神经元的激活函数**。输出层则直接使用隐层神经元的**线性组合**。

ART网络

竞争型学习（competitive learning）是神经网络中常用的一种无监督学习策略。使用该策略时，网络中的输出神经元相互竞争，**每次只有一个竞争获胜的神经元被激活**，其它输出神经元被抑制，这种机制又称为**胜者通吃（winner-take-all）**。

ART（Adaptive Resonance Theory，自适应谐振理论）网络是竞争型学习的重要代表。该网络由四部份组成：**比较层、识别层、识别阈值、重置模块**。比较层就是输入层，只负责把样本传递给识别层。识别层也即输出层，但识别层的每个神经元对应一个模式类，而且**神经元的数目可以在训练过程中动态增加以增加新的模式类**。

识别层的每个神经元有一个对应的模式类的**代表向量**，每次输入一个样本，各识别层神经元相互竞争，代表向量与样本距离最小的胜出并被激活。获胜神经元会**向其他神经元发送信号，抑制其激活**。如果样本与获胜神经元的**距离小于识别阈值，则被分到对应的模式类**。否则，**重置模块会在识别层新增一个神经元，代表向量为该样本**。

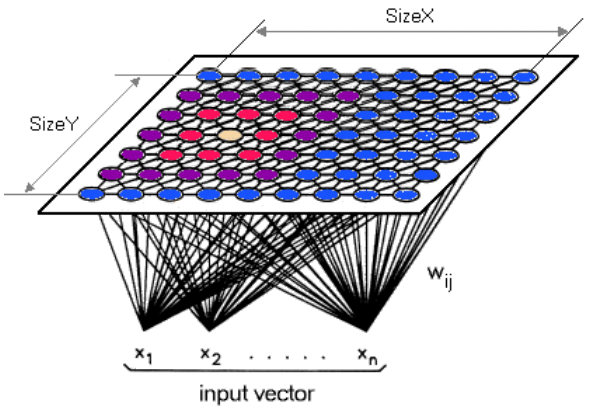
ART能有效缓解竞争型学习中的**可塑性-稳定性窘境（stability-plasticity dilemma）**。可塑性指神经网络要有**学习新知识的能力**，稳定性则指神经网络**在学习新知识时要保持对旧知识的记忆**。

ART具备可塑性和稳定性，因此能进行**增量学习（incremental learning）**和**在线学习（online learning）**。

增量学习可以理解为**建立模型后再收到新的样例时可以对模型进行更新，但不用重新训练整个模型**，先前学得的有效信息会被保存。它可以逐个新样例进行更新，也能以**批模型（batch-mode）**，每次用一批新样例来更新。

在线学习则可以理解为**每拿到一个新样本就进行一次模型更新**，不需要一开始就准备好完整的训练集，每次收到新样例都能继续训练。可以看作**增量学习的一个特例**。

SOM 网络



SOM（Self-Organizing Map，自组织映射）网络，又称为**自组织特征映射网络**或**Kohonen网络**。同样是一种竞争学习型无监督神经网络，只有输入层和输出层两层，输出层以矩阵形式排列。与样本距离最近的输出层神经元获胜，称为**最佳匹配单元（best matching unit）**。最佳匹配单元和邻近神经元的权向量会被调整，使得下次遇到相似的样本时距离更小。如此迭代，直至收敛。

级联相关网络

级联相关（Cascade-Correlation）网络是一种典型的**结构自适应网络**，这类网络不仅通过训练来学习合适的连接权和阈值等参数，还会在训练过程中**找到最符合数据特点的网络结构**。

级联相关神经网络有两个主要成分：

- **级联**：指**建立层次连接的层级结构**。开始训练时，只有输入层和输出层，随着训练进行逐渐加入隐层神经元，从而建立层级结构。注意，**隐层神经元的输入端连接权是冻结固定的**。
- **相关**：指**通过最大化新神经元的输出与网络误差之间的相关性（correlation）来训练相关的参数**。

Elman网络

递归神经网络（recurrent neural networks，简称RNN）允许网络中出现环形结构，即一些**神经元的输出可以反馈回来当输入信号**，从而能够处理与时间有关的动态变化。

Elman网络是最常用的递归神经网络之一，只有一个隐层，并且隐层神经元的输出会被反馈，在下一时刻与输入层神经元的输入信号一起作为隐层神经元的新输入。隐层神经元一般采用Sigmoid函数作为激活函数，并用BP算法训练整个网络。

Boltzmann机

神经网络中有一类**基于能量的模型（energy-based model）**，把网络状态定义为一个能量，能量最小时网络达到理想状态，**模型的学习过程就是最小化能量函数**。Boltzmann机就是这样的模型，同时也是一种RNN。

Boltzmann机的神经元**分为显层与隐层**，显层用于表达数据的输入与输出，隐层则是数据的内在。**每个神经元只有0、1两种状态**，也即抑制和激活。

标准的Boltzmann机是全连接图，即任意两个神经元之间都相连。但复杂度太高，难以用于解决现实任务，实际应用中用的是**受限Boltzmann机（Restricted Boltzmann Machine，简称RBM）**，把标准Boltzmann机退化为二部图，只保留显层和隐层之间的连接，同一层直接不相连。

深度学习

理论上，参数越多，模型复杂度就越高，**容量（capability，第12章中会讲到）**就越大，从而能完成更复杂的学习任务。**深度学习（deep learning）**正是一种极其复杂而强大的模型。

怎么增大模型复杂度呢？两个办法，一是**增加隐层的数目**，二是**增加隐层神经元的数目**。前者更有效一些，因为它不仅增加了功能神经元的数量，还增加了激活函数嵌套的层数。但是对于多隐层神经网络，经典算法如标准BP算法往往会在误差逆传播时**发散（diverge）**，无法收敛到稳定状态。

那要怎么有效地训练多隐层神经网络呢？一般来说有以下两种方法：

- **无监督逐层训练（unsupervised layer-wise training）**：每次训练一层隐结点，把上一层隐结点的输出当作输入来训练，本层隐结点训练好后，输出再作为下一层的输入来训练，这称为**预训练（pre-training）**。全部预训练完成后，再对整个网络进行**微调（fine-tuning）**训练。一个典型例子就是**深度信念网络（deep belief network，简称DBN）**。这种做法其实可以视为把大量的参数进行分组，先找出每组较好的设置，再基于这些局部最优的结果来训练全局最优。
- **权共享（weight sharing）**：令同一层神经元使用完全相同的连接权，典型的例子是**卷积神经网络（Convolutional Neural Network，简称CNN）**。这样做可以大大减少需要训练的参数数目。

事实上，深度学习可以理解为一种**特征学习（feature learning）**或者**表示学习（representation learning）**，无论是DBN还是CNN，都是**通过多个隐层来把初始与输出目标联系不大的输入表示转化为与输出目标更密切的表示**，使原来只通过单层映射难以完成的任务变为可能。也即通过多层处理，逐渐**将初始的“低层”特征表示转化为“高层”特征表示**，从而使得最后可以用简单的模型来完成复杂的学习任务。

传统任务中，样本的特征需要人类专家来设计，这称为**特征工程（feature engineering）**。特征好坏对泛化性能有至关重要的影响。而深度学习为**全自动数据分析**带来了可能，可以自动产生好的特征。

习题

5.1

> 问：试述将线性函数$f(\mathbf{x})=\mathbf{w}^T\mathbf{x}$用作神经元激活函数的缺陷。

5.2

> 问：试述使用对数几率函数作为激活函数的神经元与对率回归的联系。

5.3

> 问：对于图5.7中的v_{ih}，试推导出BP算法中的更新公式(5.13)。

5.4

> 问：试述式(5.6)中学习率的取值对神经网络训练的影响。

5.5

> 问：试编程实现标准BP算法和累积BP算法，在西瓜数据集3.0上分别用这两个算法训练一个单隐层网络，并进行比较。

5.6

> 问：试设计一个BP改进算法，能通过动态调整学习率显著提升收敛速度。编程实现该算法，并选择两个UCI数据集与标准BP算法进行实验比较。

5.7

> 问：根据式(5.18)和(5.19)，试构造一个能解决抑或问题的单层RBF神经网络。

5.8

> 问：从网上下载或自己编程实现SOM网络，并观察其在西瓜数据集3.0α上产生的效果。

5.9*

> 问：试推导用于Elman网络的BP算法。

5.10

> 问：从网上下载或自己编程实现一个卷积神经网络，并在手写字符识别数据MNIST上进行实验编程。

支持向量机

支持向量机（Support Vector Machine，简称SVM）是一种针对二分类任务设计的分类器，它的理论相对神经网络模型来说更加完备和严密，并且效果显著，结果可预测，是非常值得学习的模型。

这一章的内容大致如下：

- **间隔与支持向量**：如何计算空间中任一点到超平面的距离？什么是支持向量？什么是间隔？支持向量机求解的目标是什么？
- **对偶问题**：求取最大间隔等价于怎样的对偶问题？KKT条件揭示出支持向量机的什么性质？如何用SMO算法进行高效求解？为什么SMO算法能高效求解？
- **核函数**：如何处理非线性可分问题？什么是核函数？为什么需要核函数？有哪些常用的核函数？核函数具有什么性质？
- **软间隔与正则化**：如何应对过拟合问题？软间隔和硬间隔分别指什么？如何求解软间隔支持向量机？0/1损失函数有哪些可选的替代损失函数？支持向量机和对率回归模型有什么联系？结构风险和经验风险分别指什么？
- **支持向量回归**：什么是支持向量回归？与传统回归模型有什么不同？支持向量回归的支持向量满足什么条件？
- **核方法**：什么是表示定理？什么是核方法？如何应用？

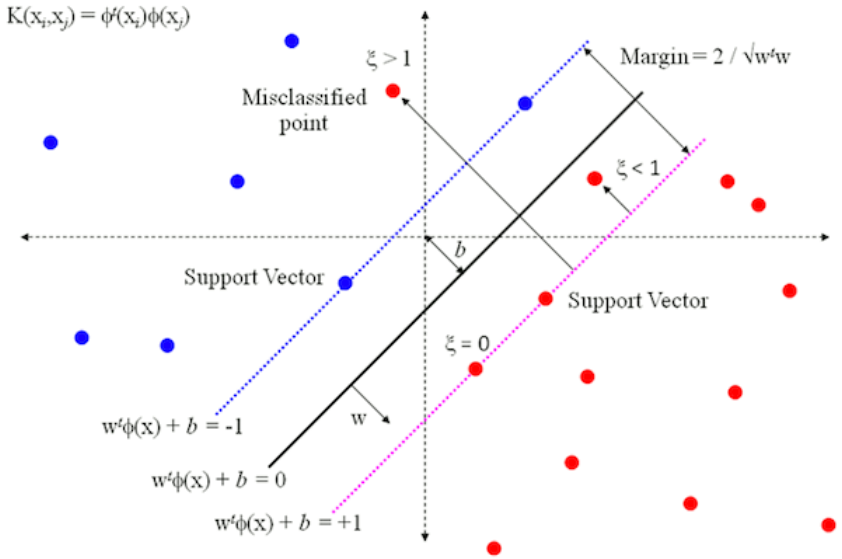
间隔与支持向量

给定一个二分类数据集，正类标记为+1，负类标记为-1（对率回归中负类标记是0，这点是不同的）。

分类学习试图从样本空间中找到一个超平面，使得该超平面可以将不同类的样本分隔开。但是满足这样条件的平面可能有很多，哪一个才是最好的呢？

支持向量

在SVM中，我们试图找到**处于两类样本正中间的超平面**，因为这个超平面**对训练数据局部扰动的容忍性最好**，新样本最不容易被误分类。也就是说这个超平面**对未见示例的泛化能力最强**。



