NTUML 1. 学习问题

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/08/01/NTUML-1-the-Learning-Problem/

机器学习的概念

我们可以从人类的学习思维入手。人类的学习过程，是从观察出发，经过大脑内化以后，变成有用的技巧。机器学习，类似地，是我们希望能让电脑模拟类似的过程。这时，电脑的观察到的东西被称作是**数据**，而思考过程实际上是**计算过程**，技巧则是**增强某一方面的表现**。因此，

> 机器学习的过程是从数据出发，经过计算过程以后，最终获得某种表现上的增进

那么为什么需要机器学习呢？想象如下的例子，给定一张照片，判断照片里的物体是不是一棵（大自然中的）树。如果我们不使用机器学习算法，就需要对“什么是树”做一个回答，给出树的定义，并且动手将这个定义实现为程序。传统的做法是按照规则进行判断，而将规则表述出来是很难的。然而，我们认识树的方法其实也是通过观察，经过经验的积累判断这个是树或者不是，并不是教条地从长辈那里学习判断规则。类似地，我们也可以编写代码，让机器自己从数据中学习树的判断方法。因此，**机器学习是构建复杂系统的另一种方法**

机器学习在以下情况下尤其适用

- 当我们不能提前想好各种情况，手工编码规则时。例如要让机器人在火星上导航，而我们不可能提前想到它在火星上会遇见什么样的情况
- 当我们无法容易地定义问题的解决方案时。例如要做语音识别/视觉识别，而我们无法对音频信号做出准确定义
- 当人们需要做出快速决策时。例如高频交易
- 当要让机器服务于海量使用者时。例如做服务个性化定制

因此，我们可以从三个关键点进行判断，看是否适合使用机器学习

1. 问题应该是“可以学习的”，即存在一些潜在的模式，以及目标
2. 这些规则难以清晰定义
3. 手里掌握了对应的数据

机器学习的应用

机器学习目前在衣食住行四个方面都得到了广泛应用

- 衣：Abu-Mostafa 2012利用销售数据和对用户的调研结果构建推荐系统给用户推荐穿搭
- 食：Sadilek et al. 2013利用机器学习，以推特上的文本和地理位置信息为数据，判断餐厅的卫生状况
- 住：Tsanas and Xifara 2012利用已有房间的特点和耗能，预测房屋的能源消耗
- 行：Stallkamp et al. 2012利用交通标志照片和对应的意义，来提升认识交通标志的准确率

此外还有两个领域：教育和娱乐

- 教育：系统根据学生的答题状况，有针对地提供题目让学生练习其薄弱的部分，同时将太难的题推后给出。即，给定一名学生的答题历史和一个题目，预测学生是否能作对这道题（KDDCup 2010）
- 娱乐：系统根据用户的历史打分，预测用户对新电影的打分（KDDCup 2011）

机器学习的过程

问题背景

以银行信用卡发卡这一问题为例。假设银行收集了一些用户的基本信息，例如下表

项目	值
年龄	23岁

项目	值
性别	女
年薪	20万人民币
在所在地居住年数	1
工龄	0.5
负债额	4万人民币

银行要解决的问题是，对于这样的客户，是否应该给她发放信用卡

问题的形式化描述

为了更加形式化地描述这个问题，我们需要定义一些符号：

- **输入**：$\mathbf{x} \in \mathcal{X}$，例如上面的这些基本信息
- **输出**：$y \in \mathcal{Y}$，是我们期望得到的答案。例如在上面的问题中就是“发”或“不发”
- **目标函数**：$f : \mathcal{X} \to \mathcal{Y}$，是我们期望学到，但是目前不知道的东西。是最理想的公式
- **数据**：$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$，是之前积累的记录
- **假设**：$g : \mathcal{X} \to \mathcal{Y}$，是机器从数据中学到的函数。我们通常都希望$g$的表现足够好，即$g \approx f$。注意这里$g$不一定等于$f$（事实上，我们永远也不知道真正的$f$是什么样子，只知道由$f$产生的数据$\mathcal{D}$）
- **机器学习算法**：$\mathcal{A}$，是由$\mathcal{D}$产生g的算法，可以理解为$\mathcal{A}$会从各种不同假设h_k（这里h_k有好有坏）构成的集合$\mathcal{H}$中挑选出来一个最好的g，使得$g \approx f$。即$\mathcal{A}$以$\mathcal{D}$和$\mathcal{H}$为输入，以g为输出。

我们所讲的机器学习模型，指的就是$\mathcal{A}$和$\mathcal{H}$

在有了这些记号以后，我们可以重新给机器学习下一个定义

> 机器学习是使用数据计算假设g以逼近目标函数f的过程

机器学习与其它名词

机器学习与数据挖掘

数据挖掘的一个简单定义是使用海量数据中以找出一些有趣的现象或性质。这里，如果“有用的性质”就是“能够逼近目标函数的假设”，那么数据挖掘和机器学习是没有区别的。假如这两个概念只是有关联，那么这两者是相辅相成的关系

传统上的数据挖掘还关注如何在大的数据库中进行有效计算。不过现在已经很难将机器学习和数据挖掘这两个概念分开了

机器学习与人工智能

人工智能要求计算机呈现出一些智能的行为。由于机器学习逼近目标函数的过程就展现了一些智能，因此我们可以说，机器学习是实现人工智能的一种手段。

机器学习与统计学

统计学是要使用数据做出推论，推测一些我们本来不知道的事实。考虑到假设g是推论结果，f是不知道的事情，那么可以说统计是实现机器学习的一种方法。但是传统统计学从数学出发，很多工具是为数学假设提供证明和推论。而机器学习看重的是如何算出结果。总而言之，统计学为机器学习提供了很多有力的工具

NTUML 2. 学习判断是与非

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/08/03/NTUML-2-Learning-to-Answer-Yes-No/

感知机假设集合

上回说到机器学习的核心就是，使用算法$\mathcal{A}$接收数据$\mathcal{D}$，从假设集合（所有可能性）$\mathcal{H}$中选出一个g，希望$g \approx f$。那么我们现在最关心就是，$\mathcal{H}$应该是什么样的。

以之前提到的银行审核发放信用卡的场景为例，假设我们把每个使用者定义为向量$\mathbf{x}$，包含d个维度，例如x_1代表年龄，x_2代表年薪，等等。我们可以将这些维度（因素）综合起来给使用者一个整体分数。如果这个分数超过了某个标准，那就给ta发放；否则拒绝发放。这样，我们需要给每个$x_i, i \in \{1, \dots, d\}$来赋一个系数$w_i$，如果特征对最后的影响是正面的，那么就给$w_i$正值，否则给负值。如果我们再规定一个阈值threshold，那么我们的决策方法就可以写为，如果$\sum_{i=1}^{d} w_i x_i > \text{threshold}$，就批准信用卡申请，否则就拒绝。

我们可以进一步地规定输出空间$\mathcal{Y} \in \{-1, +1\}$，其中$y = -1$时表示拒绝，$y = 1$时表示许可。这样做的好处是我们可以直接使用sign函数来求出y的值，具体地说，假设集合$\mathcal{H}$中的每个元素$h \in \mathcal{H}$都有如下形式

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

其中sign函数的定义为

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x > 0 \\ -1 & \text{if } x < 0 \end{cases}$$

即对用户的所有属性做一个加权打分，看它是否超过阈值。如果超过，则批准；否则就拒绝（如果正好等于阈值，这种情况很少发生，甚至可以随机决定y是-1还是1）。

这里我们说$\mathcal{H}$是一个集合的原因是，不同的$\mathbf{w}$和threshold都对应了不同的h，所有这些可能性对应的所有h构成了最后的假设集合$\mathcal{H}$。h这样的函数类型称为**感知机（perceptron）**，其中$\mathbf{w}$称为权重。进一步地，假设我们把$-\text{threshold}$看做是$(-\text{threshold}) \cdot (+1)$，然后把$+1$看作是$x_0$，那么前面的公式形式可以作进一步的简化，即

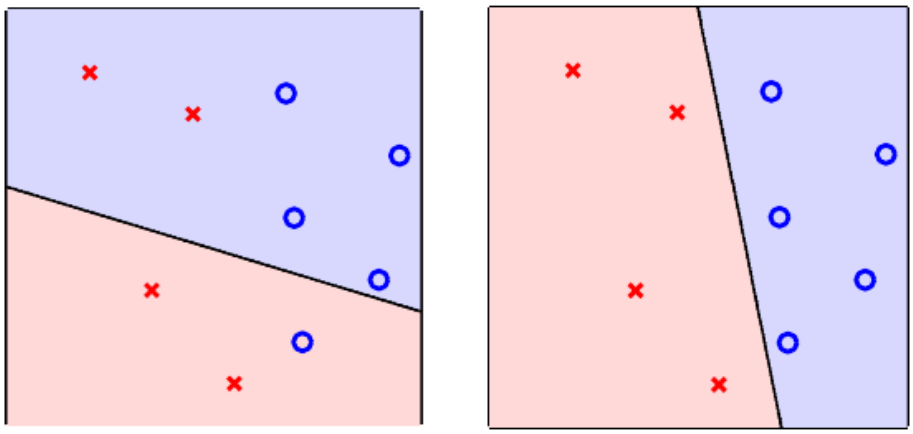
$$\begin{aligned} h(\mathbf{x}) &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \\ &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + \underbrace{(-\text{threshold})}_{w_0} \cdot \underbrace{(+1)}_{x_0}\right) \\ &= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right) \\ &= \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}) \end{aligned}$$

这里$\mathbf{w}$和$\mathbf{x}$都看作是列向量，即维度为$(d+1)1$

我们可以来看一个图例来加强理解。假如我们顾客的特征数（也就是前面说的属性维度）为2，那么我们可以把任意输入$\mathbf{x}$画在一个平面$\mathbb{R}^2$上（类似的，如果特征数为d，那么每个输入$\mathbf{x}$都可以在$\mathbb{R}^d$空间表示，只是会对我们的可视化造成困难），每个输入对应平面上的一个点。这样，$\mathbb{R}^2$上的h都有如下形式：

$$h(\mathbf{x}) = \text{sign}(w_0 + w_1 x_1 + w_2 x_2)$$

可以看出，每个h其实都对应了$\mathbb{R}^2$上的一条直线。感知机规定位于直线某一侧的样本都被判定为正例，另一侧的样本都被判定为负例。不同的权重会产生不同的直线，从而对顾客有不同的分类方式。假设我们用蓝色的圈o表示正例，红色的叉×表示负例，下图给出了两个不同感知机的图



感知机的例子

可以看出右边的感知机在训练集上效果更好，因为它对所有例子做出了正确分类。而左侧的感知机在训练集上表现稍逊（一个正例被误判为负，两个负例被误判为正）

由于感知机都对应于一个超平面，因此它也被称为是**线性分类器**（$\mathbb{R}^2$的超平面是一条直线，$\mathbb{R}^3$的超平面是一个平面，以此类推）。

感知机学习算法

在我们知道了$h \in \mathcal{H}$的形态以后，接下来的问题是设计算法$\mathcal{A}$来选出最优的g来逼近理想的f。尽管我们不知道f具体应该是什么，但是我们知道数据$\mathcal{D}$是由f生成。因此我们有理由相信，好的g满足对所有我们已经收集到的数据，其输出与f的输出尽可能接近，即$g(\mathbf{x}_n) = f(\mathbf{x}_n) = y_n$。因此，我们可以先找一个超平面，至少能够对训练集中的数据正确分类。然而难度在于，$\mathcal{H}$的大小通常都是无限的。

一种解决方案是，我们可以先初始化一个超平面g_0（为了简单起见，将其以其权重$\mathbf{w}_0$代表，称为初始权重）。我们允许这个超平面犯错，但是我们要设计算法，让超平面遇到$\mathcal{D}$中的错分样本以后可以修正自己。通常我们可以将$\mathbf{w}_0$初始化为零向量$\mathbf{0}$。然后，在每一步t，找到一个使$\mathbf{w}_t$错分的样本$(\mathbf{x}_{n(t)}, y_{n(t)})$。即有

$$\text{sign}(\mathbf{w}_t^\mathsf{T}\mathbf{x}_{n(t)}) \neq y_{n(t)}$$

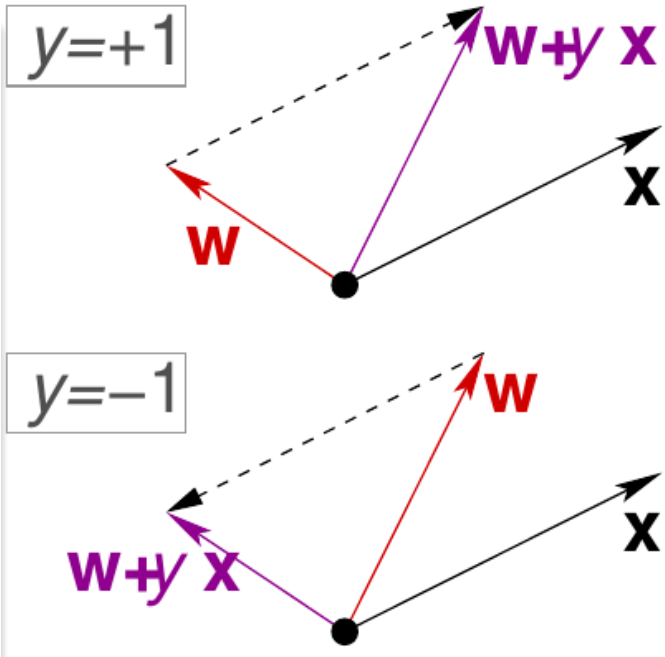
接下来我们试着修正$\mathbf{w}_t$。可以看到错分有两种情况：

- y本来应该是+1，但是模型判断出来是负值。也就是说此时$\mathbf{w}$与$\mathbf{x}$之间的角度太大，因此需要把$\mathbf{w}$往靠近$\mathbf{x}$的方向旋转使它们的角度变小。可以通过让$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$达到这个目的
- y本来应该是-1，但是模型判断出来是正值。也就是说此时$\mathbf{w}$与$\mathbf{x}$之间的角度太小，因此需要把$\mathbf{w}$往远离$\mathbf{x}$的方向旋转使它们的角度变大。考虑到符号，其实也可以通过让$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$达到这个目的

因此，在第$t+1$时刻，我们总可以通过下式来修正$\mathbf{w}_t$，即

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$

下图给出了这两种情况的示例



感知机修正权重的示意图

感知机学习算法（Perceptron Learning Algorithm, PLA）就是重复上面的过程，直到没有错误发生为止。算法将最后得到的权重$\mathbf{w}$（记做$\mathbf{w}_{\text{PLA}}$）返回为g。完整的写法如下

> **感知机学习算法**
>
> 对$t = 0, 1, \dots$
>
> 1). 找到一个使$\mathbf{w}_t$错分的样本$(\mathbf{x}_{n(t)}, y_{n(t)})$。即有
>
> $$\text{sign}(\mathbf{w}_t^\mathsf{T}\mathbf{x}_{n(t)}) \neq y_{n(t)}$$

> 2). 以如下方法修正$\mathbf{w}_t$：
>
> $$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$
>
> 直到遍历了所有样本一遍以后都没有找到错误为止。

注意这里“遍历一遍”不一定非得顺序遍历，只要遍历过样本的一个全排列即可。例如假如样本中有4条数据，可以按照诸如2,3,4,1或者3,2,1,4这样的顺序遍历

讲到这里，我们可能会有以下问题：

- 算法真的会停止吗?
- 能否确定算法返回的$g \approx f$?

我们接下来会讨论这些问题。不过我在这里想首先对Fun time中的题目做一个详细解答。为了方便起见，简记$y = y_{n(t)}, \mathbf{x} = \mathbf{x}_{n(t)}, \mathbf{w} = \mathbf{w}_t$，代入$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$，有

$$\begin{aligned} y\mathbf{w}_{t+1}^{\mathsf{T}}\mathbf{x} &= y(\mathbf{w} + y\mathbf{x})^{\mathsf{T}}\mathbf{x} \\ &= (y\mathbf{w}^{\mathsf{T}} + y(y\mathbf{x})^{\mathsf{T}})\mathbf{x} \\ &= y\mathbf{w}^{\mathsf{T}}\mathbf{x} + y\mathbf{x}^{\mathsf{T}}y^{\mathsf{T}}\mathbf{x} \\ &= y\mathbf{w}^{\mathsf{T}}\mathbf{x} + yy^{\mathsf{T}}\mathbf{x}^{\mathsf{T}}\mathbf{x} \quad (\because y \text{ is a scalar}) \\ &= y\mathbf{w}^{\mathsf{T}}\mathbf{x} + y^2\|\mathbf{x}\|_2^2 > y\mathbf{w}^{\mathsf{T}}\mathbf{x} \end{aligned}$$

感知机的有效性与确定终止性

回顾PLA算法的停止条件，它是在没有找到错误的时候才停止，这要求我们的数据可以用一条线将正例样本和负例样本分割开来（如果不存在这条线，PLA肯定是不可能停止的）。这种条件叫做**线性可分条件**。接下来，我们需要证明：如果数据集的确是线性可分的，感知机是否总能找到一个超平面把数据恰好分开。

假设数据集$\mathcal{D}$线性可分，我们先证明存在一个超平面$\mathbf{w}_f$使得对任意$i \in \{1, \dots, n\}, y_i = \text{sign}(\mathbf{w}_f^{\mathsf{T}}\mathbf{x}_i)$。这意味着对每个$\mathbf{x}_i$，它与超平面都有一定距离，即

$$\min_n y_n \mathbf{w}_f^{\mathsf{T}}\mathbf{x}_n > 0$$

其中$\mathbf{w}_f^{\mathsf{T}}\mathbf{x}_n$是点$\mathbf{x}_n$到$\mathbf{w}_f$的带符号的距离。如果它被放在了相对于超平面的正确一侧，那么这个值与其标签的乘积应该是正数，否则为负数。则在训练过程中遇到的所有错分点$(\mathbf{x}_{n(t)}, y_{n(t)})$（假设在时刻$t$遇到），肯定有

$$y_{n(t)}\mathbf{w}_f^{\mathsf{T}}\mathbf{x}_{n(t)} \geq \min_n y_n \mathbf{w}_f^{\mathsf{T}}\mathbf{x}_n > 0$$

我们可以先证明，$\mathbf{w}_t$被$(\mathbf{x}_{n(t)}, y_{n(t)})$纠正以后更加接近$\mathbf{w}_f$。我们可以通过两个向量的内积来判断它们是否接近：两个向量越接近，内积越大（可以理解为两向量$\mathbf{u}$和$\mathbf{v}$越接近，其夹角θ越小，那么$\cos\theta$越大，所以两者的内积$\mathbf{u} \cdot \mathbf{v} = \|\mathbf{u}\|\|\mathbf{v}\|\cos\theta$越大），则

$$\begin{aligned} \mathbf{w}_f^{\mathsf{T}}\mathbf{w}_{t+1} &= \mathbf{w}_t^{\mathsf{T}}(\mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}) \\ &\geq \mathbf{w}_f^{\mathsf{T}}\mathbf{w}_t + \min_n y_n \mathbf{w}_f^{\mathsf{T}}\mathbf{x}_n \\ &> \mathbf{w}_f^{\mathsf{T}}\mathbf{w}_t + 0 = \mathbf{w}_f^{\mathsf{T}}\mathbf{w}_t \quad \blacksquare \end{aligned}$$

但是这里有一个漏洞，即内积变大不一定说明两个向量接近，因为向量长度变大也会导致内积变大。因此接下来我们要证明，修正$\mathbf{w}_t$以后，新的权重长度不会发生太大变化。这里要用到一个性质，即PLA仅在遇到错误的数据时才更新权重，即如果权重$\mathbf{w}_t$被订正，意味着$\text{sign}(\mathbf{w}_t^{\mathsf{T}}\mathbf{x}_{n(t)}) \neq y_{n(t)}$，也就是$y_{n(t)}\mathbf{w}_t^{\mathsf{T}}\mathbf{x}_{n(t)} \leq 0$。考虑到$y_{n(t)}$是标量，且取值只可能为1或-1（即$y_{n(t)}^2 = 1$），$\mathbf{w}_t^{\mathsf{T}}\mathbf{x}_{n(t)}$也是标量，因此

$$\|\mathbf{w}_{t+1}\|^2 = \|\mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}\|^2$$

简记$y = y_{n(t)}, \mathbf{x} = \mathbf{x}_{n(t)}, \mathbf{w} = \mathbf{w}_t$，则

$$\begin{aligned} \|\mathbf{w}_{t+1}\|^2 &= (\mathbf{w} + y\mathbf{x})^{\mathsf{T}}(\mathbf{w} + y\mathbf{x}) \\ &= \mathbf{w}^{\mathsf{T}}\mathbf{w} + 2y\mathbf{w}^{\mathsf{T}}\mathbf{x} + \mathbf{x}^{\mathsf{T}}\mathbf{x} \\ &\leq \|\mathbf{w}\|^2 + \|\mathbf{x}\|^2 \quad (\because y\mathbf{w}^{\mathsf{T}}\mathbf{x} \leq 0) \\ &\leq \|\mathbf{w}\|^2 + \max_n \|\mathbf{x}_n\|^2 \end{aligned}$$

即权重经过修正以后，其长度最多增加$\max_n \|\mathbf{x}_n\|^2$

经由上面两个部分，假设权重的初始向量为$\mathbf{0}$，我们可以求出经过T步更新最后得到的权重$\mathbf{w}_T$与$\mathbf{w}_f$之间的夹角余弦值的下界。为了求这个值，只需求两个权重归一化以后内积的下界即可，即

$$\inf\left(\frac{\mathbf{w}_f^\mathsf{T}}{\|\mathbf{w}_f\|}\cdot\frac{\mathbf{w}_T}{\|\mathbf{w}_T\|}\right)$$

先看分子。由于初始$\mathbf{w}_0=\mathbf{0}$，因此由之前第一个证明的中间步骤，我们可以写出第一次更新、第二次更新……后分子的下界，即

$$\begin{aligned}
\mathbf{w}_f^\mathsf{T}\mathbf{w}_1&\geq\mathbf{w}_f^\mathsf{T}\cdot\mathbf{0}+\min_n y_n\mathbf{w}_f^\mathsf{T}\mathbf{x}_n\\
\mathbf{w}_f^\mathsf{T}\mathbf{w}_2&\geq\mathbf{w}_f^\mathsf{T}\cdot\mathbf{w}_1+\min_n y_n\mathbf{w}_f^\mathsf{T}\mathbf{x}_n\geq\mathbf{w}_f^\mathsf{T}\cdot\mathbf{0}+2\min_n y_n\mathbf{w}_f^\mathsf{T}\mathbf{x}_n\\
&\vdots\\
\mathbf{w}_f^\mathsf{T}\mathbf{w}_T&\geq T\min_n y_n\mathbf{w}_f^\mathsf{T}\mathbf{x}_n
\end{aligned}$$

类似地，对分母有

$$\|\mathbf{w}_T\|^2\leq T\max_n\|\mathbf{x}_n\|^2$$

因此，

$$\begin{aligned}
\frac{\mathbf{w}_f^\mathsf{T}}{\|\mathbf{w}_f\|}\cdot\frac{\mathbf{w}_T}{\|\mathbf{w}_T\|}&\geq\frac{T\min_n y_n\mathbf{w}_f^\mathsf{T}\mathbf{x}_n}{\|\mathbf{w}_f\|\sqrt{T\max_n\|\mathbf{x}_n\|^2}}\\
&=\sqrt{T}\cdot\frac{\min_n y_n\mathbf{w}_f^\mathsf{T}\mathbf{x}_n}{\|\mathbf{w}_f\|\sqrt{\max_n\|\mathbf{x}_n\|^2}}
\end{aligned}$$

按照Fun Time中的记法，记$R^2=\max_n\|\mathbf{x}_n\|^2,\rho=\min_n y_n\frac{\mathbf{w}_f^\mathsf{T}}{|\mathbf{w}_f|}\mathbf{x}_n$，则

$$\frac{\mathbf{w}_f^\mathsf{T}}{\|\mathbf{w}_f\|}\cdot\frac{\mathbf{w}_T}{\|\mathbf{w}_T\|}\geq\sqrt{T}\cdot\frac{\rho}{R}$$

由于向量除以其长度得到的是单位向量，长度为1，在这种情况下，两者内积越大一定意味着两者的夹角越小，距离越近。但是这里需要注意的是，两者的距离不会无限接近，到$\cos\theta=1$时就会停止。换一个角度看，因为两个单位向量的内积最大值为1，因此从上面的不等式可推出

$$\sqrt{T}\cdot\frac{\rho}{R}\leq1\Rightarrow T\leq\frac{R^2}{\rho^2}$$

即算法至多更新$\frac{R^2}{\rho^2}$步后一定会停止

感知机在线性不可分数据上的应用

由上面的证明，假设数据集是线性可分的，那么PLA算法最后肯定会停止，而且（对训练集）给出正确的分类。该算法非常容易实现，而且结束很快，适用于任意$\mathbb{R}^d$空间。但是这个算法最大的问题是，它要提前假设训练集是训练可分的，而且我们不知道算法什么时候会终止（因为上面给出的上限中用到了$\mathbf{w}_f$，而我们不知道它是多少——甚至不知道是否存在！（在线性不可分的时候该向量不存在））

那么我们来考虑一个最坏的情况，即数据若的确是线性不可分的话，应该如何应对。由于数据产生的过程中可能会混入噪声，这使得原本线性可分的数据也可能因为噪声的存在而不可分。但是，一般情况下，噪声应该是一小部分，即我们可以退而求其次，不去寻找一个完美的超平面，而是去寻找一个犯错误最少的超平面，即

$$\mathbf{w}_g\leftarrow\arg\min_{\mathbf{w}}\sum_{n=1}^{N}[\![y_n\neq\operatorname{sign}(\mathbf{w}^\mathsf{T}\mathbf{x}_n)]\!]$$

然而，求解这个问题被证明是NP难的，只能采用近似算法求解。例如，我们可以保存一个最好的权重，该权重到目前为止错分的数量最少。该算法称为“口袋法”，其完整细节如下

> 设定初始权重$\hat{\mathbf{w}}$
>
> 对时刻$t=0,1,\cdots$

> 1. 随机寻找一个$\mathbf{w}_t$错分的样本$\left(\mathbf{x}_{n(t)}, y_{n(t)}\right)$
> 2. 试图通过如下方法修正$\mathbf{w}_t$
>
> $$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$
>
> 3. 如果$\mathbf{w}_{t+1}$犯的错误比$\hat{\mathbf{w}}$少，那么将$\hat{\mathbf{w}}$替换为$\mathbf{w}_{t+1}$
>
> 直到足够多次迭代完成。我们将$\hat{\mathbf{w}}$（称为$\mathbf{w}_{\text{pocket}}$）返回为$g$

注意在线性可分集合上也可以使用口袋法，算法也可以返回一个无训练误差的解。但是由于每次更新权重以后，都要在所有数据上使用新旧权重各跑一遍，来计算错分数量，因此口袋法的执行时间通常比原始PLA的计算时间长很多

NTUML 3. 机器学习的类型

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/08/05/NTUML-3-Types-of-Learning/

根据输出空间$\mathcal{Y}$分类

二元分类问题

重新回顾一下“是非题”的形式。为了解决这个问题，需要我们提供一批训练数据$\mathcal{D}$，其中我们要指出对哪些用户发放信用卡，哪些不发。像这样答案只有两种可能性（“要”或“不要”）的问题称为**二元分类问题**，其输出空间$\mathcal{Y}$通常用集合$\{-1, +1\}$表示，类似于“判断题”。这种问题类型的例子有很多，包括

- 要不要发信用卡
- 电子邮件是不是垃圾邮件
- 病人有没有生病
- 广告是否会赚钱

等等。

二元分类问题是机器学习中最基本也是最核心的问题，很多理论推导和算法模型设计都是从这一类问题出发。

多元分类问题

二元分类问题很容易进行扩展，即如果答案有多个离散的可能性，那么问题演变为**多元分类问题**。假设目标类别有K种，那么$\mathcal{Y} = \{1, 2, \cdots, K\}$。一个典型的例子是对硬币进行分类，看投入的是1角、5角还是1元。这种问题类似于“选择题”。这种问题类型的例子包括

- 识别手写数字是0到9这十个数字中的哪一种
- 识别图片中的水果是哪一种水果
- 邮件的进一步分类，例如是垃圾邮件、社交网络邮件、重要邮件还是促销活动邮件等等

回归问题

如果将医疗领域中的问题对应到上述问题中，那么这两种问题可以对应如下：

- 二元分类问题：给定病人特征，判断病人是否患病
- 多元分类问题：给定病人特征，判断病人患的是哪种癌症

但是还有一类问题，例如判断病人手术后多少天可以出院。这种问题的输出是整个实数集，或者实数集中的一个连续区间。这种问题通常被称为**回归分析**。此时$\mathcal{Y} \in \mathbb{R}$或$\mathcal{Y} = [\text{lower}, \text{upper}] \subset \mathbb{R}$。这种问题类型的例子包括