上图的实线就是划分超平面，在线性模型中可以通过方程 $\mathbf{w}^T\mathbf{x}+b=0$ 来描述，在二维样本空间中就是一条直线。图中的 $\phi(\mathbf{x})$ 是使用了核函数进行映射，这里先不讨论。$\mathbf{w}$ 是线性模型的权重向量（又叫**投影向量**），也是**划分超平面的法向量，决定着超平面的方向**。偏置项 b 又被称为 **位移项，决定了超平面和空间原点之间的距离**。

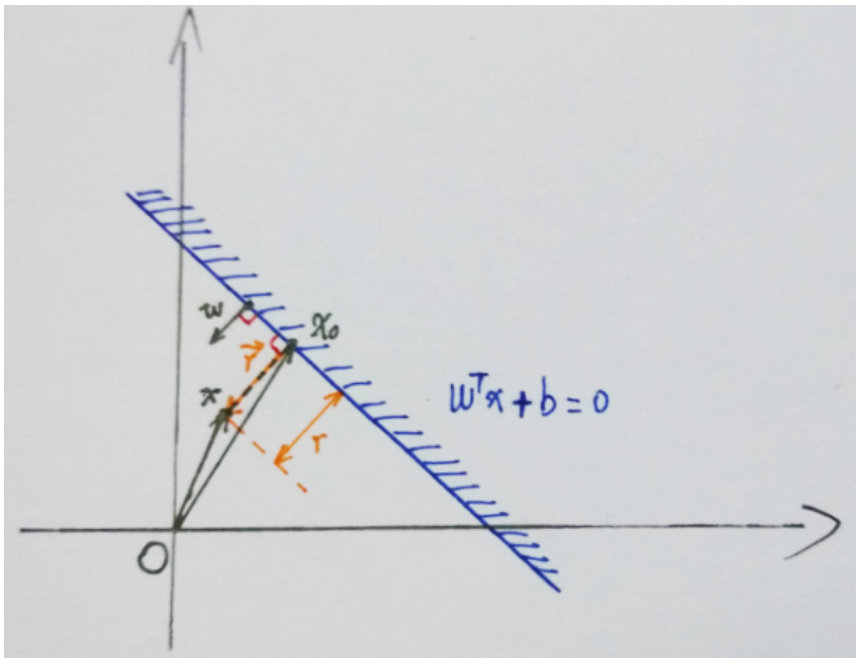
假设超平面能够将所有训练样本正确分类，也即对于所有标记为+1的点有 $\mathbf{w}^T\mathbf{x}+b>0$，所有标记为-1的点有 $\mathbf{w}^T\mathbf{x}+b<0$。只要这个超平面存在，那么我们必然可以对 $\mathbf{w}$ 和 b 进行适当的**线性放缩**，使得：

$$\mathbf{w}^T\mathbf{x}+b\geq+1,\quad y_i=+1$$

$$\mathbf{w}^T\mathbf{x}+b\leq-1,\quad y_i=-1$$

而SVM中定义**使得上式等号成立的训练样本点**就是**支持向量（support vector）**（如果叫作**支持点**可能更好理解一些，因为事实上就是样本空间中的数据点，但因为我们在表示数据点的时候一般写成向量形式，所以就称为支持向量），它们是距离超平面最近的几个样本点，也即上面图中两条虚线上的点（图中存在比支持向量距离超平面更近的点，这跟**软间隔**有关，这里先不讨论）。

在SVM中，我们希望实现的是**最大化两类支持向量到超平面的距离之和**，那首先就得知道怎么计算距离。**怎样计算样本空间中任意数据点到划分超平面的距离**呢？



画了一个图，方便讲解。图中蓝色线即超平面，对应直线方程 $\mathbf{w}^T\mathbf{x}+b=0$。投影向量 $\mathbf{w}$垂直于超平面，点 x 对应向量 $\mathbf{x}$，过点 x 作超平面的垂线，交点 x_0 对应向量 $\mathbf{x_0}$。假设**由点 x_0 指向 点 x 的向量**为 $\mathbf{r}$，长度（也即点 x 与超平面的距离）为 r。有两种方法计算可以计算出 r 的大小：

方法1：向量计算

由向量加法定义可得 $\mathbf{x}=\mathbf{x_0}+\mathbf{r}$。

那么向量 $\mathbf{r}$ 等于什么呢？它等于这个方向的单位向量乘上 r，也即有 $\mathbf{r}=\frac{\mathbf{w}}{\|\mathbf{w}\|}\cdot r$

因此又有 $\mathbf{x}=\mathbf{x_0}+\frac{\mathbf{w}}{\|\mathbf{w}\|}\cdot r$。

由于点 x_0 在超平面上，所以有 $\mathbf{w}^T\mathbf{x_0}+b=0$

由 $\mathbf{x} = \mathbf{x_0} + \frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot r$ 可得 $\mathbf{x_0} = \mathbf{x} - \frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot r$，代入直线方程消去 $\mathbf{x_0}$：

$$\mathbf{w}^T\mathbf{x_0} + b = \mathbf{w}^T(\mathbf{x} - \frac{\mathbf{w}}{\|\mathbf{w}\|} \cdot r) + b = 0$$

简单变换即可得到:

$$r = \frac{\mathbf{w}^T\mathbf{x} + b}{\|\mathbf{w}\|}$$

又因为我们取距离为正值，所以要加上绝对值符号：

$$r = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|}$$

方法2：点到直线距离公式

假设直线方程为 $ax_1 + bx_2 + c = 0$，那么有点到直线距离公式：

$$r = \frac{|ax + bx_2 + c|}{\sqrt{a^2 + b^2}}$$

令 $\mathbf{w} = (a, b)$，$\mathbf{x} = (x_1, x_2)$，则可以把 $ax_1 + bx_2$ 写成向量形式 $\mathbf{w}^T\mathbf{x}$。把截距项设为b，则直线方程变为 $\mathbf{w}^T\mathbf{x} + b = 0$，代入距离公式可得：

$$r = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\sqrt{\mathbf{w}^T\mathbf{w}}} = \frac{|\mathbf{w}^T\mathbf{x} + b|}{\|\mathbf{w}\|}$$

该式扩展到多维情况下也是通用的。

间隔

前面已经提到，我们希望实现的是**最大化两类支持向量到超平面的距离之和**，而根据定义，所有支持向量都满足：

$$\mathbf{w}^T\mathbf{x} + b = +1, \quad y_i = +1$$

$$\mathbf{w}^T\mathbf{x} + b = -1, \quad y_i = -1$$

代入前面的距离公式可以得到支持向量到超平面的距离为 $\frac{1}{\|\mathbf{w}\|}$。

定义**间隔**（margin）为**两个异类支持向量到超平面的距离之和**：

$$\gamma = 2 \cdot \frac{1}{\|\mathbf{w}\|} = \frac{2}{\|\mathbf{w}\|}$$

SVM的目标便是找到**具有最大间隔（maximum margin）**的划分超平面，也即找到使 γ 最大的参数 $\mathbf{w}$ 和 b：

$$\max_{\mathbf{w},b} \frac{2}{\|\mathbf{w}\|} \quad s.t. \quad y_i(\mathbf{w}^T\mathbf{x} + b) \geq 1, \quad i = 1, 2, \ldots, m$$

约束部分指的是全部样本都被正确分类，此时标记值（$+1$ 或 -1）乘上预测值（$\geq +1$ 或 ≤ -1）必定是一个 ≥ 1 的数值。

看上去间隔大小只与 $\mathbf{w}$ 有关，但实际上位移项 b 也通过约束影响着 $\mathbf{w}$ 的取值，进而对间隔产生影响。

由于最大化 $\|\mathbf{w}\|^{-1}$ 等价于最小化 $\|\mathbf{w}\|^2$，所以可以重写**目标函数**为：

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad s.t. \quad y_i(\mathbf{w}^T\mathbf{x} + b) \geq 1, \quad i = 1, 2, \ldots, m \qquad (1)$$

引入 $\frac{1}{2}$ 是为了求导时可以约去平方项的2，这便是**支持向量机的基本型**。

特别地，还有以下定义：

函数间隔：$y_i(\mathbf{w}^T\mathbf{x} + b)$

几何间隔：$\frac{y_i(\mathbf{w}^T\mathbf{x}+b)}{\|\mathbf{w}\|^2}$

对偶问题

式（1）是一个**带约束的凸二次规划（convex quadratic programming）问题**（凸问题就意味着必定能求到全局最优解，而不会陷入局部最优）。对这样一个问题，可以直接用现成的优化计算包求解，但这一小节介绍的是一种更高效的方法。

首先为式（1）的每条约束添加拉格朗日乘子 $a_i \geq 0$（对应m个样本的m条约束），得到该问题的拉格朗日函数：

$$L(\mathbf{w},b,\mathbf{a})=\frac{1}{2}\|\mathbf{w}\|^2+\sum_{i=1}^{m}a_i(1-y_i(\mathbf{w}^T\mathbf{x}_i+b)) \qquad (2)$$

其中 $\mathbf{a}=(a_1;a_2;\dots;a_m)$，对拉格朗日函数求 $\mathbf{w}$ 和 b 的偏导，并令偏导为0可以得到：

$$\mathbf{w}=\sum_{i=1}^{m}a_iy_i\mathbf{x}_i \qquad (3)$$

$$0=\sum_{i=1}^{m}a_iy_i \qquad (4)$$

将式（3）代入式（2）可以消去 $\mathbf{w}$，又因为式（2）中 b 的系数是 a_iy_i，由式（4）可知 b 也可以消去。然后再考虑式（4）的约束就得到了式（1）的**对偶问题**（dual problem）：

$$\max_{\mathbf{a}}\sum_{i=1}^{m}a_i-\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}a_ia_jy_iy_j\mathbf{x}_i^T\mathbf{x}_j$$

$$\text{s.t.}\sum_{i=1}^{m}a_iy_i=0,\quad a_i\geq 0,\quad i=1,2,\dots,m \qquad (5)$$

只要求出该对偶问题的解 $\mathbf{a}$，就可以推出 $\mathbf{w}$ 和 b，从而得到模型：

$$\begin{aligned}f(\mathbf{x})&=\mathbf{w}^T\mathbf{x}+b\\&=\sum_{i=1}^{m}a_iy_i\mathbf{x}_i^T\mathbf{x}+b\end{aligned} \qquad (6)$$

不过实际计算时一般不会直接求解 $\mathbf{a}$，特别是需要用核函数映射到高维空间时，因为映射后做内积很困难，而用少量支持向量进行表示，在原始空间进行计算显然更优，这点在后续章节会详细讲解。

注意，由于式（1）的约束条件是**不等式约束**，所以求解过程要求满足**KKT（Karush-Kuhn-Tucker）条件**：

$$\begin{cases}a_i\geq 0;\\ y_if(\mathbf{x}_i)-1\geq 0;\\ a_i(y_if(\mathbf{x}_i)-1)=0.\end{cases}$$

这个KKT条件说明了，对任何一个样本 $\mathbf{x}_i$ 来说，

- 要么对应的拉格朗日乘子 a_i 为0，此时样本 $\mathbf{x}_i$ 对式（6）毫无贡献，不会影响到模型；
- 要么函数间隔 $y_if(\mathbf{x}_i)=1$，此时样本 $\mathbf{x}_i$ 位于最大间隔边界上，是一个支持向量。