- 根据公司的状况，预测其次日股票价格
- 根据大气状况，预测明日气温

回归问题是一种历史悠久的统计问题，也是机器学习领域里非常核心的问题

结构化分析

在自然语言处理（NLP）这个领域里，有一项任务是对于输入句子中的每个词标注其词性（Part of Speech, POS）。例如输入“I love ML”，程序应该可以将“I”标记为代词，“love”标记为动词，“ML”标记为名词。这种任务可以看作是一种多元分类问题，但是如果输入是以句子为单位，由于句子中有结构性，因此输出也是一个结构。这样的问题可以看做是一个巨大的多类别分类问题，各个类别是隐藏的，看不到，而且不同类别之间有联系，使得穷举所有可能性变得不可能。但是我们知道输出存在一定的结构性，并希望程序能够正确给出判定。这种问题称为**结构化分析**，此时$\mathcal{Y}$是一种结构。这种问题类型的例子包括

- 给定蛋白质数据，判断蛋白质的结构
- 给定语言文本，给出语法树

根据数据标签y_n分类

有监督学习

考虑在第一节中的硬币分类问题。我们可以将所有硬币的特征收集起来，设成$\mathbf{x}_n$，同时可以将硬币的面额给出，称为y_n。这两部分可以一起给到机器学习的算法$\mathcal{A}$里，得到g。这种每个特征组$\mathbf{x}_n$都有对应的y_n的学习问题称作**有监督学习**。这里“监督”的意义在于，对每个特征都可以给出对应的标签，是一种“完整”的教学。

无监督学习

如果对所有数据，都没有标签给出，机器也可以通过类似“自学”或者“自己研究”的方式将其归类。这种学习问题称作“无监督学习”。注意这种自动分类（称为聚类）的方法并不一定能得到正确的类数，而且如何判断聚类结果的好坏也是一个难题。这种问题类型的例子包括

- 将文章按照主题归类
- 将用户聚合称为用户群

当然，无监督学习不止聚类这一种方向，还包括了

- 密度估计，即给定$X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$，判断哪里稠密哪里稀疏。例如给定一些带有地点的交通事故资料，判断哪里是事故多发区（有点像回归分析）
- 奇异点检测，即给定$X = \{\mathbf{x}_1, \dots, \mathbf{x}_n\}$，判断哪些是异常点。例如给定网络日志，判断哪些是爬取日志（有点像极端情况下的二元分类）

无监督学习的目标比较分散，难以衡量算法的好坏

半监督学习

介于有监督学习和无监督学习之间，只有少量数据有标签，大部分标签都没有，使用算法判断数据的类别。这类问题的特点是标记数据很贵（标签获取不容易）。

强化学习

类似于训练宠物的方法：当人们训练宠物时，无法直接告诉它给定讯号$\mathbf{x}_n$以后期望的y_n，但是可以在它做了错误的回应以后施加惩罚，通过这种方式告诉它这种做法是错的（当然也可以在它做了的正确的或者你不排斥的回应以后施加奖励）。即此时系统的输入包括了数据X，系统的行为$\tilde{y}$和奖惩函数$\mathrm{goodness}$。这种问题类型的例子包括

- 广告系统：X为用户特征，$\tilde{y}$为给出的广告，$\mathrm{goodness}$是该广告的收入。这样系统可以学到对给定用户应该给出什么广告
- 德州扑克：X为牌型和底池，$\tilde{y}$为叫牌策略，$\mathrm{goodness}$是牌局结束后的收益。这样系统可以学到对给定局面的叫牌策略

强化学习实际上是从一些隐含的数据中学习，这些数据通常是顺序的

根据学习方式$f \Rightarrow (\mathbf{x}_n, y_n)$分类

批处理学习

读进**所有的**数据，训练出来模型g，使用g来处理未知数据。批处理学习是最常见的一种学习方法

在线学习

每看到一个样本就做出预测，然后根据正确的标签对模型做出更新，让模型的效果越来越好。在线学习是一种循序渐进的学习方式。PLA就很容易被改写为在线学习方法，因为它看到一个错分样本就会更新权重。另外，强化学习通常都是在线学习，因为每次我们只能得到部分数据。

主动学习

批处理学习有点像填鸭式教育，在线学习有点像老师课堂讲授，但是这两种方式其实都是机器被动学习。近几年一种新提出的研究方式类似于让机器来主动提问，即对输入$\mathbf{x}_n$来提问对应的y_n。由于机器提问可以有技巧，因此我们希望这种学习可以加速学习过程，同时减少对标签的需求。当标注样本比较昂贵的情况下，主动学习比较有用

根据输入空间$\mathcal{X}$分类

具体特征

这种情况下，输入$\mathcal{X} \subseteq \mathbb{R}^d$中的每个分量（称为**特征**）都有具体且复杂的物理意义。这些特征都包含了人类的智慧，称为“领域知识”，是机器学习能处理的最简单的输入

原始特征

假设要处理的是手写数字识别问题，我们可以对输入做一些分析，提取出具体特征，包括数字是否对称，笔画是否有弯折等等。但是，也可以直接将原始的每个像素的灰度值组合成一个256维向量（假设图片是16x16的）。这个输入比具体特征要抽象一些，求解也更困难，不过这些数据里仍然包含了一些简单的物理意义。原始特征通常需要机器或者人来转换为具体特征，这个转换的过程称为特征工程

抽象特征

以之前提到的预测用户对电影评分的比赛为例，对于这个问题，输入$\mathcal{X} \subseteq \mathbb{N} \times \mathbb{N}$实际上是用户编号和电影编号组成的二元组，而这些编号对机器来讲没有任何物理意义，因此更加需要特征转换

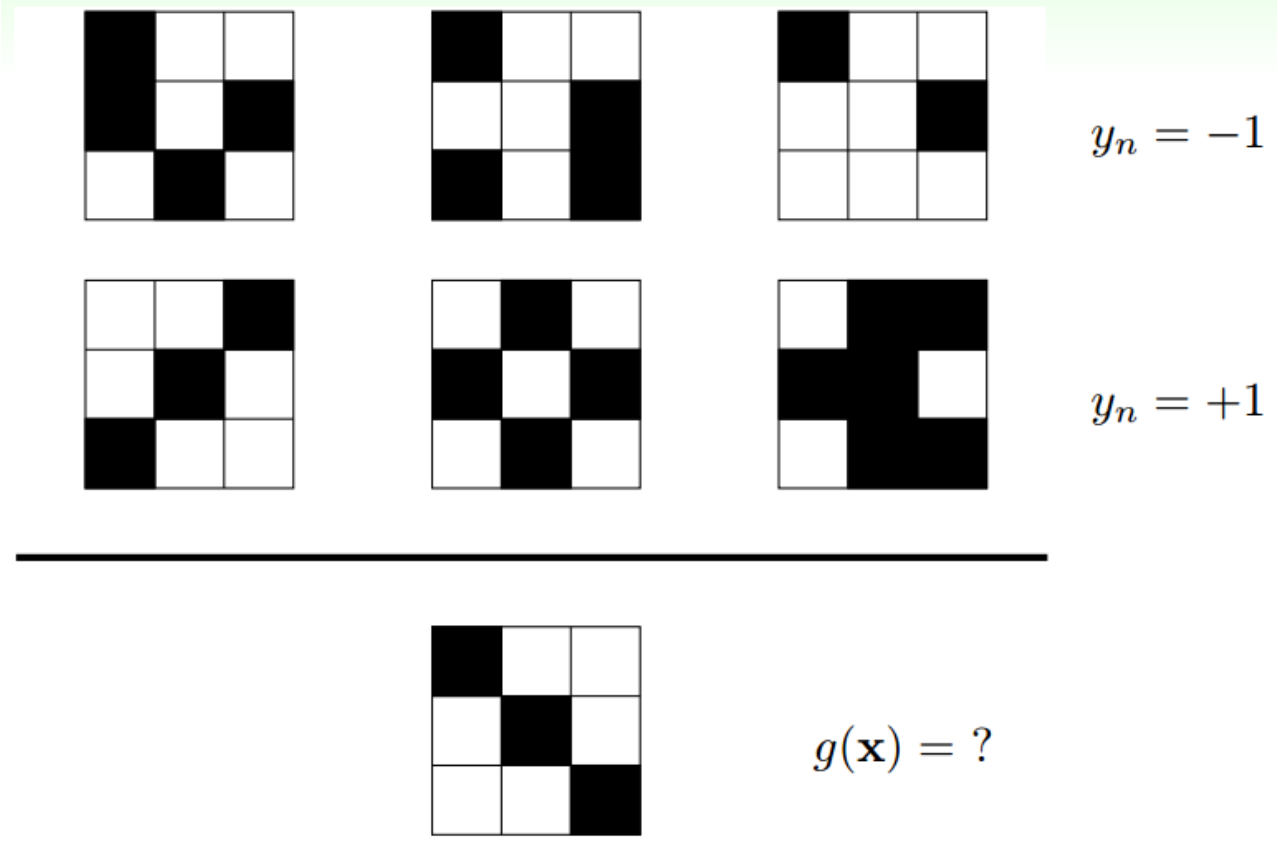
本系列课程主要关注二元分类问题或回归问题，有监督，使用批处理方式处理，数据都有具体特征

NTUML 4. 机器学习的可行性

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/08/06/NTUML-4-Feasibility-of-Learning/

不可能学习的情况

首先来看一个例子：给定下图，你是否能判断最后的图形应该属于哪个类别，是-1还是+1



一个学习谜题

可以说$g(\mathbf{x})=+1$，因为所有已知$y_n=+1$的图形都是对称的。也可以说$g(\mathbf{x})=-1$，因为所有已知$y_n=-1$的图形左上角都是黑的。甚至可以找出更加复杂的规则，让$g(\mathbf{x})$取得+1或-1。可以说，这是一道公说公有理，婆说婆有理的问题，无论怎么样都可以说得到的答案是错误的。即，这个问题不可学习。

接下来看一个更加数学一些的例子：假设输入空间$\mathcal{X}=\{0,1\}^3,\mathcal{Y}=\{\circ,\times\}$，已知的数据集$\mathcal{D}$如下表所示

$\mathbf{x}_n$	$y_n=f(\mathbf{x}_n)$
0 0 0	$\circ$
0 0 1	$\times$
0 1 0	$\times$
0 1 1	$\circ$
1 0 0	$\times$

由于输入情况比较简单，可以列举所有可能的h，也就是$\mathcal{H}$是一个有限集。我们能否学习到一个$g\in\mathcal{H}$，使得$g\approx f$? 答案是，我们只能保证g在$\mathcal{D}$中跟f一样。但是对于那些在$\mathcal{D}$外的数据，类似于上面黑白格的例子，总可以找到一个f使其与g给出的答案不尽相同（甚至完全不同）。但是，机器学习的目的就是要看算法在未知数据上的表现情况。因此可以说，**如果f可以取任何形式**，那么从$\mathcal{D}$中学习模型以对$\mathcal{D}$外的元素做判断，这件事是注定要失败的

概率成为救世主

事已至此，需要开动脑筋，想一想是否有工具可以帮助我们从已知推断未知。这里再看一个例子：假设有一个很大的罐子，里面装了不计其数的橙球和绿球，则是否有可能知道橙球的比例？由于无法一颗一颗地数，因此直接求这个问题很难，但是可以通过随机取样的方法来推断这个比例。例如假设从里面抓了10颗球出来，里面有3颗是橙色的，那么橙球的比例就可以估计为30%。

推而广之，假设罐子中橙球的比例为μ，绿球的比例为$1-\mu$，这里μ是一个未知数。通过某种随机独立程序取样后观测到橙球的比例为ν，绿球的比例为$1-\nu$，这里ν是一个已知数，那么μ和ν有没有联系？这时，无法**确定的**说ν就是μ，因为也许罐子里只有5颗绿球，却有几百颗橙球，而这次随机抽样只拿到了这5颗绿球——这种情况是可能发生的。但是可以退而求其次地说，μ和ν在**大部分情况下非常接近**。准确地说，两者之间的关系可以用Hoeffding不等式来描述。即假设采样大小为N，ϵ为一个比较小的数，那么有

$$P[|\nu-\mu|>\epsilon]\leq 2\exp(-2\epsilon^2N)$$

即“$\nu=\mu$”这个结论“大概差不多是对的”（probably approximately correct, PAC）

Hoeffding不等式对任何N和ϵ都成立，而且不需要知道真实的μ值。抽样样本越多（即N越大），或者界限越松（即ϵ越大），则$\mu\approx\nu$的概率越大

Fun time中，已知$\mu=0.4$，取样数$N=10$，求取样后$\nu\leq 0.1$的概率上界。这时可以把ϵ设为0.3，代入Hoeffding不等式右侧可知上界为$2\exp(-2\times 0.3^2\times 10)\approx 0.3306$。但是如果实际计算可知这个概率应该为$0.6^{10}+10\times 0.6^9\times 0.4\approx 0.04636$。即Hoeffding不等式只能提供一个上界，并不能对概率做出准确预估

概率与机器学习的联系

我们可以把“罐中取球”的问题与机器学习问题建立联系，即

- 未知的橙球比例μ对应于学习问题中未知的目标函数$f(\mathbf{x})$，或者也可以对应于，对给定新样本（测试数据）$\mathbf{x}$，假设函数值$h(\mathbf{x})$是否等于目标函数值$f(\mathbf{x})$
- 罐中的所有球对应于学习问题中的数据点$\mathbf{x}\in\mathcal{X}$
- 取出橙球对应于h判断错误，即$h(\mathbf{x})\neq f(\mathbf{x})$。注意，这里$h$是给定的，对下一条也如此
- 取出绿球对应于h判断正确，即$h(\mathbf{x})=f(\mathbf{x})$
- 从罐中做大小为N的取样对应于学习问题中从数据集$\mathcal{D}=\{(\mathbf{x}_n,\underbrace{y_n}_{f(\mathbf{x}_n)}\}$学习

因此，同样的道理，如果N足够大而且$\mathbf{x}_n$满足独立同分布假设（iid），那么根据已知的错分情况$[\![h(\mathbf{x}_n)\neq y_n]\!]$，就可以推断假设函数$h$在整个数据集上的错分概率$[\![h(\mathbf{x})\neq f(\mathbf{x})]\!]$。如果记$h$对已知数据的错分情况（样本内错误率）为$E_{\text{in}}$，全集上的错分情况（样本外错误率）为$E_{\text{out}}$，就会有

$$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}[\![h(\mathbf{x}_n) \neq y_n]\!]$$
$$E_{\text{out}}(h) = \mathcal{E}_{\mathbf{x}\sim P}[\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

同理套用Hoeffding不等式，在N足够大的情况下，$E_{\text{in}}(h)$和$E_{\text{out}}(h)$之间相差超过ϵ的概率为

$$P[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon] \leq 2\exp(-2\epsilon^2 N)$$

即也可以说h的样本内错误率“大概差不多”相当于其样本外错误率，$E_{\text{in}}(h) \approx E_{\text{out}}(h)$。因此，如果$h$的样本内错误率很小，其样本外错误率也不会很大。在$\mathbf{x}$全都是通过分布$P$产生的前提下，可以说$h \approx f$。那么，如果算法$\mathcal{A}$能够从$\mathcal{H}$中选择到了$E_{\text{in}}$足够小的$h$，其返回为$g$，那么我们能否说$g \approx f$? 如果对给定的$h$，$E_{\text{in}}$足够小，而且$\mathcal{A}$真的把$h$当做$g$返回，那么$g = f$是PAC的。但是如果$\mathcal{A}$被写死了，无论给定什么数据集都要返回$h$作为$g$，那么$E_{\text{in}}(h)$其实往往并不是最小的，此时$g \neq f$反而是PAC的。这意味着真正的学习不是让$\mathcal{A}$返回固定的$h$，而是要让$\mathcal{A}$在解空间$\mathcal{H}$中搜索。不过这提供了一个**验证**的思路，即可以选出一些未在训练过程中使用的历史数据送给某个候选模型h，在这之上观测其表现如何。

概率与真正学习的联系

真正的学习过程中，我们不会只有一个h来验证，而是要在$\mathcal{H} = \{h_1, \ldots, h_M\}$中做出选择。假设找到了一个$h_i$在训练集上表现完全正确，是否应该把它选择为$g$?