它揭示了SVM的一个重要性质：**最终模型只与支持向量有关，因此训练完成后，大部分的训练样本都不需保留**。

SMO算法

可以发现对偶问题式（5）是一个二次规划问题，可以使用通用的二次规划算法求解。但**问题规模正比于样本数**，因此开销相当大。为了避免这个开销，可以使用高效的**SMO（Sequential Minimal Optimization）算法**。

初始化参数 $\mathbf{a}$ 后，SMO算法重复下面两个步骤直至收敛：

1. 选取一对需要更新的变量 a_i 和 a_j
2. 固定 a_i 和 a_j 以外的参数，求解对偶问题式（5）来更新 a_i 和 a_j

怎么选取 a_i 和 a_j呢？

注意到，只要选取的 a_i 和 a_j 中有一个不满足KKT条件，那么更新后目标函数的值就会增大。而且**违背KKT条件的程度越大，则更新后导致目标函数增幅就越大**。

因此，SMO算法**先选取一个违背KKT条件程度最大的变量** a_i，然后再选一个使目标函数增长最快的变量 a_j，但由于找出 a_j 的开销较大，所以SMO算法采用了一个**启发式**，使**选取的两变量对应的样本之间间隔最大**。这样两个变量差别很大，与选取两个相似变量相比，这种方法能为目标函数带来更大的变化，从而更快搜索到全局最大值。

由于SMO算法在每次迭代中，仅优化两个选定的参数，其他参数是固定的，所以会非常高效。此时，可将对偶问题式（5）的约束重写为：

$$a_iy_i+a_jy_j=c,\quad a_i\geq 0,a_j\geq 0 \qquad (7)$$

其中，$c=-\sum_{k\neq i,j}a_ky_k$ 看作是固定的常数。

利用式（7），我们可以把 a_j 从式（5）中消去，这样就得到了一个**单变量二次规划问题**，只需考虑 $a_i\geq 0$ 这个约束。这样的问题具有**闭式解**，所以我们连数值优化方法都不需要了，可以直接算出 a_i 和 a_j。

使用SMO算法计算出最优解之后，我们关注的是如何推出 $\mathbf{w}$ 和 b，从而得到最终模型。获得 $\mathbf{w}$ 很简单，直接用式（3）就可以了。而位移项 b 则可以通过支持向量导出，因为对于任一支持向量 $(\mathbf{x}_s, y_s)$，都有函数间隔等于1，所以有：

$$y_s f(\mathbf{x}) = y_s(\sum_{i \in S} a_i y_i \mathbf{x}_i^T \mathbf{x}_s + b) = 1 \qquad (8)$$

这里的 S 是所有支持向量的下标集（事实上，用所有样本的下标也行，不过非支持向量的拉格朗日乘子等于0，对求和没贡献，这一点前面已经提到了）。

理论上，我们只要选取任意一个支持向量代入式（8）就可以把 b 算出来了。但实际任务中往往采用一种**更鲁棒**的做法：用所有支持向量求解的平均值。

$$b = \frac{1}{|S|} \sum_{s \in S} (\frac{1}{y_s} - \sum_{i \in S} a_i y_i \mathbf{x}_i^T \mathbf{x}_s)$$

核函数

如何处理非线性划分

在现实任务中，我们更常遇到的是**在原始样本空间中非线性可分**的问题。对这样的问题，一种常用的思路是将样本从原始空间映射到一个更高维的特征空间，使得样本在该特征空间中线性可分。幸运的是，**只要原始空间是有限维的（也即属性数目有限），那就必然存在一个高维特征空间使样本线性可分**。

举个例子，二维平面上若干样本点呈如下分布：

此时要划分两类样本，需要一个非线性的圆型曲线。假设原始空间中两个属性是 x 和 y，如果我们做一个映射，把样本点都映射到一个三维特征空间，维度取值分别为 x^2，y^2 和 y，则得到下面的分布：

可以看到这个时候，我们只需要一个线性超平面就可以将两类样本完全分开了，也就是说可以用前面的方法来求解了。

什么是核函数

在上面的例子中，我们是把每个样本对应的二维的特征向量 $\mathbf{x}$ 映射为一个三维的特征向量，假设我们用 $\phi(\mathbf{x})$ **来表示映射所得的特征向量**。则**在映射的高维特征空间中**，用于划分的线性超平面可以表示为：

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

类似式（1），可以得到此时的目标函数为：

$$\min_{\mathbf{w},b} \frac{1}{2} \|\mathbf{w}\|^2 \quad s.t. \quad y_i(\mathbf{w}^T \phi(\mathbf{x}) + b) \geq 1, \quad i = 1, 2, \ldots, m \qquad (9)$$

对应的对偶问题为：

$$\max_{\mathbf{a}} \sum_{i=1}^{m} a_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} a_i a_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j) \quad s.t. \quad \sum_{i=1}^{m} a_i y_i = 0, \quad a_i \geq 0, \quad i = 1, 2, \ldots, m \qquad (10)$$

注意到对偶问题中，涉及到 $\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ 的计算，也即 x_i 和 x_j 映射到高维特征空间后的内积（比如 $x_i = (1, 2, 3)$，$x_j = (4, 5, 6)$，那么内积 $x_i^T x_j$ 就等于 $1 * 4 + 2 * 5 + 3 * 6 = 32$），由于**特征空间维数可能很高**，所以**直接计算映射后特征向量的内积是很困难的**，如果映射后的特征空间是无限维，根本无法进行计算。

为了解决这样的问题，就引入了**核函数（kernel function）**。

打个比方，假设输入空间是二维的，每个样本点有两个属性 x 和 y，存在映射将每个样本点映射到三维空间：

$$\phi(\mathbf{x}) = \phi(x, y) = (x^2, \sqrt{2}xy, y^2)$$

给定原始空间中的两个样本点 $\mathbf{v}_1 = (x_1, y_1)$ 和 $\mathbf{v}_2 = (x_2, y_2)$，则它们映射到高维特征空间后的内积可以写作：

$$\begin{aligned}
\phi(\mathbf{v}_1)^T \phi(\mathbf{v}_2) &=< \phi(\mathbf{v}_1), \phi(\mathbf{v}_2) > \\
&=< (x_1^2, \sqrt{2}x_1 y_1, y_1^2), (x_2^2, \sqrt{2}x_2 y_2, y_2^2) > \\
&= x_1^2 x_2^2 + 2x_1 x_2 y_1 y_2 + y_1^2 y_2^2 \\
&= (x_1 x_2 + y_1 y_2)^2 \\
&=< \mathbf{v}_1, \mathbf{v}_2 >^2 \\
&= \kappa(\mathbf{v}_1, \mathbf{v}_2)
\end{aligned}$$

可以看到在这个例子里，高维特征空间中两个点的内积，可以写成一个**关于原始空间中两个点的函数** $\kappa(\cdot;\cdot)$，这就是核函数。

特别地，上面的例子中，映射用的是**多项式核**，多项式的次数 d 取2。

为什么需要核函数

这里的例子为了计算方便，映射的空间维数依然很低，这里稍微解释一下**为什么需要核函数**？假设原始空间是二维的，那么对于两个属性 x 和 y，取一阶二阶的组合只有5个（也即 x^2，y^2，x，y，xy）。但当原始空间是三维的时候，仍然取一阶二阶，组合就多达19个了（也即 x，y，z，xy，xz，yz，x^2y，x^2z，y^2x，y^2z，z^2x，z^2y，x^2yz，xy^2z，xyz^2，x^2y^2z，x^2yz^2，xy^2z^2，xyz）。**随着原始空间维数增长，新空间的维数是呈爆炸性上升的**。何况现实中我们遇到的问题的原始空间往往本来就已经是高维的，如果再进行映射，新特征空间的维度是难以想象的。

然而有了核函数，我们就可以在原始空间中通过函数 $\kappa(\cdot;\cdot)$ 计算（这称为**核技巧（kernel trick）**），而**不必直接计算高维甚至无穷维特征空间中的内积**。

使用核函数后，对偶问题式（10）可以重写为：

$$\max_{\mathbf{a}} \sum_{i=1}^{m} a_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} a_i a_j y_i y_j \kappa(\mathbf{x}_i;\mathbf{x}_j) \quad s.t. \quad \sum_{i=1}^{m} a_i y_i = 0, \quad a_i \geq 0, \quad i = 1,2,\ldots,m \qquad (11)$$

求解后得到的模型可以表示为：

$$\begin{aligned} f(\mathbf{x}) &= \mathbf{w}^T\phi(\mathbf{x}) + b \\ &= \sum_{i=1}^{m} a_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b \\ &= \sum_{i=1}^{m} a_i y_i \kappa(\mathbf{x}_i;\mathbf{x}) + b \end{aligned}$$

这条式子表明了**模型最优解可通过训练样本的核函数展开**，称为**支持向量展式（support vector expansion）**。

在需要对新样本进行预测时，我们**无需把新样本映射到高维（甚至无限维）空间**，而是可以**利用保存下来的训练样本（支持向量）和核函数 κ 进行求解**。

注意，**核函数本身不等于映射！！！**它只是一个与**计算两个数据点映射到高维空间之后的内积**等价的函数。 当我们发现数据在原始空间线性不可分时，会有把数据映射到高维空间来实现线性可分的想法，比方说引入原有属性的幂或者原有属性之间的乘积作为新的维度。假设我们把数据点都映射到了一个维数很高甚至无穷维的特征空间，而**模型求解和预测的过程需要用到映射后两个数据点的内积**，这时直接计算就没辙了。但我们又幸运地发现，原来**高维空间中两点的内积在数值上等于原始空间通过某个核函数算出的函数值**，无需先映射再求值，就很好地解决了计算的问题了。

核函数的性质

核函数定理：给定一个输入空间 $\mathcal{X}$，函数 $\kappa(\cdot;\cdot)$ 是定义在 $\mathcal{X}\times\mathcal{X}$ 上的**对称函数**。当且仅当对于任意数据集 $D=\{\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_m\}$, 对应的**核矩阵（kernel matrix）**都是半正定的时候，κ 是核函数。

核矩阵是一个规模为 $m\times m$ 的函数矩阵，每个元素都是一个函数，比如第 i 行 j 列的元素是 $\kappa(\mathbf{x}_i,\mathbf{x}_j)$。也即是说，**任何一个核函数都隐式地定义了一个称为“再生核希尔伯特空间（Reproducing Kernel Hilbert Space，简称RKHS）”的特征空间**。

做映射的初衷是希望**样本在新特征空间上线性可分**，新特征空间的好坏直接决定了支持向量机的性能，但是我们并**不知道怎样的核函数是合适的**。一般来说有以下几种常用核函数：

名称	表达式	参数
线性核	$\kappa(\mathbf{x}_i,\mathbf{x}_j)=\mathbf{x}_i^T\mathbf{x}_j$	-
多项式核	$\kappa(\mathbf{x}_i,\mathbf{x}_j)=(\mathbf{x}_i^T\mathbf{x}_j)^d$	$d\geq 1$为多项式的次数，d=1时退化为线性核
高斯核（亦称RBF核）	$\kappa(\mathbf{x}_i,\mathbf{x}_j)=\exp(-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|^2}{2\sigma^2})$	$\sigma>0$ 为高斯核的带宽（width）
拉普拉斯核	$\kappa(\mathbf{x}_i,\mathbf{x}_j)=\exp(-\frac{\|\mathbf{x}_i-\mathbf{x}_j\|}{\sigma})$	$\sigma>0$
Sigmoid核	$\kappa(\mathbf{x}_i,\mathbf{x}_j)=\tanh(\beta\mathbf{x}_i^T\mathbf{x}_j+\theta)$	$tanh$ 为双曲正切函数，$\beta>0,\theta<0$