先不考虑这个问题。假设有150个人，每个人掷同一枚硬币5次，如果有一个人能连续扔出5个正面，那么这是否意味着这枚硬币不是均匀的？答案是不可能。因为假设硬币均匀，由150人每人掷5次，至少有一个人扔出连续5个正面的概率是$1 - (\frac{31}{32})^{150} > 99$。那么返回刚才的例子，这意味着$h_i$即便在训练集上全对，也不能确定它就比其它的$h'$好（可能大家都是瞎猜）。假设将造成$E_{\text{in}}$和$E_{\text{out}}$相差比较大的取样称为“不好的采样”，可以看到当存在选择的时候，选择会恶化不好的采样（即让你选到错误h的概率变得很大）。

同样的道理，也可以把$E_{\text{in}}(h)$和$E_{\text{out}}(h)$差得特别大的数据称为“不好的数据”。注意Hoeffding不等式只能保证h遇到“不好的数据”的概率不是很大。事实上，每个h_i都会有对应的一些数据$\mathcal{D}'$成为“不好的数据”，例如下表

	$\mathcal{D}_1$	$\mathcal{D}_2$	$\cdots$	$\mathcal{D}_{1126}$	$\cdots$	$\mathcal{D}_{5678}$
h_1	BAD					BAD
h_2		BAD				
h_3	BAD	BAD				BAD
$\cdots$						
h_M	BAD					BAD

从上表可以看到，给出的这四条数据里，只有第1126条是“好的”数据，其余的都会给不同的h带来麻烦。接下来的问题是，如果对某条数据$\mathcal{D}_i$，只要存在$h_i \in \mathcal{H}$使得$\mathcal{D}_i$是h_i“不好的数据”，那就称该数据为“不好的数据”，那么对所有$\mathcal{D}$，如果算法$\mathcal{A}$能够自由选择数据，则算法遇上“不好的数据”的概率有没有上界

答案是有的。使用事件的并的概率的不等式，假设$|\mathcal{H}| = M$，可以做如下推导

$$\begin{aligned}
&P_{\mathcal{D}}[\text{BAD } \mathcal{D}] \\
=&P_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_1 \text{ or BAD } \mathcal{D} \text{ for } h_2 \text{ or } \ldots \text{ or BAD } \mathcal{D} \text{ for } h_M] \\
\leq&P_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_1] + P_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_2] + \ldots + P_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_M] \\
\leq&2\exp(-2\epsilon^2 N) + 2\exp(-2\epsilon^2 N) + \ldots + 2\exp(-2\epsilon^2 N) \\
=&2M\exp(-2\epsilon^2 N)
\end{aligned}$$

即在假设集$\mathcal{H}$为有限集，数据量足够的情况下，每个h都是安全的，即无论$\mathcal{A}$如何选，其返回的g都会有$E_{\text{in}}(g) = E_{\text{out}}(g)$是PAC的。所以，最理性的$\mathcal{A}$会选择$E_{\text{in}}(h_m)$的那个假设函数返回为$g$

但是当M无限大时，如何证明机器学习是可行的？这个问题留在接下来的课程中解决

到此为止，台大机器学习课的第一个部分已全部结束。这一部分解决的是**机器何时能够学习**的问题

NTUML 5. 训练 vs. 测试

- **本文作者：** Tingxun Shi

- **本文链接：** http://txshi-mt.com/2017/08/14/NTUML-5-Training-versus-Testing/
- **版权声明：** 本博客所有文章除特别声明外，均采用 CC BY-NC-SA 3.0 许可协议。转载请注明出处！

再谈E_{in}与E_{out}

上回提到，如果有足够多的根据统计方法生成的数据，而且$\mathcal{H}$是有限集，那么问题是可学习的。即如果假设集$\mathcal{H}$的基数是有限的，记为M，而且有样本量N足够大，那么对任意算法$\mathcal{A}$选出来的g，都会有$E_{\text{out}}(g) \approx E_{\text{in}}(g)$。同样的道理，如果$\mathcal{A}$找到的$g$满足了$E_{\text{in}}(g) \approx 0$，那么有PAC地$E_{\text{out}}(g) \approx 0$，也就是说此时学习是可能的。需要注意的是，要想单纯地通过训练让$E_{\text{in}}(g) \approx 0$是很容易的，只需要把每条数据对应的标签记住就好。但是研究机器学习算法的目的是在于要让模型在**未知的**数据上一样表现良好，因此才需要有测试的过程来验证是否有$E_{\text{in}} \approx E_{\text{out}}$。

这里再回顾一下之前几讲讲过的东西：

- 第一讲主要是强调，我们希望机器学习能够找到一个g使得其接近目标函数f。这个目标可以改写为$E_{\text{out}}(g) \approx 0$。
- 第二讲讲的是，如何寻找一个g，使得$E_{\text{in}}(g) \approx 0$
- 第三讲施加了一个前提条件，即主要讨论批处理、有监督的二元分类问题
- 第四讲则证明在假设集有限、样本量足够的情况下，有$E_{\text{out}}(g) \approx E_{\text{in}}(g)$。这样第二讲做的工作可以联结到第一讲提出的目标

实际上核心是两个问题

- 到底$E_{\text{out}}(g)$和$E_{\text{in}}(g)$会不会接近。如果答案是否定的，那么第一讲和第二讲的内容就无法连接
- 能否让$E_{\text{in}}(g)$尽可能小

那么$|\mathcal{H}| = M$跟这两个问题有什么关系呢？可以分两个情况讨论：

- M比较小的时候，第一个问题能够满足，因为“坏事情”发生的概率（即E_{in}和E_{out}不接近的概率）有上界$2M\exp(-2\epsilon^2 N)$。M小意味着上界小，即E_{in}和E_{out}大部分情况下会接近。但是此时算法的选择有限，很难找到一个让E_{in}接近0的g。
- M比较大的时候，由于有足够的选择，因此可以找到g让E_{in}尽量小，但是“坏事情”发生的概率会大大增加，对两者接近的把握变弱

如果就这么看，那么$M = \infty$是不是一件特别不好的事？感知机会不会并不可用？回顾之前E_{in}和E_{out}不接近这一事件概率的上界，有

$$P[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon] \leq 2M\exp(-2\epsilon^2 N)$$

这里可以先把无穷大的数M替换为一个有穷的数$m_{\mathcal{H}}$看看，即

$$P[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon] \leq^{?} 2m_{\mathcal{H}}\exp(-2\epsilon^2 N)$$

如果换完以后之前的性质仍然成立，那么就可以用有穷的$m_{\mathcal{H}}$来替换无穷的M。这样就可以证明在无限假设集上学习的可行性，并为之后决定如何选择正确的假设集提供支持

有效线性分类器的数量

重新回顾一下对假设h_m所谓”坏事情“这一定义的概念。如果对分类器h_m有$|E_{\text{in}}(h_m) - E_{\text{out}}(h_m)| > \epsilon$，就称为坏事件$\mathcal{B}_m$会发生。由于假设集合比较大，为了让算法$\mathcal{A}$能自由选择，需要计算任一坏事件发生的概率，即求出$P[\mathcal{B}_1 \text{ or } \mathcal{B}_2 \text{ or } \ldots \text{ or } \mathcal{B}_M]$的上界。在最坏的情况下，每个$\mathcal{B}_m$都不互相交叠，所以由事件并的概率上界定理，有

$$P[\mathcal{B}_1 \text{ or } \mathcal{B}_2 \text{ or } \ldots \text{ or } \mathcal{B}_M] \leq P[\mathcal{B}_1] + P[\mathcal{B}_2] + \ldots + P[\mathcal{B}_M]$$

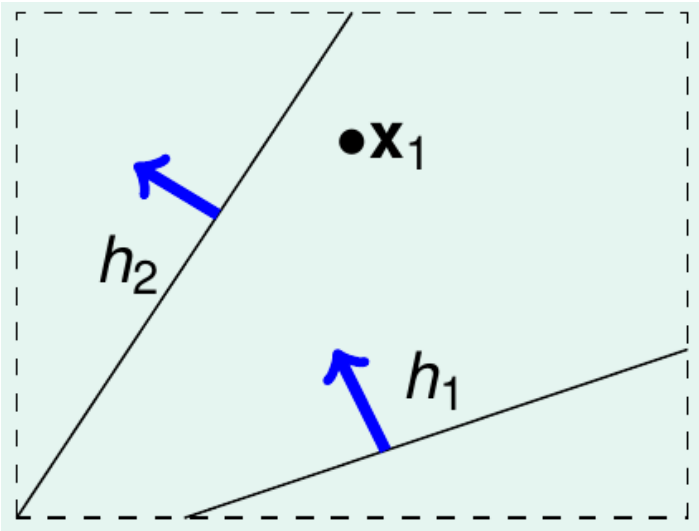
当$M = \infty$时，由于是无限多个概率连加，因此这个上界有可能也是无限大的值，至少会是一个很大的值，会大于1，所以这个上界就没有实际意义了。

问题出在哪儿呢？回顾上界定理中取到上界的条件，是所有子事件互不交叠。但是这个现象在当前讨论的问题中很难出现。相反地，对于相似的假设$h_1 \approx h_2$，其E_{out}是相似的，而对大部分$\mathcal{D}$甚至其E_{in}是相等的。可以通过考虑之前提到的感知机的例子来理解这个结论。对于一个已知的感知机，如果将其稍微旋转或平移一点，就会得到一个新的感知机。这两个感知机只会对那些处于旋转或平移经过的区域中的点给出不同的分类结果，而由于这些区域非常小，$\mathcal{D}$中数据落在这些区域里的概率也会非常小。因此很显然，之前给出的概率上界是被过分估计了的。那么，接下来，就要找出这些”坏事情“重叠的事情

一种自然的想法是，能否把$\mathcal{H}$中所有的h做一个归类，来分析重叠的状况。对$\mathbb{R}^2$上的所有线$\mathcal{H}$，尽管有无限多种可能性，但是可以分类讨论

只有一个数据点

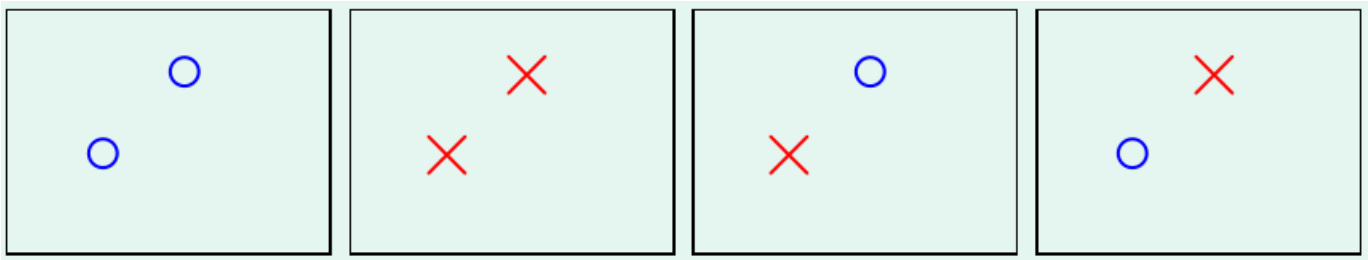
如果数据点只有一个，即输入向量$\mathbf{x}_1$只有一个，那么尽管线有无限多条，但是总体来讲只需要分成两类：h_1类的线把$\mathbf{x}_1$判为正例，h_2类的线把$\mathbf{x}_1$判为负例，如图所示



输入只有一个数据点时，超平面只被分为两类

有两个数据点

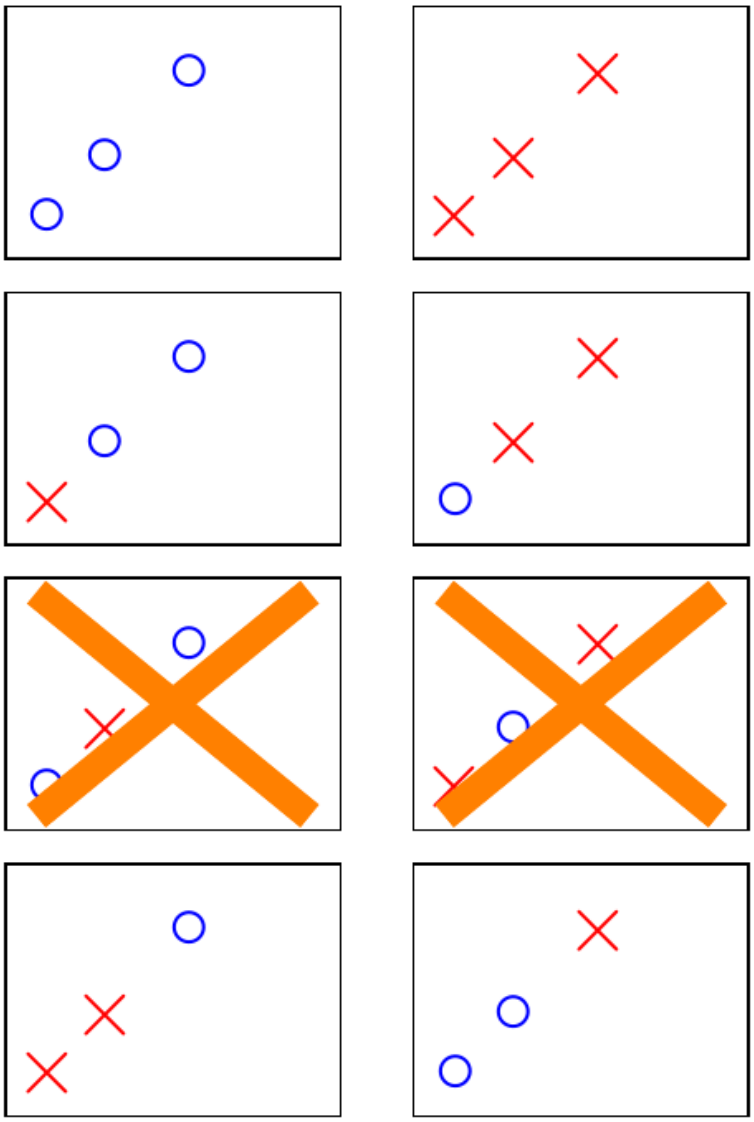
如果输入向量有两个，所有线可以分为四类，这四类线对数据集的分类结果可以如下图所示



输入如果有两个数据点，分类器一共有四种

有三个数据点

如果输入向量有三个，大概率情况下，所有线可以分为八类。但是需要注意的是，**这种情况并不总是成立的！**如果输入的向量三点共线，那么所有分类器只能分成六类（其中有两类情况是线性不可分的，如下图所示）



有三个输入向量时的最坏情况。打叉的两种情况线性不可分

有四个数据点

如果输入向量有四个，所有线最多可以分为14种（当然如果共线或者正负例交叠，分类器的种类就会更少）。

这里就不再放图占用篇幅了，不过可以参考所谓的”异或分类问题“

总结

之前所枚举的例子，实际上是指出了对N个不同的输入$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$，将其分开的线性分类器最多可以分为几类。这称之为**有效线性分类器的数量**（effective number of lines），可以简记为$\mathrm{effective}(N)$。当$N \in \{1, 2, 3, 4\}$时，$\mathrm{effective(N)}$对应为2、4、8、14。可以看出一个很重要的结论：对线性分类器，有

$$\mathrm{effective}(N) \leq 2^N$$

这样，可以做如下猜想：尽管$\mathbb{R}^2$上的直线数量M是无限的，但是对有限的输入N，其有效线性分类器的数量$\mathrm{effective}(N)$是有限的，那么在之前Hoeffding不等式中就可以用$\mathrm{effective}(N)$来取代M，这样可以得到一个有限的上界。特别的，如果还能满足$\mathrm{effective}(N) \ll 2^N$，那么Hoeffding不等式的上界可以再次减小，而且有效线性分类器的数量也会变得很小。这说明即便假设集是无限集，学习也是可能的

假设集的有效大小与成长函数

考虑更加宽泛的情况，即分类器不是一个线性分类器时的情况。如果分类器h在数据集$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$上的结果只可能是$\times$或$\circ$，即

$$h(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = (h(\mathbf{x}_1), h(\mathbf{x}_2), \ldots, h(\mathbf{x}_N)) \in \{\times, \circ\}^N$$

那么称h是一个二分法（dichotomy）分类器。我们要看的是一个假设集$\mathcal{H}$可以做出多少个不同的二分法分类器，可以将$\mathcal{H}$用作$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$上得到的所有不同的二分法分类器组成的集合记做$\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$。例如，如果$\mathcal{H}$是$\mathbb{R}^2$上所有的直线，那么当

$N=4$时，$\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N)$是类似$\{\circ\circ\circ\circ,\circ\circ\circ\times,\circ\circ\times\times,\dots\}$这样的集合。接下来要做的是，看能否用$|\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N)|$取代之前Hoeffding不等式里的$M$

然而这里有一点不妥，就是$|\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N)|$与输入的形式关系紧密。例如，还是回到前一节中所举的例子，当$N=3$，$\mathcal{H}$为$\mathbb{R}^2$上的所有直线时，如果输入三点共线，则$|\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N)|=6$。如果两两共线，三足鼎立，则$|\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N)|=8$。为了摆脱其对输入的依赖，可以对所有不同输入进行考虑，选出最大的$|\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N)|$值，记其为$m_{\mathcal{H}}(N)$，就有

$$m_{\mathcal{H}}(N)=\max_{\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N\in\mathcal{X}}|\mathcal{H}(\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N)|$$

实际上$m_{\mathcal{H}}(N)$是一个关于N的函数，其中N是输入的点的个数。如果回到前一讲的例子，可以看出来在这种情况下$m_{\mathcal{H}}(N)$实际上就是前一讲中的$\mathrm{effective}(N)$。我们把这个函数起一个更通用的名字，叫做**成长函数**（grow function）。成长函数的上限确定存在，其上界为2^N（N个样本每个有两种取值）。

对不同的假设集，确定其成长函数具体值的难易程度有不同。如果是之前感知机的问题，那么确定成长函数的具体值是很难的。但是接下来可以看一些相对简单的例子：

正射线（Positive Rays）的成长函数

如果输入就是一维实数，即$\mathcal{X}=\mathbb{R}$，假设集集合$\mathcal{H}$中的每个h都会定义一个阈值a，其分类方法为

$$h(x)=\mathrm{sign}(x-a)$$

所以每个不一样的阈值a就会定义一个不一样的假设函数h

如果给定了N个数据点，那么对该$\mathcal{H}$，有$m_{\mathcal{H}}(N)=N+1$。因为N个点可以把一条直线切成$N+1$个不同的区间，任何落在同一个区间内的阈值a其决定的h效果都一样。注意当N很大时，会有$(N+1)\ll 2^N$

正区间（Positive Interval）的成长函数

接下来，输入不变，对$\mathcal{H}$的定义稍作变化：不再让大于某个阈值ℓ的所有x都被分为+1，而是施加一个上限r，将所有$x\in[\ell,r)$分为+1，即此时对$h\in\mathcal{H}$有

$$h(x)=+1\text{ iff }x\in[\ell,r),-1\text{ otherwise}$$

此时，N个点还是可以把一条直线切成$N+1$个不同的区间。如果ℓ和r都落在不同的区间，那么一部分x会被标为+1，另一部分被标为-1。这样的ℓ和r的选法共有$\binom{N+1}{2}$种。还有一种情况，就是ℓ和r都落在同一个区间，此时所有x都会被标为-1。因此对于这样的$\mathcal{H}$，其成长函数为

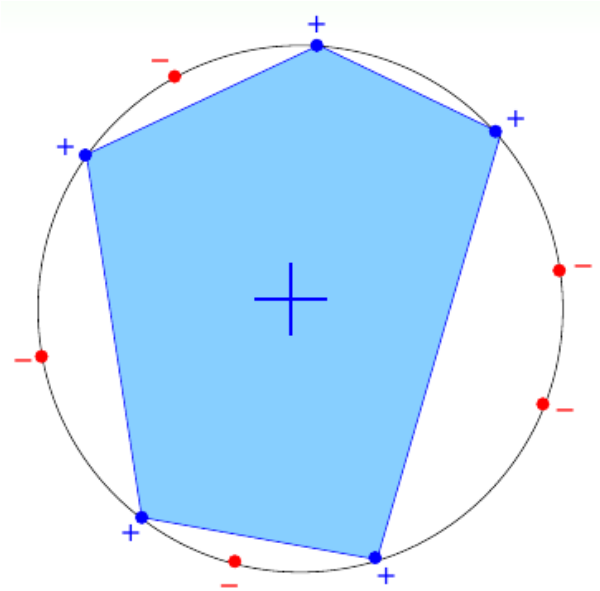
$$\begin{aligned}m_{\mathcal{H}}(N)&=\binom{N+1}{2}+1\\&=\frac{1}{2}N^2+\frac{1}{2}N+1\end{aligned}$$

注意此时，仍然有N很大时，$m_{\mathcal{H}}(N)\ll 2^N$

凸集合的成长函数

接下来，考虑一个稍微复杂点的情况，即输入空间$\mathcal{X}\in\mathbb{R}^2$，对$h\in\mathcal{H}$，如果$\mathbf{x}$在一个凸集合里（可以看做是二元平面上的凸多边形），则$h(\mathbf{x})=+1$，否则$h(\mathbf{x})=-1$，计算$\mathcal{H}$的成长函数

可以先看最极端的情况，即所有$\mathbf{x}_1,\mathbf{x}_2,\dots,\mathbf{x}_N$都放在一个大的圆上。此时，任何可能的二分法分类器都可以通过如下方法得到：找出所有标记为+1的点，所有相邻的点（跳过它们中间的负例点）以线相连，这样会得到一个凸多边形，凸多边形上（及其内部）的点都标记为+1，其外的点都标记为-1。下图给出了一个例子



凸集的成长函数

因此，无论输入的N有多大，总可以做出来$\mathcal{H}$上的**全部**二分法分类器，即此时$m_{\mathcal{H}}(N)=2^N$。此时称这N个点可以被假设集合$\mathcal{H}$**打散**（shatter）

停止点（break point）

回到最开始提出$m_{\mathcal{H}}(N)$的目的，是为了取代Hoeffding不等式上界中的M值。从前面的讨论可以发现，成长函数有两种复杂度：一种是以正射线为代表的多项式复杂度，一种是以凸集合为代表的指数复杂度。由于原有的不等式上界里有一个指数为负的指数项，因此如果成长函数是多项式的，那么随着N的增大，指数项衰减的速度远快于成长函数增长的速度，可以保证N很大的时候$E_{\text{in}}(g)$与$E_{\text{out}}(g)$之间相差很大的概率趋近0。但如果成长函数是指数的，就没有这样的性质

那么感知机的成长函数是“好的”（多项式复杂度）还是“不好的”（指数复杂度）呢？正规证明在下讲给出，不过这里给出一个辅助概念。由前面的分析，可以得知，只有在对某个规模的输入，不能做出全部二分法分类器，使得$m_{\mathcal{H}}(N)$不能等于2^N时，才有可能避免指数复杂度的成长函数。例如，对线性分类器，当输入的数据大小为4时，不可能找出$2^4=16$种不同的线性分类器来完全区分所有情况（只能找到14种）。称这样的点叫做**停止点**，即

> 如果**任意**k个输入都不能被$\mathcal{H}$打散，就称k为$\mathcal{H}$的**停止点**。即若k是$\mathcal{H}$的停止点，就有
>
> $$m_{\mathcal{H}}(k-1)=2^{k-1},\ m_{\mathcal{H}}(k)<2^k$$

这里需要注意停止点的任意性。还是以线性分类器为例，尽管之前例子证明在$N=3$时，如果三点共线，也无法找到$2^3=8$种线性分类器，但是仍然存在3个输入被$\mathcal{H}$打散的情况（三点不共线），因此还是有$m_{\mathcal{H}}(3)=8$，即3不是$\mathcal{H}$的停止点

停止点有一个很好的性质：如果k是$\mathcal{H}$的停止点，那么$k+1,k+2,\ldots$都是停止点。所以我们尤其感兴趣$\mathcal{H}$的第一个停止点（在不引起混淆的情况下，可以直接简记为“$\mathcal{H}$的停止点”）。根据之前的介绍有

- 正射线的停止点为2
- 正区间的停止点为3
- 二维平面感知机的停止点为4
- 凸集合没有停止点

从上面的停止点值可知，显然当$\mathcal{H}$没有停止点时，$m_{\mathcal{H}}(N)=2^N$。但是似乎还有一个性质，就是如果$\mathcal{H}$的停止点为k，那么是不是有$m_{\mathcal{H}}(N)=O(N^{k-1})$？这个问题留待接下来的课程中证明。

NTUML 6. 一般化理论

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/08/16/NTUML-6-Theory-of-Generalization/

停止点对成长函数值的限制

如果对于某个$\mathcal{H}$，其停止点$k=2$，这透露了什么信息？这意味着当$N=1$时，可以产生两种二分法分类器，$m_{\mathcal{H}}=2$；但是当$N=2$时，由于达到了停止点，由定义必然有$m_{\mathcal{H}}(N)<2^2=4$，因此最多产生三种分类器。

那么当$N=3$呢？我们把分类器集合先置为空集，然后一点点加入可能的分类器来看

1. 第一个分类器，将$\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$分别标记为$\circ,\circ,\circ$，这个分类器是合法的。因为任意取两个$\mathbf{x}_i$只有一种组合，没有被$\mathcal{H}$打散，保持了停止点的性质
2. 第二个分类器标记为$\circ,\circ,\times$，也合法
3. 第三个分类器标记为$\circ,\times,\circ$，仍然合法
4. 第四个分类器，此时要小心。如果标记为$\circ,\times,\times$，那么结合前面已经有的分类器，如果把$\mathbf{x}_1$去掉，看$\mathbf{x}_2$和$\mathbf{x}_3$组合的情况，就会有$\{\circ,\circ\}$、$\{\circ,\times\}$、$\{\times,\circ\}$、$\{\times,\times\}$四种情况，即这两个输入被$\mathcal{H}$打散。因此这样的分类器打破了停止点的性质，是非法的。经过分析可知第四个分类器只能标记为$\times,\circ,\circ$
5. 第五个分类器，此时只有三个选择（因为$\circ,\times,\times$已经在上一步判定为不可行），可以逐个考察
   - 标记为$\times,\circ,\times$是不行的，因为$\mathcal{H}$打散了$\mathbf{x}_1$和$\mathbf{x}_3$
   - 标记为$\times,\times,\circ$是不行的，因为$\mathcal{H}$打散了$\mathbf{x}_1$和$\mathbf{x}_2$
   - 标记为$\times,\times,\times$是不行的，因为$\mathcal{H}$还是打散了$\mathbf{x}_1$和$\mathbf{x}_2$

即当$N=3$时，最多产生四种分类器。因此我们可以初步得到一个推论，即**只要出现了停止点k，它就会对所有$N>k$时最大可能的成长函数$m_{\mathcal{H}}(N)$施加了一些限制。**这样就可以得出一个想法

$$\begin{aligned}&m_{\mathcal{H}}(N)\\\leq&\text{ maximum possible }m_{\mathcal{H}}(N)\text{ given }k\\\leq&poly(N)\end{aligned}$$

如果上述猜测能够被证明，而且$m_{\mathcal{H}}$可以取代Hoeffding不等式上界中的M，那么感知机算法就是可行的

上限函数：基本情况

由前面的讨论，在成长函数的基础上，可以定义一个**上限函数**（bounding function）$B(N,k)$，表示当停止点为k时最大可能的$m_{\mathcal{H}}(N)$值。可以通过一个更抽象的排列组合问题来研究上限函数的值：假设输入是一个长度为N的，由$\circ$和$\times$组成的向量，如果限定任意长度为k的子向量都不能被打散（即对任意长度为k的子向量，都不会出现$\circ$和$\times$的所有2^k中组合），则这个向量最多有多少种可能?