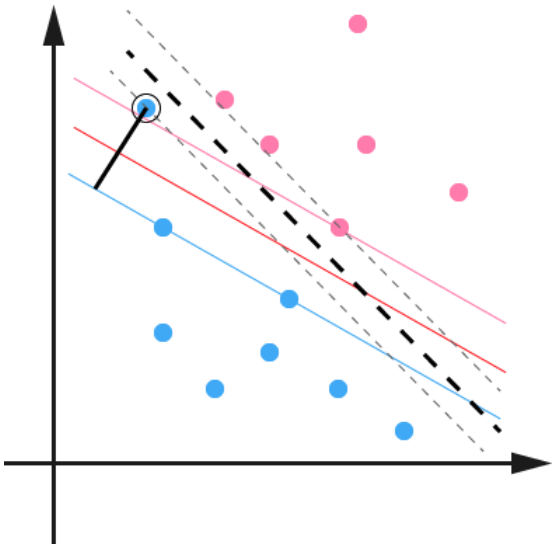
特别地，**文本数据一般用线性核，情况不明可尝试高斯核**。

除了这些常用的核函数，要**产生核函数还可以使用组合的方式**：

- 若 κ_1 和 κ_2 都是核函数，则 $a\kappa_1+b\kappa_2$ 也是核函数，其中 $a>0,b>0$。
- 若 κ_1 和 κ_2 都是核函数，则其直积 $\kappa_1\otimes\kappa_2(\mathbf{x},\mathbf{z})=\kappa_1(\mathbf{x},\mathbf{z})\kappa_2(\mathbf{x},\mathbf{z})$ 也是核函数。
- 若 κ_1 是核函数，则对于任意函数 $g(\mathbf{x})$，$\kappa(\mathbf{x},\mathbf{z})=g(\mathbf{x})\kappa_1(\mathbf{x},\mathbf{z})g(\mathbf{z})$ 也是核函数。

软间隔与正则化

上一节中，通过利用核函数映射来解决非线性可分的问题，但现实中很难找到合适的核函数，即使某个核函数能令训练集在新特征空间中线性可分，也难保这不是**过拟合**造成的结果。



比方说上面这张图，黑色虚线是此时的划分超平面，最大间隔很小。但事实上，黑色圆圈圈起的蓝点是一个 **outlier**，可能是噪声的原因，它**偏离了正确的分布**。而训练模型时，我们并没有考虑这一点，这就导致**把训练样本中的 outlier当成数据的真实分布拟合了**，也即过拟合。

但当我们**允许这个 outlier 被误分类**时，得到的划分超平面可能就如图中深红色线所示，此时的最大间隔更大，预测新样本时误分类的概率也会降低很多。

在实际任务中，outlier 的情况可能更加严重。比方说，如果图中的 outlier 再往右上移动一些距离的话，我们甚至会**无法构造出一个能将数据划分开的超平面**。

缓解该问题的一个思路就是**允许支持向量机在一些样本上出错**，为此，引入**软间隔（soft margin）**的概念。软间隔是相对于**硬间隔（hard margin）**的一个概念，**硬间隔要求所有样本都必须划分正确**，也即约束：

$$y_i(\mathbf{w}^T\mathbf{x}_i+b)\geq 1$$

软间隔则**允许某些样本不满足约束**（根据约束条件的不同，有可能某些样本出现在间隔内，甚至被误分类）。此时目标函数可以重写为：

$$\min_{\mathbf{w},b}\frac{1}{2}\|\mathbf{w}\|^2+C\sum_{i=1}^{m}\ell_{0/1}(y_i(\mathbf{w}^T\mathbf{x}+b)-1) \qquad (12)$$

其中 $\ell_{0/1}$ 是**0/1损失函数**：

$$\ell_{0/1}(z)=\begin{cases}1, & if\ z<0;\\ 0, & otherwise.\end{cases}$$

它的含义很简单：如果分类正确，那么函数间隔必定大于等于1，此时损失为0；如果分类错误，那么函数间隔必定小于等于-1，此时损失为1。

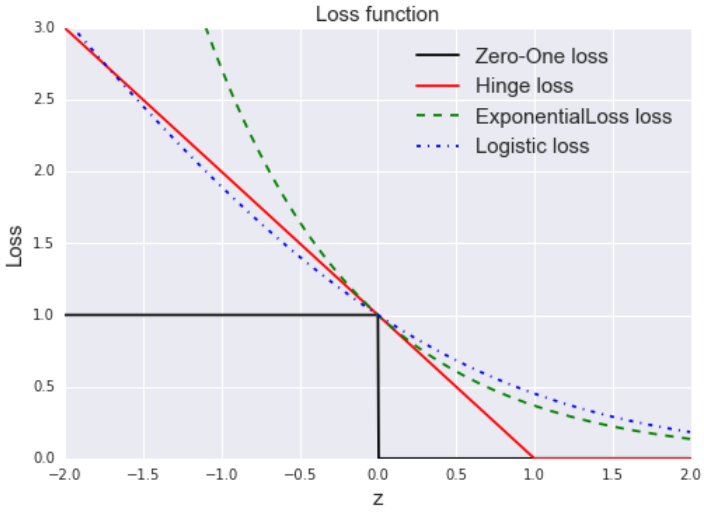
而 C 则是一个大于0的常数，当 C 趋于无穷大时，式（12）等效于带约束的式（1），因为此时对误分类的惩罚无限大，也即要求全部样本分类正确。**当 C 取有限值时，允许某些样本分类错误**。

由于**0/1损失函数是一个非凸不连续函数**，所以式（12）难以求解，于是在实际任务中，我们采用一些**凸的连续函数**来取替它，这样的函数就称为**替代损失（surrogate loss）函数**。

最常用的有以下三种：

- hinge损失：$\ell_{hinge}(z)=\max(0,1-z)$
- 指数损失（exponential loss）：$\ell_{\exp}(z)=\exp(-z)$
- 对率损失（logistic loss）：$\ell_{\log}(z)=\log(1+\exp(-z))$

不妨作图观察比较一下这些损失函数（code文件夹下有实现代码）：



这里有个问题是，书中提到对率损失中 $\log$ 指 $\ln$，也即底数为自然对数，但这种情况下对率损失在 $z=0$ 处不为1，而是0.693。但是书中的插图里，对率损失经过 $(0,1)$ 点，此时底数应为2，上面的插图就是按底数为2计算的。

实际任务中最常用的是hinge损失，这里就以hinge损失为例，替代0/1损失函数，此时目标函数式（12）可以重写为：

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\max(0, 1-y_i(\mathbf{w}^T\mathbf{x}+b)) \qquad (13)$$

引入松弛变量（slack variables） $\xi_i \geq 0$，可以把式（13）重写为：

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \quad s.t. \quad y_i(\mathbf{w}^T\mathbf{x}+b) \geq 1-\xi_i, \quad \xi_i \geq 0, \quad i=1,2,\ldots,m \qquad (14)$$

该式描述的就是软间隔支持向量机，其中每个样本都对应着一个松弛变量，用以表示该样本误分类的程度，松弛变量的值越大，程度越高。

求解软间隔支持向量机

式（14）仍然是一个二次规划问题，类似于前面的做法，分以下几步：

1. 通过拉格朗日乘子法把 m 个约束转换 m 个拉格朗日乘子，得到该问题的拉格朗日函数。
2. 分别对 $\mathbf{w}, b, \xi$ 求偏导，代入拉格朗日函数得到对偶问题。
3. 使用SMO算法求解对偶问题，解出所有样本对应的拉格朗日乘子。
4. 需要进行新样本预测时，使用支持向量及其对应的拉格朗日乘子进行求解。

特别地，因为式（14）有两组各 m 个不等式约束，所以该问题的拉格朗日函数有 $a_i \geq 0$ 和 $\mu_i \geq 0$ 两组拉格朗日乘子。特别地，对松弛变量 ξ 求导，令导数为0会得到一条约束式：

$$C = a_i + \mu_i \qquad (15)$$

有意思的是，软间隔支持向量机的对偶问题和硬间隔几乎没有不同，只是约束条件修改了一下：

$$\max_{\mathbf{a}} \sum_{i=1}^{m} a_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} a_i a_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \quad s.t. \quad \sum_{i=1}^{m} a_i y_i = 0, \quad 0 \leq a_i \leq C, \quad i=1,2,\ldots,m \qquad (16)$$

这里的 a_i 不仅要求大于等于0，还要求小于等于 C。

类似地，由于式（14）的约束条件是不等式约束，所以求解过程要求满足KKT（Karush-Kuhn-Tucker）条件：

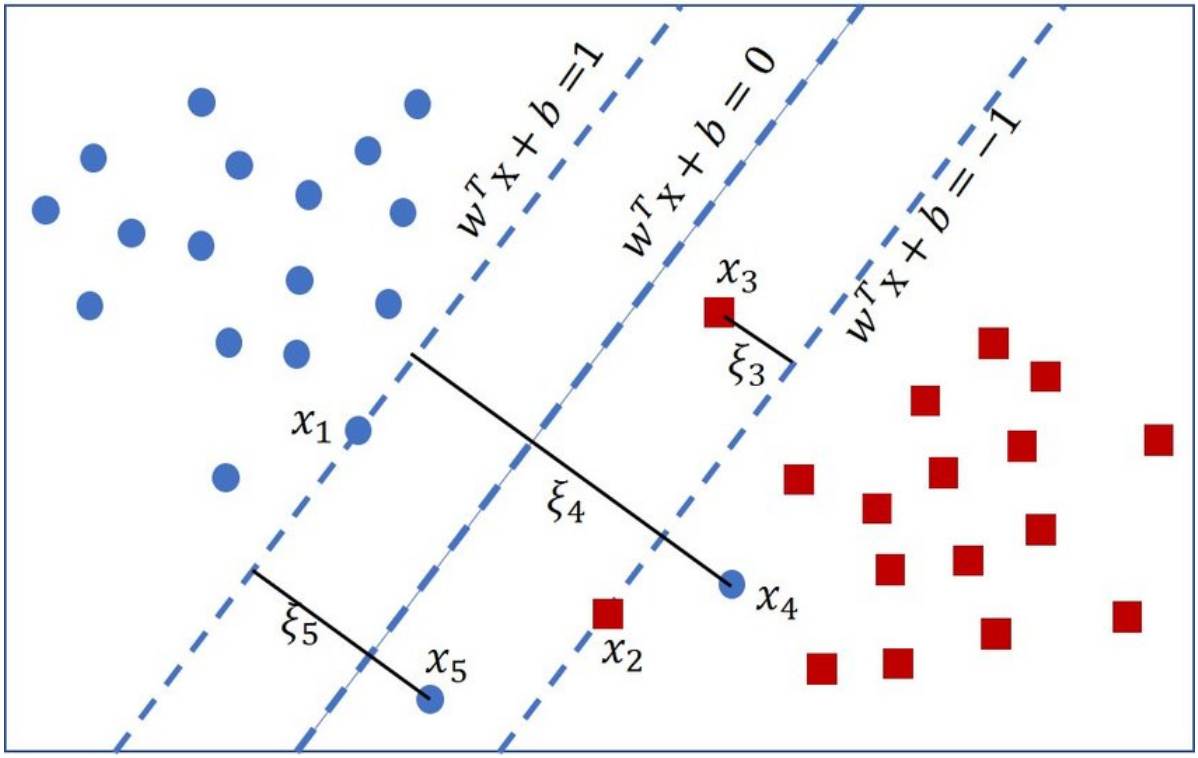
$$\begin{cases} a_i \geq 0; \\ \mu_i \geq 0; \\ y_i f(\mathbf{x}_i) - 1 + \xi_i \geq 0; \\ a_i(y_i f(\mathbf{x}_i) - 1 + \xi_i) = 0; \\ \xi_i \geq 0; \\ \mu_i \xi_i = 0. \end{cases}$$