对这个问题进行抽象的好处是，抽象以后不再需要$\mathcal{H}$的具体形态。例如，无论$\mathcal{H}$是正区间还是1维感知机，它们的停止点都是3，因此上限函数都是$B(N,3)$

在有了上限函数的定义以后，我们想要证明

$$B(N,k)\leq poly(N)$$

首先，根据前一节的讨论，有

> $$B(2,2)=3,B(3,2)=4$$

然后，根据前一节最后Fun time的小题，有

> $$B(N,1)=1$$

再次，如果$N<k$，那么此时不受停止点约束，可以随意让$\mathcal{H}$打散，因此

> $$B(N,k)=2^N\text{ for }N<k$$

最后，如果$N=k$，只需要从所有2^k种分类器中删掉一个就能满足停止点约束，因此

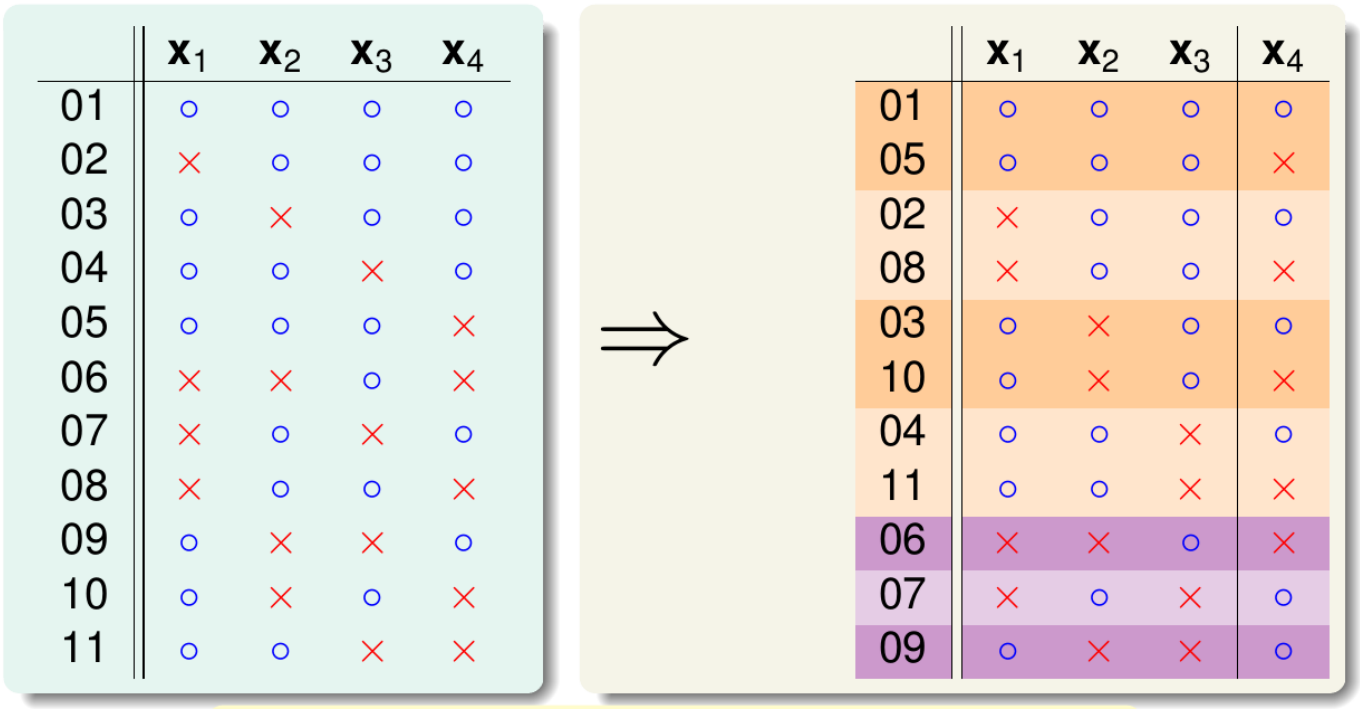
> $$B(N,k)=2^N-1\text{ for }N=k$$

注意，已经证明有二维空间中的感知机的停止点$k=4$，$N=4$时的成长函数值$m_{\mathcal{H}}(4)$为14，而$B(4,4)=15$，因此上限函数实际给出的是一个宽松的上界

上限函数：其它情况的推导

$N>k$且$k\neq 1$的情况比较复杂。不过经过观察，似乎$B(4,3)$与$B(3,?)$有联系：把已有的4个点去掉一个，或者向原有的3个点增加一个，问题就会互相转化。

暴力搜索以后，发现$B(4,3)=11$。但是这个值怎么与前一行的值联系起来呢？可以将$B(4,3)$的解重新整理，归为两类，如下图所示



	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
01	○	○	○	○
02	×	○	○	○
03	○	×	○	○
04	○	○	×	○
05	○	○	○	×
06	×	×	○	×
07	×	○	×	○
08	×	○	○	×
09	○	×	×	○
10	○	×	○	×
11	○	○	×	×

⇒

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
01	○	○	○	○
05	○	○	○	×
02	×	○	○	○
08	×	○	○	×
03	○	×	○	○
10	○	×	○	×
04	○	○	×	○
11	○	○	×	×
06	×	×	○	×
07	×	○	×	○
09	○	×	×	○

B(4,3)合法分类器的全部形式与归类

其中橙色部分的解都是“成双成对”的，即(1, 5)，(2, 8)，(3, 10)，(4, 11)这些分类器只是对$\mathbf{x}_4$的分法不同。而紫色部分的解相对“形单影只”，没有其它组合可以对应。如果记橙色部分的数量是2α，紫色部分的数量是β，那么首先可以立即得到$B(4,3)=11=2\alpha+\beta$

进一步地，从输入集中去掉$\mathbf{x}_4$以后，可以得到$\alpha+\beta$个二分法分类器。由上限函数中参数值可知，目前的假设集合$\mathcal{H}$停止点为3，意味着不存在能打散三个输入的假设函数，所以$\alpha+\beta$也不应该打散这三个输入。因此可以得到第一个推导

$$\alpha+\beta\leq B(3,3)$$

接下来，对橙色部分的分类器去掉$\mathbf{x}_4$，得到去重以后在$\mathbf{x}_1,\mathbf{x}_2,\mathbf{x}_3$上分类器的集合。如果这个集合里的某个分类器子集能够将任意两个$\mathbf{x}_i,\mathbf{x}_j$组成的输入打散，则由于其处于橙色部分，对$\mathbf{x}_4$可以产生完全的$2^1=2$个不同的分类结果，因此可以把$\mathbf{x}_i,\mathbf{x}_j,\mathbf{x}_4$打散，与$\mathcal{H}$停止点$k$为3这一假设相悖。因此这$\alpha$个分类器不能打散任意两个输入，即

$$\alpha\leq B(3,2)$$

将上面的推导组合并推广，可以得到如下结论

$$\begin{aligned}B(N,k)&=2\alpha+\beta\\ \alpha+\beta&\leq B(N-1,k)\\ \alpha&\leq B(N-1,k-1)\\ \Rightarrow B(N,k)&\leq B(N-1,k)+B(N-1,k-1)\end{aligned}$$

即得到了上限函数的上限。最后可以填出如下的表格

$B(N,k)$		k				
	1	2	3	4	5	6
	1	1	2	2	2	2
	2	1	3	4	4	4
	3	1	4	7	8	8
N	4	1	≤ 5	11	15	16
	5	1	≤ 6	≤ 16	≤ 26	31
	6	1	≤ 7	≤ 22	≤ 42	≤ 57

上限函数的上限表

由上表的边界情况和递推式，有

$$B(N,k) \leq \sum_{i=0}^{k-1} \binom{N}{i}$$

而不等式的右边最高项为N^{k-1}。因此，对假设集合$\mathcal{H}$，如果其存在停止点且值为k，则其成长函数$m_{\mathcal{H}}(N)$的上限由上限函数$B(N,k)$决定，上限函数的上限由一个多项式函数$O(N^{k-1})$决定，综上所述，如果停止点存在，则$m_{\mathcal{H}}(N)$是多项式的。实际上，还可以证明$B(N,k) \geq \sum_{i=0}^{k-1} \binom{N}{i}$，即两者之间是相等的

$\mathcal{H}$中“坏情况”发生概率上界的证明

在证明了成长函数是多项式复杂度以后，一个自然的想法是把成长函数值放进Hoeffding不等式右侧，替换掉M。然而事情实际上并不如此遂意，直接替换的结果是不对的。尽管如此，还是可以得到一个形式差不多的上界，即**在N足够大的情况下**，有

$$P[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon] \leq 2 \cdot 2m_{\mathcal{H}}(2N) \cdot \exp\left(-2 \cdot \frac{1}{16}\epsilon^2 N\right)$$

接下来，给出一个概要的证明，主要是针对一些多出来的常数是如何发生的：

首先，在不等号左侧，有一个棘手的E_{out}需要解决。因为训练集的大小总是有限的，因此$E_{\text{in}}(h)$也是有限的。但是E_{out}却是无限的。所以关键问题是怎么把这个无限值转变成有限值。如何用有限多个点知道E_{out}呢？核心思想是借鉴之前训练-测试的方法，使用验证集来进行估计。可以取样出N个点构成验证集$\mathcal{D}'$，估计出E'_{in}。由于E_{in}、E'_{in}的分布应该都类似于期望为E_{out}的正态分布，因此如果某个假设函数h有其E_{in}跟E_{out}差得很远，那么有很大的概率（大于1/2）其E_{in}离E'_{in}也很远，所以可以用E'_{in}换掉E_{out}。换完以后，有（更具体的数学证明不再列出）

$$\frac{1}{2}P[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon] \leq P[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E'_{\text{in}}(h)| > \frac{\epsilon}{2}]$$

接下来，由于E_{in}有$m_{\mathcal{H}}(N)$种，E'_{in}也有$m_{\mathcal{H}}(N)$种，因此综合起来，一共有$m_{\mathcal{H}}(2N)$种假设函数可以使用。使用之前事件并的概率的上界，有

$$\text{BAD} \leq 2m_{\mathcal{H}}(2N)P[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E'_{\text{in}}(h)| > \frac{\epsilon}{2}]$$

最后看上述不等式中P的上界。还是使用之前类似的罐子举例子，只不过此时罐子中有$2N$个球，要比较的是挑出的N个球观测到的概率和罐里N个球反映的概率之间差别，由于全体观测反映的概率可以用取出球观测到的概率和罐内球反映的概率取均值得到，因此通过简单的计算，两者有以下关系

$$|E_{\text{in}} - E'_{\text{in}}| > \frac{\epsilon}{2} \Leftrightarrow \left|E_{\text{in}} - \frac{E_{\text{in}} + E'_{\text{in}}}{2}\right| > \frac{\epsilon}{4}$$

其中可以看做是一个总体的概率，因此概率的上界可以用Hoeffding不等式求出（不过注意这里其实是无放回的Hoeffding不等式），即

$$P[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E'_{\text{in}}(h)| > \frac{\epsilon}{2}] \leq 2\exp\left(-2\left(\frac{\epsilon}{4}\right)^2 N\right)$$

综上所述，证明了VC（Vapnik-Chervonenkis）上界，即

$$P[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon] \leq 4m_{\mathcal{H}}(2N) \cdot \exp\left(-\frac{1}{8}\epsilon^2 N\right)$$

最后回到感知机算法。由于二维空间感知机的停止点为4，因此$m_{\mathcal{H}}(N)$是$O(N^3)$的，多项式时间复杂度。当N增大时，$m_{\mathcal{H}}(N)$的增长速度小于指数函数的衰减速度，整体上“坏事情”发生的概率大幅度减小。所以如果感知机算法$\mathcal{A}$选出了E_{in}最小的g，这个g也会有最小的E_{out}，达到机器学习的效果

NTUML 7. VC维

VC维的定义

第六讲证明，如果样本数N足够大且假设集合$\mathcal{H}$的成长函数$m_{\mathcal{H}}(N)$存在突破点，那么就有$E_{\text{out}} \approx E_{\text{in}}$。其中更具体地，如果成长函数存在突破点$k$，那么当$N > k$时，成长函数的值由上限函数$B(N,k)$所限制，该值等于$\sum_{i=0}^{k-1} \binom{N}{i}$，其最高项为$N^{k-1}$是$N$的多项式函数。而观察$B(N,k)$的值表和$N^{k-1}$的值表，可以发现当$N \geq 2, k \geq 3$时，$B(N,k)$就已经很小于$N^{k-1}$了。因此可以得到成长函数一个宽松的上界，即

$$m_{\mathcal{H}}(N) \leq B(N,k) = \sum_{i=0}^{k-1} \binom{N}{i} \leq N^{k-1}$$

由于机器学习算法$\mathcal{A}$最后只会选择出一个模型g，因此这个模型遭遇“坏事情”（即$|E_{\text{in}}(g) - E_{\text{out}}(g)|$比较大）的概率肯定小于$\mathcal{H}$中任意一个模型$h$遭遇“坏事情”的概率。再代入前面推出的VC上界和成长函数的上界，对统计上足够大的数据集$\mathcal{D}$和算法$g = \mathcal{A}(\mathcal{D}) \in \mathcal{H}$，如果$k \geq 3$有（因为已经说了$\mathcal{D}$足够大，因此肯定有$N$足够大，即$N \geq 2$）

$$\begin{aligned}
&P_{\mathcal{D}}\left[|E_{\text{in}}(g) - E_{\text{out}}(g)| > \epsilon\right] \\
\leq &P_{\mathcal{D}}[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon] \\
\leq &4 m_{\mathcal{H}}(2N) \exp\left(-\frac{1}{8}\epsilon^2 N\right) \\
\leq &4(2N)^{k-1} \exp\left(-\frac{1}{8}\epsilon^2 N\right) \quad (\text{if } k \text{ exists})
\end{aligned}$$

即在以下三种条件同时满足的情况下，机器可能会学习。包括

- 成长函数的停止点为k（$\mathcal{H}$足够好）
- N足够大，可以一般化结论$E_{\text{out}} \approx E_{\text{in}}$（$\mathcal{D}$足够好）
- 算法$\mathcal{A}$选出的模型g能有足够小的E_{in}（$\mathcal{A}$足够好）

（以及一点好的运气）

那么这些推导与VC维有什么关系？简单说，**VC维是“最大非停止点”的更正式的说法**，是假设函数集$\mathcal{H}$的一种性质。在N小于等于$\mathcal{H}$的VC维（记为$d_{\text{VC}}(\mathcal{H})$）时，存在某种形式的$N$个输入可以被$\mathcal{H}$打散；当$N > d_{\text{VC}}(\mathcal{H})$时，任意$N$个输入都不能被$\mathcal{H}$打散，即达到了停止点。这个定义可以写作

> $\mathcal{H}$的VC维，记做$d_{\text{VC}}(\mathcal{H})$，指的是满足$m_{\mathcal{H}}(N) = 2^N$的最大的$N$。从数值上，其等于最小的停止点$k$减去1

如果$\mathcal{H}$没有停止点，则$d_{\text{VC}}(\mathcal{H}) = \infty$

化用前面的结论，有当$N \geq 2, d_{\text{VC}} \geq 2$时，$m_{\mathcal{H}}(N) \leq N^{d_{\text{VC}}}$

那么第六讲中分析的一些$\mathcal{H}$的VC维如下所示：

- 正射线的VC维是1
- 正区间的VC维是2
- 凸集的VC维是∞
- 二维空间中感知机的VC维是3

即“好的假设集”总是有有限的VC维，这意味着其会将$E_{\text{out}}(g) \approx E_{\text{in}}(g)$一般化，而且与算法$\mathcal{A}$的形式、数据的分布$P$和目标函数$f$的形式都无关

（注意由本节的fun time，即便对**某**N个输入不能被$\mathcal{H}$打散，也不能确定$d_{\text{VC}}(\mathcal{H})$与$N$的关系。如果要有$d_{\text{VC}}(\mathcal{H}) < N$，要求的是**任意**$N$个输入都不能被$\mathcal{H}$打散）

感知机的VC维

对于二维的感知机，一方面，如果数据集$\mathcal{D}$线性可分，意味着感知机PLA算法可以收敛，即当迭代了足够的步数T以后，有$E_{\text{in}}(g) = 0$。另一方面，如果数据来自于某个概率分布，且标签由某个确定的目标函数所产生（即$\mathbf{x}_n \sim P, y_n = f(\mathbf{x}_n)$，那么由于二维感知机的VC维是3，因此“坏事情”发生的概率很小，N足够大时，有$E_{\text{out}}(g) \approx E_{\text{in}}(g)$。两者综合起来，有$E_{\text{out}}(g) \approx 0$，即二维感知机算法是能达到学习效果的

同样的推论能否应用在$\mathbf{x}$有多于两个特征的情况？回顾之前的分析，一维感知机（正/负射线）的VC维是2，二维感知机的VC维是3，那么是否有d维感知机的VC维是$d+1$？接下来试着证明之（本节简记“d维空间感知机的VC维”为d_{VC}）：

首先证明$d_{\text{VC}} \geq d+1$。若要证明该命题，只需构造$d+1$个输入，使得该输入可以被感知机假设集$\mathcal{H}$打散即可。我们做如下构造：这$d+1$条数据中，第一条是d维空间的零点，第二条到第d+1条中第i $(i = \{1, 2, \ldots, d\})$个分量为1其它分量为0。将$x_0 = 1$补入矩

阵，有

$$\mathrm{X}=\begin{bmatrix}-\mathbf{x}_1^{\mathsf{T}}-\\-\mathbf{x}_2^{\mathsf{T}}-\\-\mathbf{x}_3^{\mathsf{T}}-\\\vdots\\-\mathbf{x}_{d+1}^{\mathsf{T}}-\end{bmatrix}=\begin{bmatrix}1&0&0&\cdots&0\\1&1&0&\cdots&0\\1&0&1&\cdots&0\\\vdots&\vdots&&\ddots&\vdots\\1&0&0&\cdots&1\end{bmatrix}$$

注意X是可逆的，即其逆矩阵存在且唯一。而“打散”意味着，对于任意一种$y_1,y_2,\cdots,y_{d+1}$的排列组合$\mathbf{y}$，总有$\mathrm{sign}(\mathrm{X}\mathbf{w})=\mathbf{y}$。而由于若$\mathrm{X}\mathbf{w}=\mathbf{y}$，前面的式子也成立。又因为$\mathrm{X}$可逆，因此只需要使权重$\mathbf{w}$为$\mathrm{X}^{-1}\mathbf{y}$，就可以满足对任意的$\mathbf{y}$有$\mathrm{sign}(\mathrm{X}\mathbf{w})=\mathbf{y}$，因此这个$\mathrm{X}$可以被$\mathcal{H}$打散，$d_{\mathrm{VC}}\geq d+1$

接下来证明$d_{\mathrm{VC}}\leq d+1$，即任何$d+2$个输入都不能被$\mathcal{H}$打散。要证明这个任意性比较困难，但是可以先回到二维空间再仔细研究一番。

之前提到，如果输入三个点位于直角坐标系中$(0,0),(0,1),(1,0)$这三个点，那么输入可以被$\mathcal{H}$打散。如果加入第四个点$(1,1)$会怎么样？首先可以把这四个点写出来放进矩阵中，即

$$\mathrm{X}=\begin{bmatrix}-\mathbf{x}_1^{\mathsf{T}}-\\-\mathbf{x}_2^{\mathsf{T}}-\\-\mathbf{x}_3^{\mathsf{T}}-\\-\mathbf{x}_4^{\mathsf{T}}-\end{bmatrix}=\begin{bmatrix}1&0&0\\1&1&0\\1&0&1\\1&1&1\end{bmatrix}$$

进一步，假设$y_2=y_3=\circ(+1),y_1=\times(-1)$，由前面的分析可知$y_4$一定不能为$\times$，否则整个数据集线性不可分。接下来解释为什么会出现这样的情况（或者说，来看如果数据集线性可分那为什么y_4一定为$\circ$）

根据矩阵中的关系，可以得到$\mathbf{x}_4=\mathbf{x}_2+\mathbf{x}_3-\mathbf{x}_1$。由于数据集线性可分，因此存在一个最优的权重$\mathbf{w}$。将这个权重左乘到等式中，有$\mathbf{w}^{\mathsf{T}}\mathbf{x}_4=\mathbf{w}^{\mathsf{T}}\mathbf{x}_2+\mathbf{w}^{\mathsf{T}}\mathbf{x}_3-\mathbf{w}^{\mathsf{T}}\mathbf{x}_1$。又$y_2=y_3=+1,y_1=-1$，由sign函数的性质可知必然有$\mathbf{w}^{\mathsf{T}}\mathbf{x}_2>0,\mathbf{w}^{\mathsf{T}}\mathbf{x}_3>0,\mathbf{w}^{\mathsf{T}}\mathbf{x}_1<0$，但是由于$\mathbf{w}^{\mathsf{T}}\mathbf{x}_1$这一项前面是符，因此将这些负号系代入，可知$\mathbf{w}^{\mathsf{T}}\mathbf{x}_4$必然大于0，对应为$y_4$必然是正的。造成这种现象的根源是$\mathbf{x}_4$是其它三项的线性组合，数据集线性相关。因此可以得到结论：**输入数据的线性相关性限制了二分法分类器的种类数目**

将上面的推导过程推广到d维空间。假设$\mathbf{x}$有d个维度，补入x_0以后有$d+1$个维度，现在数据集里有$d+2$条数据。由线性代数的知识可知，这$d+2$个变量肯定线性相关，因此存在不全为0的a_i使得$\mathbf{x}_{d+2}=a_1\mathbf{x}_1+a_2\mathbf{x}_2+\ldots+a_{d+1}\mathbf{x}_{d+1}$。如果$\mathbf{w}$与各个$\mathbf{x_i}$的内积得到的符号恰好是$a_i$的符号，那么$\mathbf{w}^{\mathsf{T}}\mathbf{x}_{d+2}$的符号一定是正的，$y_{d+2}$不可能属于负类。因此任何$\mathrm{X}$不能被打散

（这里有一种更直接而且更严谨的说法：由于线性相关性的存在，$\mathbf{x}_{d+2}$的符号完全剩余$d+1$个$\mathbf{x}$决定。如果最优的$\mathbf{w}$也存在，那么$\mathbf{w}^{\mathsf{T}}\mathbf{x}_{d+2}$的符号一定是确定的，如果$y_{d+2}$的符号和这个确定的符号不一致，那么就会线性不可分，也就是说X不能被打散）

因此，$d_{\mathrm{VC}}\geq d+1,d_{\mathrm{VC}}\leq d+1\Rightarrow d_{\mathrm{VC}}=d+1$

■

VC维的物理意义

在前面的证明中可以发现一个有趣的现象：d维感知机的VC维是$d+1$，同时其权重又有$d+1$个分量，两者之间是否存在关系？可以认为假设参数$\mathbf{w}=（w_0,w_1,\ldots,w_d)$定义了**自由度**的概念，而VC维的物理意义可以理解为假设集$\mathcal{H}$在做二元分类问题时自由度的的大小。自由度越大，VC维越大，分类器能力越强。

例如，正射线的VC维是1，其自由变量正好也有一个，是a（大于a的都标为正）。正区间的VC维是2，其自由变量正好也有两个，是ℓ和r（落在区间$[\ell,r)$的都标为正）。因此，*大致上*可以理解为，VC维就是模型可调节的自由参数数量（不过这个说法并不总是对的）。

既然VC维代表了$\mathcal{H}$的自由度，回顾第五讲，当时有一个大M小M的权衡问题：小M可以保证E_{in}和E_{out}接近，但是选择太少，可能找不到使E_{in}很小的g；而大M可以找到使E_{in}很小的g，却不能保证“坏事情”发生的概率很小。由于前面对VC上界的推导证明当$\mathcal{H}$无限时M可以被$2(2N)^{d_{\mathrm{VC}}}$所代替，因此可以类似地说，VC维小可以保证E_{in}和E_{out}接近，但是模型的能力太弱，可能找不到使E_{in}很小的g；而VC维大意味着模型能力足够强，可以找到使E_{in}很小的g，却不能保证“坏事情”发生的概率很小。这说明，**选择合适的VC维（也就是选择合适的假设集合$\mathcal{H}$），是非常重要的！**

对VC维的更深入解释

要想更深入地解释VC维，需要对原有VC上界的形式做一个改写。记原有的VC上界为δ，即

$$\delta = 4(2N)^{d_{\mathrm{VC}}}\exp\left(-\frac{1}{8}\epsilon^2 N\right)$$

从另一个角度看，该上界是“坏事情”发生概率的上界，同时也可以据此求出是“好事情”（即$|E_{\mathrm{in}}(\mathrm{g}) - \mathrm{E}_{\mathrm{out}}(\mathrm{g})| \leq \epsilon$）发生的概率的下界。有

$$P[|E_{\mathrm{in}}(\mathrm{g}) - \mathrm{E}_{\mathrm{out}}(\mathrm{g})| \leq \epsilon] \geq 1 - \delta$$

对刚才δ的表达式做简单代数变换，可以得到ϵ的表达式

$$\epsilon = \sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{\mathrm{VC}}}}{\delta}\right)}$$

称$|E_{\mathrm{in}}(\mathrm{g}) - \mathrm{E}_{\mathrm{out}}(\mathrm{g})|$为一般化误差，可知

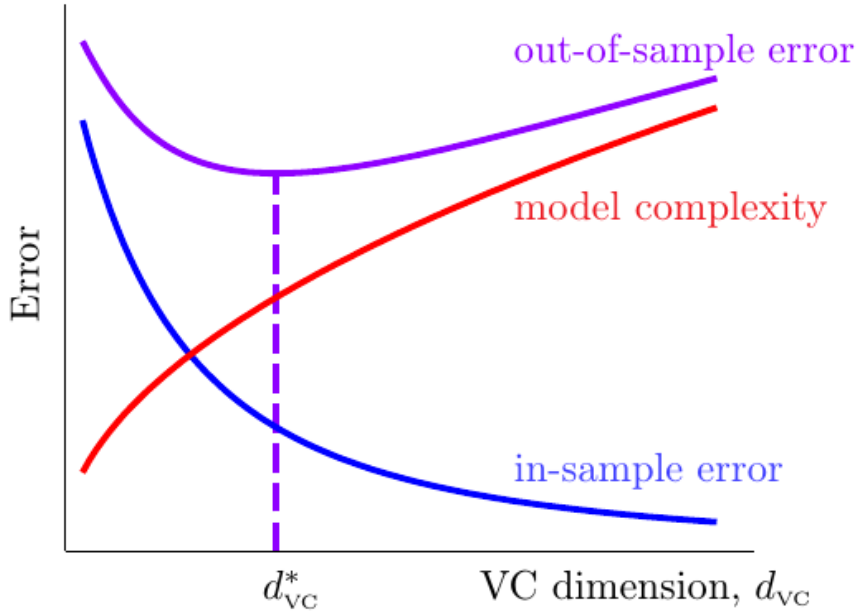
$$|E_{\mathrm{in}}(\mathrm{g}) - \mathrm{E}_{\mathrm{out}}(\mathrm{g})| \leq \sqrt{\frac{8}{\mathrm{N}}\ln\left(\frac{4(2\mathrm{N})^{\mathrm{d_{VC}}}}{\delta}\right)}$$

因此可以得到E_{out}类似“置信区间”的范围，为

$$E_{\mathrm{in}}(g) - \sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{\mathrm{VC}}}}{\delta}\right)} \leq E_{\mathrm{out}}(g) \leq E_{\mathrm{in}} + \sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{\mathrm{VC}}}}{\delta}\right)}$$

一般不关心E_{out}的下界，但是会关心其上界。其中根号里面的项称为**模型复杂度**，可以看作是关于样本大小、假设集合和VC上界的函数$\Omega(N, \mathcal{H}, \delta)$，而$\sqrt{\Omega(N, \mathcal{H}, \delta)}$称作*模型复杂度提升带来的代价*。

可以把VC维和模型在样本内外的误差之间的关系作图表示



VC维与样本内外误差之间的关系

从图中可知，随着d_{VC}变大，模型可以打散的输入越多，因此E_{in}会变小。但是这样会使得模型复杂度提高。当d_{VC}维不那么大的时候，E_{in}下降的速度超过模型复杂度提升的速度，E_{out}会下降。但是超过某个临界点d^*_{VC}以后，模型复杂度提升的速度更快，E_{out}经过最小值以后会上升。这意味着最好的VC维d^*_{VC}通常在中间取得，并不是越大越好。考虑到VC维和模型复杂度的关系，这说明$\mathcal{H}$**并非总是越强大越好！**

VC维的另外一个作用就是可以决定样本复杂度，即做多大的采样能够将“坏事件“（训练集内外误差差值大于ϵ）发生的概率控制在δ以内。例如，如果$\epsilon = 0.1, \delta = 0.1, d_{\mathrm{VC}} = 3$，那么理论上需要采集29300条数据才能满足要求，此时样本数$N \approx 10000 d_{\mathrm{VC}}$。不过实践中，通常$N \approx 10 d_{\mathrm{VC}}$就够用了。这意味着，VC上界是一个非常宽松的上界，这是因为

- 使用了Hoeffding不等式，这个上界估计独立于产生数据的概率分布，也独立于目标函数的形式
- 使用了成长函数$m_{\mathcal{H}}(N)$而没有使用假设函数个数$|\mathcal{H}(\mathbf{x}_1, \ldots \mathbf{x}_N)|$，因此独立于具体的数据（成长函数也是高估）
- 使用了多项式上限$N^{d_{\mathrm{VC}}}$来代替成长函数$m_{\mathcal{H}}(N)$，因此独立于任何假设函数集合$\mathcal{H}$，只用考虑VC维这一个参数

- 使用了事件并的发生概率的上界，即将算法$\mathcal{A}$选出遇到”坏事件“假设函数的概率提高了很多

即计算VC上界时采用了很多宽松的估计。不过在理论上，这个上界想改进，让它更紧一些，也比较难。因此之后设计算法时，更多要注意的是VC上界背后透露的哲学思想，以此来作指导（例如不能让$\mathcal{H}$太过强大等等）。