KKT条件可以理解为下面几点：

- 对任意训练样本，
  - 要么对应的拉格朗日乘子 $a_i = 0$；

    - 要么函数间隔等于1和对应的松弛变量之差（$y_i(\mathbf{w}^T\mathbf{x}+b)=1-\xi_i$）。

- 如果一个样本的拉格朗日乘子 $a_i=0$，则它对模型没有任何影响，不需要保留。
- 如果一个样本的拉格朗日乘子大于0，则它是支持向量。
    - 如果拉格朗日乘子 a_i 小于 C，按照式（15）有 $\mu_i>0$，
      因此松弛变量 $\xi_i=0$，此时函数间隔为1，样本落在最大间隔边界上。
    - 如果拉格朗日乘子 a_i 等于 C，按照式（15）有 $\mu_i=0$，因此松弛变量 $\xi_i>0$。
        - 若 $\xi_i<1$，则样本落在间隔内，但依然被正确分类。
        - 若 $\xi_i>1$，则样本落在另一个类的间隔外，被错误分类



上图就展示了一个典型的软间隔支持向量机。图中就有一些异常点，这些点有的虽然在虚线与超平面之间（$0<y_i(\mathbf{w}^T\mathbf{x}+b)<1$），但也能被正确分类（比如 $\mathbf{x}_3$）。有的点落到了超平面的另一侧，就会被误分类（比如 $\mathbf{x}_4$ h和 $\mathbf{x}_5$）。

特别地，在 R. Collobert. 的论文 Large Scale Machine Learning 中提到，**常数 C 一般取训练集大小的倒数**（$C=\frac{1}{m}$）。

支持向量机和逻辑回归的联系与区别

上面用的是hinge损失，不过我们也提到了还有其他一些替代损失函数，事实上，使用对率损失时，SVM得到的模型和LR是非常类似的。

支持向量机和逻辑回归的**相同点**：

- 都是线性分类器，模型求解出一个划分超平面；
- 两种方法都可以增加不同的正则化项；
- 通常来说性能相当。

支持向量机和逻辑回归的**不同点**：

- LR使用对率损失，SVM一般用hinge损失；
- 在LR的模型求解过程中，每个训练样本都对划分超平面有影响，影响力随着与超平面的距离增大而减小，所以说**LR的解受训练数据本身的分布影响**；SVM的模型只与占训练数据少部分的支持向量有关，所以说，**SVM不直接依赖数据分布**，所得的划分超平面不受某一类点的影响；
- 如果数据**类别不平衡**比较严重，LR需要先做相应处理再训练，SVM则不用；
- SVM依赖于**数据表达的距离测度**，需要先把数据**标准化**，LR则不用（但实际任务中可能会为了方便选择优化过程的初始值而进行标准化）。如果数据的距离测度不明确（特别是高维数据），那么最大间隔可能就变得没有意义；
- LR的输出有**概率意义**，SVM的输出则没有；
- LR可以直接用于**多分类任务**，SVM则需要进行扩展（但更常用one-vs-rest）；
- LR使用的对率损失是光滑的单调递减函数，**无法导出支持向量**，解依赖于所有样本，因此预测开销较大；SVM使用的hinge损失有“零区域”，因此**解具有稀疏性**（书中没有具体说明这句话的意思，但按我的理解是解出的拉格朗日乘子 $\mathbf{a}$ 具有稀疏性，而

不是权重向量 $\mathbf{w}$），从而不需用到所有训练样本。

在实际运用中，LR更常用于大规模数据集，速度较快；SVM适用于规模小，维度高的数据集。

在 Andrew NG 的课里讲到过：

1. 如果Feature的数量很大，跟样本数量差不多，这时候选用LR或者是Linear Kernel的SVM；
2. 如果Feature的数量比较小，样本数量一般，不算大也不算小，选用SVM+Gaussian Kernel；
3. 如果Feature的数量比较小，而样本数量很多，需要手工添加一些feature变成第一种情况。

正则化

事实上，无论使用何种损失函数，SVM的目标函数都可以描述为以下形式：

$$\min_f \Omega(f) + C\sum_{i=1}^{m}\ell(f(\mathbf{x}_i), y_i) \qquad (17)$$

在SVM中第一项用于描述划分超平面的“间隔”的大小，第二项用于描述在训练集上的误差。

更一般地，第一项称为**结构风险（structural risk）**，用来描述**模型的性质**。第二项称为**经验风险（empirical risk）**，用来描述**模型与训练数据的契合程度**。参数 C 用于权衡这两种风险。

前面学习的模型大多都是在最小化经验风险的基础上，再考虑结构风险（避免过拟合）。**SVM却是从最小化结构风险来展开的。**

从最小化经验风险的角度来看，$\Omega(f)$ 表述了我们希望得到具有何种性质的模型（例如复杂度较小的模型），为引入领域知识和用户意图提供了路径（比方说贝叶斯估计中的先验概率）。

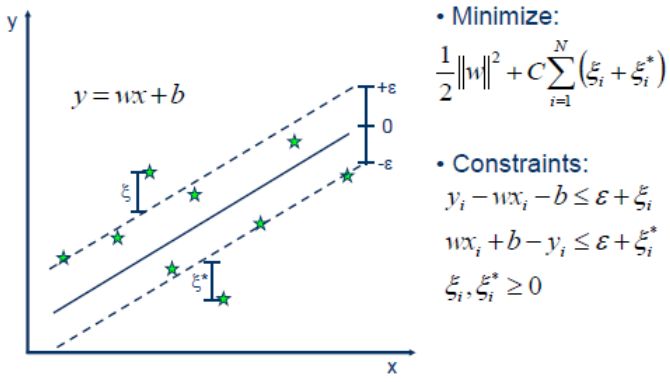
另一方面，$\Omega(f)$ 还可以帮我们削减假设空间，从而降低模型过拟合的风险。从这个角度来看，可以称 $\Omega(f)$ 为**正则化（regularization）项**，C 为正则化常数。正则化可以看作一种**罚函数法**，即对不希望出现的结果施以惩罚，从而使优化过程趋向于期望的目标。

L_p 范数是常用的正则化项，其中 L_2 范数 $\|\mathbf{w}\|_2$ 倾向于 $\mathbf{w}$ 的分量取值尽量稠密，即非零分量个数尽量多； L_0 范数 $\|\mathbf{w}\|_0$ 和 L_1 范数 $\|\mathbf{w}\|_1$ 则倾向于 $\mathbf{w}$ 的分量取值尽量稀疏，即非零分量个数尽量少。

支持向量回归

同样是利用线性模型 $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$ 来预测，回归问题希望预测值和真实值 y 尽可能相近，而不是像分类任务那样，旨在令不同类的预测值可以被划分开。

传统的回归模型计算损失时直接取真实值和预测值的差，**支持向量回归（Support Vector Regression，简称SVR）**则不然。SVR假设我们能容忍最多有 ϵ 的偏差，**只有当真实值和预测值之间相差超出了 ϵ 时才计算损失。**



如图所示，以SVR拟合出的直线为中心，两边各构建出一个宽度为 ϵ 的地带，落在这个**宽度为 2ϵ 的间隔带**内的点都被认为是预测正确的。

因此，问题可以形式化为目标函数：

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\ell_\epsilon(f(\mathbf{x}_i) - y_i) \qquad (18)$$

其中 C 为正则化常数，ℓ_ϵ 称为 ϵ **－不敏感损失（ϵ －insensitive loss）**函数。定义如下：

$$\ell_{、epsilon}(z) = \begin{cases} 0, & if\ |z| \le \epsilon; \\ |z| - \epsilon, & otherwise. \end{cases}$$

引入松弛变量 ξ_i 和 $\hat{\xi}_i$，分别表示**间隔带两侧的松弛程度**，它们**可以设定为不同的值**。此时，目标函数式（18）可以重写为：

$$\begin{aligned}\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}(\xi_i + \hat{\xi}_i) \qquad (19)\\ s.t.\ f(\mathbf{x}_i) - y_i \leq \epsilon + \xi_i,\\ y_i - f(\mathbf{x}_i) \leq \epsilon + \xi_i\\ \xi_i \geq 0, \hat{\xi}_i \geq 0, i = 1,2,\ldots,m.\end{aligned}$$

注意这里有四组 m 个约束条件，所以对应地有四组拉格朗日乘子。

接下来就是用拉格朗日乘子法获得问题对应的拉格朗日函数，然后求偏导再代回拉格朗日函数，得到对偶问题。然后使用SMO算法求解拉格朗日乘子，最后得到模型，这里不一一详述了。

特别地，**SVR中同样有支持向量的概念**，解具有稀疏性，所以训练好模型后不需保留所有训练样本。此外，SVR同样可以通过引入核函数来获得拟合非线性分布数据的能力。

核方法

无论是SVM还是SVR，如果不考虑偏置项b，我们会发现模型总能表示为核函数的线性组合。更一般地，存在**表示定理（representer theorem）**：

令 $\mathbb{H}$ 为核函数 κ 对应的再生希尔伯特空间，$\|h\|_{\mathbb{H}}$ 表示 $\mathbb{H}$ 空间中关于 h 的范数，对于任意**单调递增**函数 $\Omega: [0,\infty] \longmapsto \mathbb{R}$ 和任意**非负**损失函数 $\ell: \mathbb{R}^m \longmapsto [0,\infty]$，优化问题

$$min_{h\in\mathbb{H}} F(h) = \Omega(\|h\|_{\mathbb{H}}) + \ell(h(\mathbf{x}_1, h(\mathbf{x}_2,\ldots,h(\mathbf{x}_m)) \qquad (20)$$

的解总可写为：

$$h^x(\mathbf{x}) = \sum_{i=1}^{m} a_i \kappa(\mathbf{x},\mathbf{x}_i)$$

这个定理表明，对于形如式（20），旨在最小化损失和正则化项之和的优化问题，解都可以表示为核函数的线性组合。

基于核函数的学习方法，统称为**核方法（kernal methods）**。最常见的就是通过**核化**（引入核函数），将线性学习器扩展为非线性学习器。这不仅限于SVM，事实上LR和LDA也都可以采用核函数，只是SVM使用hinge损失，解具有稀疏性所以用得更多。

书中还介绍了如何核化LDA，这部分不作详细记录了。

习题

6.1

> 问：试证明样本空间中任一点$\mathbf{x}$到超平面的距离公式。

6.2

> 问：试使用LIBSVM，在西瓜数据集3.0a上分别用线性核和高斯核训练一个SVM，并比较其支持向量的差别。

6.3

> 问：选择两个UCI数据集，分别用线性核和高斯核训练一个SVM，并与BP神经网络和C4.5决策树进行实验比较。

6.4

> 问：试讨论线性判别分析与线性核支持向量机在何种条件下等价。

6.5

> 问：试述高斯核SVM与RBF神经网络之间的联系。

6.6

> 问：试析SVM对噪声敏感的原因。

6.7

> 问：试给出式(6.52)的完整KKT条件。

6.8

> 问：以西瓜数据集3.0α的“密度”为输入，“含糖率”为输出，试使用LIBSVM训练一个SVR。

6.9

> 问：是使用核技巧推广对率回归，产生“核对率回归”。

6.10*

> 问：试设计一个能显著减少SVM中支持向量的数目而不显著降低泛化性能的方法。

分享一些蛮不错的问题和讲解，我在笔记中参考了部分内容：

- 支持向量机(SVM)是什么意思？
- 支持向量机中的函数距离和几何距离怎么理解？
- 几何间隔为什么是离超平面最近的点到超平面的距离？
- 支持向量机(support vector machine)--模型的由来
- SMO优化算法（Sequential minimal optimization）
- 机器学习有很多关于核函数的说法，核函数的定义和作用是什么？
- Linear SVM 和 LR 有什么异同？
- SVM与LR的比较
- SVM计算最优分类超平面时是否使用了全部的样本数据？
- 现在还有必要对SVM深入学习吗？
- SVM的核函数如何选取？
- 支持向量机通俗导论（理解SVM的三层境界）

贝叶斯分类器

贝叶斯分类器（**Bayes Classifier**）是一种通过最大化后验概率进行单点估计的分类器。

这一章的内容大致如下：

- **贝叶斯决策论**：如何计算某个样本误分类的期望损失/条件风险？贝叶斯判定准则是怎样的？什么是判别式模型？什么是生成式模型？贝叶斯定理中各个概率代表什么？估计后验概率有什么难处？
- **极大似然估计**：如何估计类条件概率？频率学派和贝叶斯学派对参数估计有什么不同的见解？极大似然估计的思想是什么？如何处理概率连成造成的下溢？试想一下连续属性和离散属性的极大似然估计。这种估计方法有什么缺点？
- **朴素贝叶斯分类器**：朴素贝叶斯分类器是基于什么假设的？表达式怎么写？为什么估计概率值时需要进行平滑？拉普拉斯修正是怎样的？现实任务中中如何使用朴素贝叶斯分类器？
- **半朴素贝叶斯分类器**：半朴素贝叶斯分类器是基于什么假设的？什么是独依赖估计？独依赖分类器有哪些学习方法？AODE有什么优点？是否可以通过考虑属性之间的高阶依赖来进一步提升模型的泛化性能？
- **贝叶斯网络**：什么是贝叶斯网络？它的结构是怎样的？如何进行模型的学习？如何对新样本进行推断？
- **EM算法**：什么是隐变量？EM算法的步骤是怎样的？和梯度下降有什么不同？

贝叶斯决策论

贝叶斯决策论（**Bayesian decision theory**）是概率框架下实施决策的基本方法。具体来说，在分类任务中，贝叶斯决策论基于概率和误判损失选择出最优的类别标记。

以多分类任务为例，假设有 N 种标记，即 $\mathcal{Y}=c_1,c_2,\dots,c_N$，用 λ_{ij} 表示把一个真实标记为 c_i 的样本误分类为 c_j 所产生的损失。那么将样本 $\mathbf{x}$ 分类为 c_i 的**期望损失**（**expected loss**）或者说，在样本 $\mathbf{x}$ s上的**条件风险**（**conditional risk**）：

$$R(c_i|\mathbf{x})=\sum_{j=1}^{N}\lambda_{ij}P(c_j|\mathbf{x})$$

它描述的是，给定一个样本 $\mathbf{x}$，把它分类为 c_i 需要冒多大的风险。或者说，当样本真实标记不是 c_i 时，会有多大的损失。这个损失是一个求和，每一个求和项都是某一类别的后验概率和对应误分类损失的积。（**注**：书中这个地方不够细致，求和项的下标是要排除 i 本身的）

在单个样本条件风险的基础上，可以定义**总体风险**：

$$R(h)=\mathbb{E}_{\mathbf{x}}[R(h(\mathbf{x})\mid\mathbf{x})]$$

它描述的是，**所有样本的条件风险的数学期望**。其中 h 是一种用于产生分类结果的判断准则。

那么我们的目标就是找出能最小化总体风险 $R(h)$ 的判断准则。怎样的判断准则能满足这个要求呢？很直观地，如果一个判断准则 h 能最小化所有样本 $\mathbf{x}$ 的条件风险，那它对应的总体风险必然也是最小的。由此，可以得到**贝叶斯判定准则（Bayes decision rule）**：要最小化总体风险，只需**在每个样本上选择能使对应的条件风险 $R(c \mid \mathbf{x})$ 最小的标记**。即：

$$h^*(\mathbf{x}) = \arg\min_{c \in \mathcal{Y}} R(c \mid \mathbf{x})$$

这个判断准则 h^* 称为**贝叶斯最优分类器（Bayes optimal classifier）**，对应的总体风险 $R(h^*)$ 称为**贝叶斯风险（Bayes risk）**，而 $1 - R(h^*)$ 则反映了分类器所能达到的最好性能，也即**模型精度的理论上限**。

进一步地，如果我们学习模型的目标是**令分类错误率最小**，那么分类正确时误分类损失 λ_{ij} 为0，反之为1。这是条件风险就是：

$$R(c \mid \mathbf{x}) = 1 - P(c \mid \mathbf{x})$$

要令风险最小，我们只需要选择使样本 $\mathbf{x}$ 后验概率最大的一个类别标记就可以了。

问题在于，**怎样获取后验概率呢？**

事实上，从概率的角度来理解，机器学习的目标就是**基于有限的训练样本集尽可能准确地估计出后验概率**（当然，大多数机器学习技术无需准确估计出后验概率）。要实现这个目标，主要有两种策略：

- 构建**判别式模型（discriminative models）**：给定样本 $\mathbf{x}$，直接对后验概率 $P(\mathbf{x} \mid c)$ 建模来预测 c。这类模型包括决策树、BP神经网络、支持向量机等等。
- 构建**生成式模型（generative models）**：给定样本 $\mathbf{x}$，先对联合概率分布 $P(\mathbf{x}, c)$ 建模，然后再利用联合概率计算出后验概率 $P(c \mid \mathbf{x})$，也即 $P(c \mid \mathbf{x}) = \frac{P(\mathbf{x},c)}{P(\mathbf{x})}$。

又因为联合概率 $P(\mathbf{x}, c) = P(c \mid \mathbf{x}) \times P(\mathbf{x}) = P(\mathbf{x} \mid c) \times P(c)$，由此，能得到**贝叶斯定理**：

$$P(c \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c) \times P(c)}{P(\mathbf{x})}$$

在贝叶斯定理中，每个概率都有约定俗成的名称：

- $P(c \mid \mathbf{x})$ 是类标记 c 相对于样本 $\mathbf{x}$ 的条件概率，也由于得自 $\mathbf{x}$ 的取值而被称作 c 的后验概率。
- $P(\mathbf{x} \mid c)$ 是样本 $\mathbf{x}$ 相对于类标记 c 的**类条件概率（class-conditional probability）**，或称为**似然（likelihood）**，也由于得自 c 的取值而被称作 $\mathbf{x}$ 的后验概率。
- $P(c)$ 是 c 的先验概率（也称为边缘概率），之所以称为"先验"是因为它不考虑任何 $\mathbf{x}$ 方面的因素。在这里又称为**类先验（prior）概率**。
- $P(\mathbf{x})$ 是 $\mathbf{x}$ 的先验概率。在这里是用作归一化的**证据（evidence）因子**，与类标记无关。

有了贝叶斯定理，如何估计后验概率 $P(c \mid \mathbf{x})$ 的问题就转化为**如何计算类先验概率 $P(c)$ 和类条件概率 $P(\mathbf{x} \mid c)$** 了。

类先验概率 $P(c)$ 表示的是**样本空间中各类样本的比例**，根据大数定律，**当训练集包含足够多的独立同分布样本**时，类先验概率可以直接通过**训练集中各类样本出现的频率**进行估计。

类条件概率 $P(\mathbf{x} \mid c)$ 的情况就复杂多了，它涉及到类 c 中**样本 $\mathbf{x}$ 所有属性的联合概率**，假设每个样本有 d 个二值属性，那么可能的取值组合就多达 2^d 个，这个数目可能**远多于训练集的规模**，也就意味着很多样本的取值没有在训练集中出现，所以**直接用训练集出现的频率进行估计是不可行的**。必须注意**未被观测到**和**出现概率为0**的区别。

注意，上述讨论中，均假设属性是离散型，对于连续型属性，只需把概率质量函数 $P(\cdot)$ 换为概率密度函数 $p(\cdot)$ 就可以了。

极大似然估计

估计类条件概率的一种常用策略是：先**假定该类样本服从某种确定的概率分布形式**，然后再**基于训练集中的该类样本对假定的概率分布的参数进行估计**。比方说假定该类样本服从高斯分布，那么接下来就是利用训练集中该类样本来估计高斯分布的参数——均值和方差。

具体来说，如果类 c 的样本服从参数为 θ_c（可能不止一个参数）的分布，那么我们从样本空间抽取到该类的某一个样本 $\mathbf{x}$ 的概率就是 $P(\mathbf{x} \mid \theta_c)$。使用 D_c 来表示训练集中类 c 的子集，可以定义数据集 D_c 的**似然（likelihood）**为：

$$P(D_c \mid \theta_c) = \prod_{\mathbf{x} \in D_c} P(\mathbf{x} \mid \theta_c)$$

由于**连乘操作容易造成下溢**，实际任务中通常使用**对数似然（log-likelihood）**代替：

$$LL(\theta_c) = \log P(D_c \mid \theta_c) = \sum_{\mathbf{x} \in D_c} \log P(\mathbf{x} \mid \theta_c)$$

所谓**极大似然估计（Maximum Likelihood Estimation，简称MLE）**就是**找出令似然最大的参数** θ_c。也即从 θ_c 的所有可能取值中找到一个**令所抽取样本出现的可能性最大**的值。

求解的过程也很简单，就是求似然函数的导数，令导数为0，得到**似然方程**，解似然方程得到最优解，也即该类样本分布的参数。

特别地，对于参数估计，**频率主义学派（Frequentist）**和**贝叶斯学派（Bayesian）**有不同的见解。前者认为，参数虽然未知，却是**客观存在的固定值**，因此可以用优化似然函数等准则确定参数值；后者认为，参数是**未观测到的随机变量，参数本身也存在分布**。所以可以先假定参数服从一个先验分布，然后再根据观测到的数据计算参数的后验分布。这一节讨论的极大似然估计方法源于频率主义学派。

尽管极大似然估计能使我们求取类条件概率的过程变得相对简单，但它有最大的一个缺点就是：估计结果的**准确性严重依赖于所假设的概率分布形式是否符合潜在的真实数据分布**。在实际任务中，我们需要利用任务所属领域的一些经验知识，全凭猜测是很容易产生误导性结果的。

P.S. 关于最大似然、最大后验与贝叶斯估计的爱恨纠缠可以看我写的另外一篇文章参数估计：最大似然，最大后验与贝叶斯。

朴素贝叶斯分类器

前面提到了，估计后验概率 $P(c \mid \mathbf{x})$ 最大的一个难处是：类条件概率 $P(\mathbf{x} \mid c)$ 是所有属性上的联合概率，而多个属性的不同属性值组合并不一定全部囊括在训练集内，所以很难通过训练集估计。

为了避免这个障碍，**朴素贝叶斯分类器（naive Bayes clssifier）**采用**属性条件独立性假设（attribute conditional independence assumption）**。也就是说，假设**所有属性相互独立，单独地对分类结果产生影响**。

基于这个假设，可以把类条件概率写成连乘的形式，因此贝叶斯定理可重写为：

$$P(c \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid c) \times P(c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^{d} P(x_i \mid c) \qquad (1)$$

其中 d 为属性数目， x_i 为样本 $\mathbf{x}$ 在第 i 个属性上的取值。

又因为 $P(\mathbf{x})$ 与类别无关，所以**朴素贝叶斯分类器的表达式**可以写为：

$$h(\mathbf{x}) = \arg\max_{c \in \mathcal{Y}} P(c) \prod_{i=1}^{d} P(x_i \mid c)$$

前面已经提到过，当训练集包含足够多独立同分布样本时，类先验概率 $P(c)$ 可以直接算出，也即训练集该类样本的数目占训练集规模的比例：

$$P(c) = \frac{|D_c|}{|D|} \qquad (2)$$

而条件概率 $P(x_i \mid c)$，根据属性类型分离散和连续两种情况：

- 离散型属性：条件概率 $P(x_i \mid c)$ 可以估计为，在类别 c 的样本子集中，第 i 个属性取值为 x_i 的样本所占的比例：

$$P(x_i \mid c) = \frac{|D_{c,x_i}|}{|D_c|} \qquad (3)$$

- 连续性属性：替换为概率密度函数，假设第 i 个属性服从高斯分布，那么条件概率就写成 $p(x_i \mid c) \sim \mathcal{N}(\mu_{c,i}, \sigma_{c,i}^2)$。我们利用类别 c 的样本子集在该属性上的取值算出分布的均值和方差，然后把属性取值 x_i 代入概率密度函数就可算出这个条件概率。

平滑

注意了，若**某个属性值在训练集中没有与某个类同时出现过**，那么它对应的条件概率 $P(x_i \mid c)$ 就为0。在连乘中，这就意味着整个式子值为0了，**其他属性携带的信息都被抹去了**。这是很常见的情况，举个例子，假设有一篇新闻应该在体育版发布的，它包含了 “罗纳尔多” 这个词，但由于我们构造分类器时，训练集中所有 “体育” 类的文本都没有出现这个词，于是，该新闻按照式（1）计算出的体育类的条件概率必定为0；而恰好 “娱乐” 类的文本中有一篇包含了这个词，那么计算出的娱乐类的条件概率就大于0，从而使得这篇新闻被误分到娱乐版发布了，这显然很不合理。

此时，我们就需要对概率值进行**平滑（smoothing）**了，最常用的是**拉普拉斯修正（Laplacian correction）**，假设**训练集中**包含 N 个类别，第 i 个属性包含 N_i 种取值，则拉普拉斯修正把式（2）和式（3）修改为：

$$P(c) = \frac{|D_c| + 1}{|D| + N} \qquad (4)$$

$$P(x_i \mid c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i} \qquad (5)$$

再回想一下上面新闻分类的例子，尽管所有 “体育” 类的文本都没有出现 “罗纳尔多” 这个词，但使用拉普拉斯修正后，这个词（文本分类中每个词是一个属性）对应的条件概率就不为0了，而是一个很小的值；而该新闻的其他词，比如 “足球”、“球场” 等等在体育类新闻中都出现得很频繁，所以最后累乘计算出的体育类的类条件概率就大于其他类，从而能正确地进行划分了。

拉普拉斯修正保证了**不会因为训练集样本不充分导致概率估值为零**。但它实际上是假设了**类别和属性值是均匀分布的**，相当于额外引入了先验，这个**假设并不总是成立**。不过当训练集规模足够大时，引入先验所产生的影响会变得非常低。也可以理解为，此时式（4）和式（5）的分母很大，使得分子中引入的1带来的变化非常小，此时概率的估计值会趋向于真实值。

实际使用

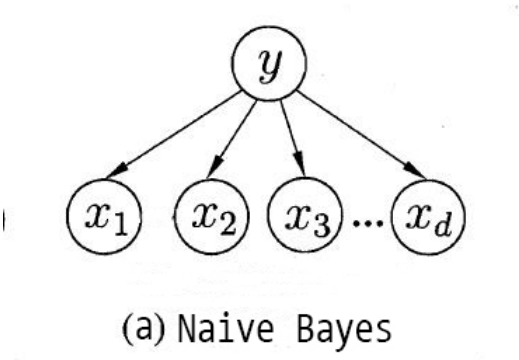
朴素贝叶斯分类器和前面学习的模型有一个不同的地方就是，我们并不是基于训练集和某些算法来学习模型的参数；而是利用训练集来算出一些概率，在预测时，根据新样本的情况，使用不同的概率计算出它被分到各个类的后验概率，然后取后验概率最大的一个类作为结果。

在实际任务中，有两种使用方式：

- **查表**：若对预测速度要求较高，可以先根据训练集把所有涉及到的概率计算出来，然后存储好，在预测新样本时只需要查表然后计算就可以了。
- **懒惰学习**：若数据更替比较频繁，也可以理解为用训练集算出的概率可能很快就失效了，更新换代的速度很快，那就采取**懒惰学习（lazy learning）**的方式，仅当需要预测时才计算涉及到的概率。

特别地，当我们采取了预先计算所有概率的方式时，如果有新数据加入到训练集，我们只需要更新新样本涉及到的概率（或者说计数）就可以了，可以很方便地实现**增量学习**。

半朴素贝叶斯分类器



(a) Naive Bayes

朴素贝叶斯分类器基于属性条件独立性假设，每个属性仅依赖于类别，如上图。但这个假设往往很难成立的，有时候**属性之间会存在依赖关系**，这时我们就需要对属性条件独立性假设适度地进行放松，**适当考虑一部分属性间的相互依赖信息**，这就是**半朴素贝叶斯分类器（semi-naive Bayes classifier）**的基本思想。

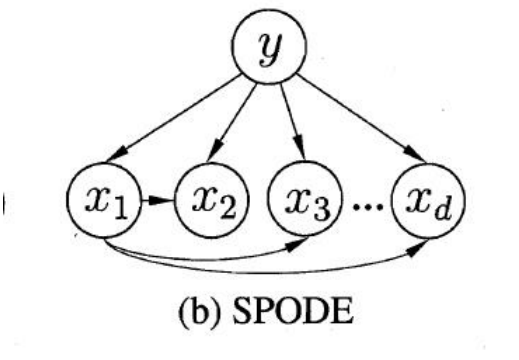
独依赖估计（One-Dependent Estimator，简称ODE）是半朴素贝叶斯分类器最常用的一种策略，它假设的是**每个属性在类别之外最多仅依赖于一个其他属性**。也即：

$$P(c \mid \mathbf{x}) \propto P(c) \prod_{i=1}^{d} P(x_i \mid c, pa_i)$$

其中 pa_i 是属性 x_i 依赖的另一属性，称为 x_i 的**父属性**。若已知父属性，就可以按式（5）来计算了。现在问题转化为**如何确定每个属性的父属性**？

这里介绍两种产生独依赖分类器的方法：

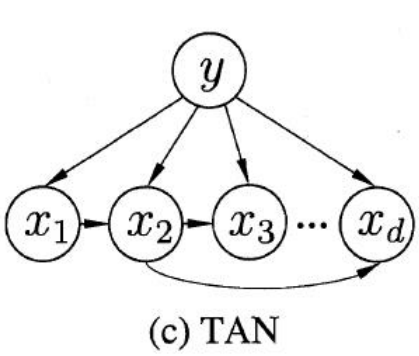
SPODE



(b) SPODE

在**SPODE（Super-Parent ODE）**中，所有属性都依赖于一个共同的属性，称为**超父（super-parent）**，比方说上图中的 x_1。可以通过交叉验证等模型选择方法来确定最合适的超父属性。

TAN



(c) TAN

TAN（Tree augmented naive Bayes）则是一种基于**最大带权生成树（maximum weighted spanning tree）**的方法，包含以下四个步骤：

1. 计算任意两个属性间的**条件互信息（conditional mutual information）**：

$$I(x_i, x_j \mid y) = \sum_{x_i, x_j; c \in \mathcal{Y}} P(x_i, x_j \mid c) \log \frac{P(x_i, x_j \mid c)}{P(x_i \mid c) P(x_j \mid c)}$$

2. 以属性为节点构建完全图，每条边的权重为对应的条件户信息。

3. 构建此完全图的最大带权生成树。选一个节点作为根节点，把边转换为有向边。

4. 加入类别节点 y，并增加从 y 到每个属性的有向边。

AODE

特别地，有一种更为强大的独依赖分类器——**AODE(Average One-Dependent Estimator)**，它基于集成学习机制。无须通过模型选择来确定超父属性，而是尝试将每个属性都作为超父属性来构建模型，然后把有足够训练数据支撑的SPODE模型集成起来导出最终结果。

类似于朴素贝叶斯分类器，AODE**无需进行模型选择**，既可以**通过预计算来节省预测时间**，又可以**采取懒惰学习**，需要预测时再进行计数，并且可以容易地实现增量学习。

高阶依赖

ODE假设每个属性最多依赖类别以外的另一属性，但如果我们继续放宽条件，**允许属性最多依赖类别以外的 k 个其他属性**，也即考虑属性间的**高阶依赖**，那就得到了 kDE。

是不是考虑了高阶依赖就一定能带来性能提升呢？并不是这样的。随着 k 的增加，要准确估计条件概率 $P(x_i \mid c, \mathbf{pa}_i)$ **所需的训练样本会呈指数上升**。如果训练样本不够，很容易陷入高阶联合概率的泥沼；但如果训练样本非常充足，那就有可能带来泛化性能的提升。

贝叶斯网

贝叶斯网（Bayesian network）亦称**信念网（belief network）**，它借助**有向无环图（Directed Acyclic Graph，简称DAG）**来刻画属性之间的依赖关系，并使用**条件概率表（Conditional Probability Table，简称CPT）**来描述属性的联合概率分布。

贝叶斯网的学习包括结构的学习和参数的学习，而预测新样本的过程则称为**推断（inference）**。这部分内容设计到一些后面章节，相对复杂一些，所以暂且放下，之后有时间再写详细的笔记。

EM算法

前面讨论的极大似然估计方法是一种常用的参数估计方法，它是假设分布的形式，然后用训练样本来估计分布的参数。但实际任务中，我们遇到一个很大的问题就是**训练样本不完整**。这时就需要用到**EM（Expectation-Maximization）算法**了。

所谓不完整的样本，说的是这个样本某些属性的值缺失了。将每个属性的取值看为一个变量，那么缺失的就可以看作“未观测”变量，比方说有的西瓜根蒂脱落了，没办法看出根蒂是“蜷缩”还是“硬挺”，那么这个西瓜样本的根蒂属性取值就是一个“未观测”变量，更准确地说，称作**隐变量（latent variable）**。

整个训练集可以划分为已观测变量集 X 和隐变量集 Z 两部分。按照极大似然的思路，我们依然是想找出令训练集被观测到的概率最大的参数 Θ。也即最大化对数似然：

$$LL(\Theta \mid X, Z) = \ln P(X, Z \mid \Theta)$$

但是，由于 Z 是隐变量，我们没办法观测到，所以上面这个式子实际是没法求的。

怎么办呢？EM算法的思路很简单，步骤如下：

1. 设定一个初始的 Θ
2. 按当前的 Θ 推断隐变量 Z 的（期望）值
3. 基于已观测变量 X 和 步骤2得到的 Z 对 Θ 做最大似然估计得到新的 Θ
4. 若未收敛（比方说新的 Θ 与旧的 Θ 相差仍大于阈值），就回到步骤2，否则停止迭代

EM算法可以看作是用**坐标下降（coordinate descent）**法来最大化对数似然下界的过程，每次固定 Z 或者 Θ 中的一个去优化另一个，直到最后收敛到局部最优解。

理论上，用梯度下降也能求解带隐变量的参数估计问题，但按我的理解，由于隐变量的加入，使得求导这个步骤非常困难，计算也会随着隐变量数目上升而更加复杂，EM算法避免了这些麻烦。

补充内容

朴素贝叶斯分类器的属性条件独立性假设在现实中很难成立，但事实上它在大多数情形下都有不错的性能。关于这点，有以下两种解释：

1. 对分类任务来说，只需各类别的条件概率**排序正确**，即使概率值不准确，也可以产生正确的分类结果；
2. 若属性间的相互依赖对所有类别影响都相同，或者依赖关系互相抵消，则属性条件独立性假设在降低开销的同时不会给性能带来负面影响；

注意，本章讨论的**贝叶斯分类器**和一般意义上的**贝叶斯学习（Bayesian learning）**是有很大差别的，本章讨论的贝叶斯分类器只是通过最大化后验概率来进行**单点估计**，获得的仅仅是一个数值；而贝叶斯学习则是进行**分布估计**或者说**区间估计**，获得的是一个分布。

习题

7.1

> 问：试使用极大似然法估算西瓜数据集3.0中前3个属性的类条件概率。

7.2*

> 问：试证明：条件独立性假设不成立时，朴素贝叶斯分类器仍有可能产生最优贝叶斯分类器。

7.3

> 问：试编程实现拉普拉斯修正的朴素贝叶斯分类器，并以西瓜数据集3.0为训练集，对 page.151 的 “测1” 样本进行判别。

7.4

> 问：实践中使用式（7.15）决定分类类别时，若数据的维数非常高，则概率连乘 $\prod_{i=1}^{d} P(x_i|c)$ 的结果通常会非常接近于0从而导致下溢。试述防止下溢的可能方案。

7.5

> 问：试证明：二分类任务中两类数据满足高斯分布且方差相同时，线性判别分析产生贝叶斯最优分类器。

7.6

> 问：试编程实现AODE分类器，并以西瓜数据集3.0为训练集，对 page.151 的 “测1” 样本进行判别。

7.7

> 问：给定d个二值属性的二分类任务，假设对于任何先验概率项的估算至少需30个样例，则在朴素贝叶斯分类器式（7.15）中估算先验概率项 $P(c)$ 需 $30 \times 2 = 60$ 个样例。试估计在AODE式（7.23）中估算先验概率项 $P(c, x_i)$ 所需的样例数（分别考虑最好和最坏情形）。

7.8

> 问：考虑图7.3，试证明：在同父结构中，若 x_1 的取值未知，则 $x_3 \perp\!\!\!\perp x_4$ 不成立；在顺序结构中，$y \perp z|x$，但 $y \perp\!\!\!\perp z$ 不成立。

7.9

> 问：以西瓜数据集2.0为训练集，试基于BIC准则构建一个贝叶斯网。

7.10

> 问：以西瓜数据集2.0中属性 “脐部” 为隐变量，试基于EM算法构建一个贝叶斯网。

集成学习

集成学习（ensemble learning）通过构建并结合多个学习期来完成学习任务，有时也被称为**多分类器系统（multi-classifier system）**、**基于委员会的学习（committee-based learning）**等。

这一章的内容大致如下：

- **个体与集成**：同质集成和异质集成有什么不同？集成学习对个体学习器有什么要求？集成学习研究的核心是什么？集成学习分哪两大类？
- **Boosting**：Boosting的基本概念？AdaBoost算法的流程？如何基于加性模型最小化指数损失函数来推导？怎样处理基学习算法无法处理带权样本的情况？Boosting的优势是什么？
- **Bagging与随机森林**：Bagging算法如何获得各个基学习器的训练集？预测时如何进行结合？相比AdaBoost有什么优势？随机森林的核心思想是什么？相比Bagging有什么优势？
- **结合策略**：结合多个学习器有什么好处？平均法是怎样的？投票法是怎样的？学习法又是怎样的？
- **多样性**：如何进行误差-分歧分解？说明了什么？有哪些多样性度量指标？有哪些增强多样性的手段？

个体与集成

集成学习先产生一组**个体学习器（individual learner）**，再用某种策略将它们结合起来。

当所有个体学习器都由同样的学习算法生成时，也即集成中只包含同种类型的个体学习器时，称为**同质（homogeneous）**集成，这些个体学习器又被称为**基学习器（base learner）**，相应的学习算法称为**基学习算法（base learning algorithm）**；

当个体学习器由不同的学习算法生成时，称为**异质（heterogenous）**集成，这些个体学习器称为**组件学习器（component learner）**或直接称为个体学习器。

集成学习通过结合多个学习器，通常能获得比单一学习器更优越的泛化性能，对**弱学习器（weak learner）**的提升尤为明显。
注：弱学习器即略优于随机猜测的学习器，例如二分类任务中精度略高于50%。虽然理论上，集成弱学习器已经能获得很好的性能。但现实任务中，人们往往会使用比较强的学习器。

当我们使用集成学习的时候，我们其实是希望不同的个体学习器产生互补的效果，使得某个学习器错误的地方能有机会被其他学习器纠正过来，这就要求个体学习器应该尽量“不同”，即具有多样性；同时我们希望这种互补不会把正确改为错误，这就要求个体学习器应该尽量“准确”。合起来就是集成学习研究的核心——**如何产生并结合“好而不同”的个体学习器**。

根据个体学习器的**生成方式**，集成学习方法大致可分为两个大类：

- 个体学习器间存在强依赖关系，必须串行生成的序列化方法，例如：Boosting算法族；
- 个体学习器间不存在强依赖关系，可同时生成的并行化方法，例如：Bagging，随机森林；

习题

8.1

> 问：假设抛硬币正面朝上的概率为 p，反面朝上的概率为 $1-p$。令 $H(n)$ 代表抛 n 次硬币所得正面朝上的次数，则最多 k 次正面朝上的概率为

$$P(H(n)\leq k)=\sum_{i=0}^{k}\binom{n}{i}p^i(1-p)^{n-i}$$

> 对 $\delta>0,\quad k=(p-\delta)n$，有 Hoeffding 不等式

$$P(H(n)\leq (p-\delta)n)\leq e^{-2\delta^2 n}$$

> 试推导出式（8.3）

8.2

> 问：对于 0/1 损失函数来说，指数损失函数并非仅有的一直替代函数，考虑式（8.5），试证明：任意损失函数 $\ell(-f(\mathbf{x}H(\mathbf{x})))$，若对于 $H(\mathbf{x})$ 在区间 $[-\inf,\delta](\delta>0)$ 上单调递减，则 ℓ 是 0/1 损失函数的一致替代函数。

8.3

> 问：从网上下载或自己编程实现 Adaboost，以不剪枝决策树为基学习器，在西瓜数据集 3.0α 上训练一个 AdaBoost 集成，并与图8.4进行比较。

8.4

> 问：GradientBoosting 是一种常用的 Boosting 算法，试析其与 AdaBoost 的异同。

8.5

> 问：试编程实现 Bagging，以决策树桩为基学习器，在西瓜数据集 3.0α 上训练一个 Bagging 集成，并与图8.6进行比较。

8.6

> 问：试析 Bagging 通常为何难以提升朴素贝叶斯分类器的性能。

8.7

> 问：试析随即森林为何比决策树 Bagging 集成的训练速度更快。

8.8

> 问：MultiBoosting 算法将 AdaBoost 作为 Bagging 的基学习器，Iterative Bagging 算法则是将 Bagging 作为 AdaBoost 的基学习器，试比较二者的优缺点。

8.9*

> 问：试设计一种可视的多样性度量，对习题8.3和习题8.5中得到的集成进行评估，并与 κ-误差图比较。

8.10*

> 问：试设计一种能提升 k 近邻分类器性能的集成学习算法。

聚类

在无监督学习（**unsupervised learning**）中，训练样本的标记信息是未知的，学习的目的是揭示数据的内在性质及规律，为进一步的数据分析提供基础。这类学习任务中研究最多，应用最广的是聚类（**clustering**）。

这一章的内容大致如下：

- **聚类任务**：聚类过程是怎样的？聚类有什么用途？聚类的两个基本问题是什么？
- **性能度量**：聚类的目标是什么？聚类性能度量的两大类指什么？各包含哪些度量指标？
- **距离计算**：距离度量需要满足哪些基本性质？怎样度量有序属性？怎样度量无序属性？相似度度量和距离度量有什么区别？
- **原型聚类**：什么是原型聚类？k均值算法是怎样的？学习向量量化算法是怎样的？高斯混合聚类是怎样的？
- **密度聚类**：什么是密度聚类？DBSCAN算法是怎样的？
- **层次聚类**：什么是层次聚类？AGNES算法是怎样的？

聚类任务

聚类试图将数据集中的样本划分为若干个通常是不相交的子集，每个子集称为一个**簇**（**cluster**）。通常来说每个簇可能对应一些特征，比方说音乐可以聚类成古典音乐、摇滚乐、流行乐等等。但**聚类过程仅产生簇结构，簇对应的概念语义需要使用者来把握和定名**。

简单来说，聚类可以分为两种用途：

- 作为一个单独过程，用于寻找数据内在的分布结构；
- 作为其他学习任务的前驱过程，比方说根据聚类结果定义类标记，然后再进行分类学习；

聚类算法的两大基本问题是性能度量和距离计算。

习题

9.1

> 问：试证明：$p \geq 1$ 时，闵可夫斯基距离满足距离度量的四条基本性质；$0 \leq p \leq 1$ 时，闵可夫斯基距离不满足直递性，但满足非负性、同一性、对称性；p 趋向无穷大时，闵可夫斯基距离等于对应分量的最大绝对距离，即

$$\lim_{p \to +\inf} (\sum_{u=1}^{n} |x_{iu} - x_{ju}|^p)^{\frac{1}{p}} = \max_u |x_{iu} - x_{ju}|$$

9.2

> 问：同一样本空间中的集合 X 与 Z 之间的距离可通过“豪斯多夫距离”（Hausdorff distance） 计算：

$$dist_H(X, Z) = \max(dist_h(X, Z), dist_h(Z, X))$$

> 其中

$$dist_h(X, Z) = \max_{x \in X} \min_{z \in Z} \|\mathbf{x} - \mathbf{z}\|_2$$

> 试证明：豪斯多夫距离满足距离度量的四条基本性质。

9.3

> 问：试析 k 均值算法能否找到最小化式（9.24）的最优解。

9.4

> 问：试编程实现 k 均值算法，设置三组不同的 k 值、三组不同初始中心点，在西瓜数据集4.0上进行实验比较，并讨论怎样的初始中心有利于取得好结果。

9.5

> 问：基于 DBSCAN 的概念定义，若 $\mathbf{x}$ 为核心对象，由 $\mathbf{x}$ 密度可达的所有样本构成的集合为 X，试证明：X满足连接性（9.39）与最大性（9.40）。

9.6

> 问：试析 AGNES 算法使用最小距离和最大距离的区别。

9.7

> 问：聚类结果中若每个簇都有一个凸包（包含簇样本的凸多面体），且这些凸包不相交，则称为凸聚类。试析本章介绍的哪些聚类算法只能产生凸聚类，哪些能产生非凸聚类。

9.8

> 问：试设计一个聚类性能度量指标，并与9.2节中的指标比较。

9.9*

> 问：试设计一个能用于混合属性的非度量距离。

9.10*

> 问：试设计一个能自动确定聚类数的改进 k 均值算法，编程实现并在西瓜数据集4，0上运行。