NTUML 8. 噪声和错误

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/08/21/NTUML-8-Noise-and-Error/

噪声和目标分布

在感知机模型的口袋算法中曾经简单介绍过一点关于噪声的概念。需要注意的是，噪声可能出现在各种地方，例如

- 噪声出现在标签y中，例如好的用户被错误地标记为坏的用户
- 噪声出现在标签y中的另一种形式是相同的顾客被打上了不同的标签
- 噪声还可能出现在数据$\mathbf{x}$中，例如我们收集到的顾客信息不准确

那么在有噪声的情况下，VC上界是否还有意义？要解答这个问题，需要理解VC上界的核心是什么。回顾之前讲解VC维时，都是拿小球举的例子。这里，每个小球都可以对应为学习时看到的$\mathbf{x}$，每个球被选中的概率遵循分布$P(\mathbf{x})$。小球颜色的确定方法为，如果对这条数据，假设函数h的值和目标函数f的值不一样，就将小球漆为橙色。那么有了噪声怎么办？噪声的出现使得对确定的$\mathbf{x}$其颜色可能会变化，不再由f所决定。即小球是否为橙色（对应于$y \neq h(\mathbf{x})$）现在由概率分布$y \sim P(y|\mathbf{x})$决定，不再是一个恒定的值（课程里的说法是小球的颜色会一会儿一变）。然而即便如此，仍然可以估算出罐中（同一时刻）橙色球的比例，只需要在拿出球的一刹那将球的颜色记录下来。这意味着，无论小球的颜色是否会变，橙色球的比例都可以通过抽样的方法来确定。因此，只要$\mathbf{x} \sim^{\text{i.i.d}} P(\mathbf{x}), y \sim^{\text{i.i.d}} P(y|\mathbf{x})$，即$(,y)P(,y)$

由于现在y不再由f决定而是由$P(y|\mathbf{x})$决定，因此类似于f“目标函数”的叫法，这个$P(y|\mathbf{x})$被称为“目标分布”，即其说明了对每个$\mathbf{x}$最理想的预测是什么，同时还说明了噪声产生的概率。例如，假设对某个$\mathbf{x}$有$P(\circ|\mathbf{x}) = 0.7, P(\times|\mathbf{x}) = 0.3$，则其对应的标签应该是$\circ$，同时有30%的情况下会带来噪声。在这样的定义下，目标函数的概念也可以用目标分布来表示，即

$$P(y|\mathbf{x}) = \begin{cases} 1 & \text{if } y = f(\mathbf{x}) \\ 0 & \text{elsewhere} \end{cases}$$

目标函数的概念被扩展了以后，机器学习的目标也就发生了变化，其是通过常见的输入（分布$P(\mathbf{x})$告知我们哪些点是重要的）来预测最理想的标签值（由分布$P(y|\mathbf{x})$决定）

错误的衡量

在用机器学习算法解决问题的过程中，最后一步是要看选择的假设g是否能满足$g \approx f$。为了判断这一点，之前使用的方法是看E_{out}，其中

$$E_{\text{out}}(g) = \mathcal{E}_{\mathbf{x} \sim P}[\![g(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

更一般地说，需要给g一个衡量的方法。之前使用的这种度量方法满足了三个特点

- 使用的是样本外数据来衡量。E_{out}反映的是对所有未知$\mathbf{x}$所犯错误的平均值
- 可以逐点操作。尽管反映的是所有错误的平均值，但是对一条数据也可以度量其错误情况。不需要取样一批数据以后才能衡量
- 适用于分类问题

这里的错误度量方法度量的是分类错误，通常也叫作0/1误差

记E_{out}中被求期望的函数为$\text{err}(g(\mathbf{x}), f(\mathbf{x}))$，这里这个err可以对任意给定的数据点求出其错误值，因此也被称为每个点上衡量错误的方式。这样，E_{in}和E_{out}就可以表示为

$$E_{\text{in}}(g) = \frac{1}{N}\sum_{n=1}^{N} \text{err}(g(\mathbf{x}_n), f(\mathbf{x}_n))$$
$$E_{\text{out}}(g) = \mathcal{E}_{\mathbf{x} \sim P} \text{err}(g(\mathbf{x}), f(\mathbf{x}))$$

在有误差存在的情况下，f会被之前讨论过的y和P所取代

本次课程主要会使用这种逐点衡量错误的方式，因为这种方式比较简单。将$g(\mathbf{x})$记为$\tilde{y}$，将$f(\mathbf{x})$记为y，那么错误衡量方法一共可以分为两种：

- 0-1误差。求出的$\tilde{y}$只有两种可能：与y一样，或者不一样。用数学方法表示为$\text{err}(\tilde{y}, y) = [\![\tilde{y} \neq y]\!]$。这种误差方法通常用在分类问题中
- 平方误差。用数学方法表示为$\text{err}(\tilde{y}, y) = (\tilde{y} - y)^2$，看的是$\tilde{y}$与$y$有多接近

错误衡量函数（也称为误差函数）和目标分布一起决定了最理想的目标函数的形式。例如，假设现在有$P(y = 1|\mathbf{x}) = 0.2, P(y = 2|\mathbf{x}) = 0.7, P(y = 3|\mathbf{x}) = 0.1$ 如果使用0-1误差作为误差函数，则误差值与$\tilde{y}$之间的关系如下

$$\text{err} = \begin{cases} 0.8 & \text{if } \tilde{y} = 1 \\ 0.3 & \text{if } \tilde{y} = 2 \\ 0.9 & \text{if } \tilde{y} = 3 \\ 1.0 & \text{if } \tilde{y} = 1.9 \end{cases}$$

这里$\tilde{y} = 2$是最优的预测，同时$\tilde{y} = 1.9$的效果非常不好。但是如果使用平方误差作为误差函数，结果就大有不同

$$\text{err} = \begin{cases} 1.1 & \text{if } \tilde{y} = 1 \\ 0.3 & \text{if } \tilde{y} = 2 \\ 1.5 & \text{if } \tilde{y} = 3 \\ 0.29 & \text{if } \tilde{y} = 1.9 \end{cases}$$

这时之前效果很差的预测值$\tilde{y} = 1.9$反而变成了最好的预测值！事实上，可以证明，如果误差函数是0-1误差，则最好的预测值应该为$f(\mathbf{x}) = \arg\max_{y \in \mathcal{Y}} P(y|\mathbf{x})$；如果误差函数是平方误差，则最好的预测值应该为$f(\mathbf{x}) = \sum_{y \in \mathcal{Y}} y \cdot P(y|\mathbf{x})$

因此，错误衡量函数扮演了一个非常重要的角色。一方面，它指导机器学习算法$\mathcal{A}$选出一个最好的g，另一方面，它评估了所得到的模型与目标函数之间的相似性$g \approx f$

算法误差的衡量

这些错误的衡量从哪里来？想象要做一个指纹辨识的应用，如果是你本人的指纹，输出+1，否则输出-1。可以看出来这里实际上有两种类型的错误：假阳性（应该输出-1的样本输出了+1）和假阴性（应该输出+1的样本输出了-1）。0-1误差对这两种错误的惩罚程度是相同的，但是在各个情况下这样做都是最合理的方式么？考虑以下两种类型问题：

- 超市的指纹识别系统，如果指纹识别出来是一个熟客，就给予一定折扣。这种情况下，如果用户身份没有被识别因此没有折扣（假阴性），顾客会非常生气，可能会造成客户流失；而如果用户的身份被错误地认定为是熟客（假阳性），超市顶多会少赚点钱，同时由于顾客留下了指纹，万一真的需要追查也容易。因此在这种情况下，假阴性造成的后果更加严重，应该多惩罚一些，例如权重放大到10倍
- 国家安全部门的指纹识别系统，如果指纹识别出来是一个员工，就放行。这种情况下，如果发生假阴性，员工多刷几次指纹无所谓；但是如果假阳性，放进了非员工，就会造成情报泄露，非常严重。这种情况下，假阳性的惩罚权重应该增加更多，例如1000倍

由此可知，不同的任务使用的误差衡量函数不尽相同，即便两者都属于分类问题也如此。因此设计算法$\mathcal{A}$时最好就要把误差衡量函数包括进去。但是很多时候这种时候无法做到，比如难以将权重数值化等等。通常有两种替代方式，来找到一个算法中可用的误差函数$\hat{\text{err}}$

- 让误差函数“看上去更加合理”，就是使用0/1误差函数或者平方误差（更具体地讲，前者最小化了“翻转误差”，后者最小化了“高斯误差”）
- 让算法$\mathcal{A}$更容易优化，例如使用存在闭合形式解的目标函数或凸目标函数

加权的分类问题

对分类问题，由于假阴性和假阳性带来的后果不同，因此通常会有一个矩阵，这个矩阵通常被称作代价矩阵（也可以被称为误差矩阵、损失矩阵等等）。例如，国安部门问题的代价矩阵如下所示

		$h(\mathbf{x})$	
		+1	-1
y	+1	0	1
	-1	1000	0

国安部门问题的代价矩阵

在这种情况下，E_{out}和E_{in}的形式也有所变化，为

$$E_{\text{out}}(h) = \mathcal{E}_{(\mathbf{x},y)\sim P}\left\{\begin{array}{ll} 1 & \text{if } y=+1 \\ 1000 & \text{if } y=-1 \end{array}\right\} \cdot [\![y \neq h(\mathbf{x})]\!]$$

$$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}\left\{\begin{array}{ll} 1 & \text{if } y=+1 \\ 1000 & \text{if } y=-1 \end{array}\right\} \cdot [\![y \neq h(\mathbf{x}_n)]\!]$$

这种问题称作**加权的分类问题**，因为对不同的$(\mathbf{x}, y)$权重不同

这种问题如何求解？首先VC理论仍然适用，所以可以肯定问题是可以求解的。接下来的问题就是怎么让E_{in}^{w}越小越好（这里上角标w表示是加权问题）。目前讲过的两种模型中，如果数据线性可分，那么PLA肯定可以收敛而且不会受权重改变的影响；如果数据线性不可分，就要使用口袋算法，不过要做一些修改，即当$\mathbf{w}_{t+1}$所达到的$E_{\text{in}}^{\mathbf{w}}$更小时，再用它替换$\hat{\mathbf{w}}$

但是这样的修改是否合理？换句话说，修改后的算法能否达到使$E_{\text{in}}^{\mathbf{w}}$最小这个目标？由于负样本被错分的代价是原来是1000倍，那么可以构造一个新的数据集，其中原来的数据集中每个标签为-1的样本都被复制了1000份放在新的数据集中。称原来的数据集叫原始数据集，被复制了1000倍负样本的数据集为等价数据集，这使得对任何模型h，如果它对某个点$(\mathbf{x}_i, -1)$在原始数据集上犯了错，它在等价数据集上会犯1000次同样的错误。这样，h在等价数据集上的0/1错误率$E_{\text{in}}^{0/1}$实际上等于其在原始数据集上的加权错误率$E_{\text{in}}^{\mathbf{w}}$。由于口袋算法在等价数据集上可用，因此其在原始数据集上也可用

当然，真正使用算法时，将负样本数据复制一千次可能是费力的且无必要的。可以采用“虚拟复制”的方法来修改得到一种加权口袋算法，主要修改两点：

- 增加负样本被采样的频率，使其1000倍于正样本被采样的频率
- 当$\mathbf{w}_{t+1}$所达到的$E_{\text{in}}^{\mathbf{w}}$更小时，再用它替换$\hat{\mathbf{w}}$

本节的fun time给出了一个非常实际问题的场景。对于某些分类问题，其类别可能会非常不平衡，例如题目中给出的比率是负例只占了全体样本的0.001%。在这种情况下，即便是把负例的权重提高到1000，一个“常数分类器”（即对所有样本都无脑设置成+1）的分类错误率仍然能被控制在1%。这意味着，对于这种问题，应该精巧地设计罕见类样本的权重，避免这种“常数分类器”的出现

NTUML 9. 线性回归

线性回归问题

回顾之前举的银行信用卡问题，如果要关注的问题不再是“给不给发卡”而是“发多大的额度”，那么这就不再是一个二元分类问题，而输出空间已经是实数集（这里实际上是实数集的一个子集）了，因此是一个回归问题。

回归问题的假设函数应该是什么样的？假设已经提取好了d个特征$x_1, x_2, \ldots, x_d$，可以像感知机问题那样再加入一个新的常数项x_0组成输入向量$\mathbf{x}$。一个直观的做法是为每个特征赋予一个权重w_i，这样这些特征的加权均值就应该接近于期望得到的y，即

$$y \approx \sum_{i=0}^{d} w_i x_i$$

这其实就是线性回归的假设函数了，即$h(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x}$。可以看到这个假设函数跟感知机的有点像，但是不需要取符号，因为最后要得到的结果就应该是个连续值。

要洞悉线性回归的思想，可以从最简单的情况入手。假设输入只有一个维度，平面$\mathbb{R}^2$上的直角坐标系横轴为输入x，纵轴为输出y，那么训练数据就是落在坐标系上的若干个点，而对任意$x \in \mathbb{R}$，假设函数求出来的对应的值$h(x) \in \mathbb{R}$应该连成一条直线。期望是对于训练集里任意的点x_i，假设函数值$h(x_i)$应该尽量接近于这个点已知的值y_i。那么如何衡量$h(x_i)$和y_i的接近程度？一种方法就是计算这两个值之间差的绝对值$|h(x_i) - y(i)|$，称作**残差**。线性回归的目的就是找到合适的超平面，使得残差尽量小。类似的道理推广到n维空间$\mathbb{R}^n$也是一样的

既然问题求解的目的已经确定，那么对应的误差衡量函数就很明朗。通常情况下，回归问题所使用的误差函数叫做平方误差函数，即

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

如前面讨论的那样，误差通常会被分为样本内误差和样本外误差。样本内误差被记为

$$E_{\text{in}}(h) = \frac{1}{N}\sum_{n=1}^{N}(h(\mathbf{x}_n) - y_n)^2$$

由于有$h(\mathbf{x}_n) = \mathbf{w}^\mathsf{T}\mathbf{x}_n$，因此一个$h$通常会对应于一个$\mathbf{w}$，$E_{\text{in}}$也可以表示为一个关于$\mathbf{w}$的函数。类似地，样本外误差也可以表示为一个关于$\mathbf{w}$的函数

$$E_{\text{out}}(\mathbf{w}) = \mathcal{E}_{(\mathbf{x},y)\sim P}(\mathbf{w}^\mathsf{T}\mathbf{x} - y)^2$$

根据之前的课程，接下来要着重解决的问题就是怎么使E_{in}尽可能小

线性回归算法

首先可以试着把上一节得到的E_{in}用矩阵形式表示，以使得其形式变得更加简洁。

$$\begin{aligned}
E_{\text{in}}(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^\mathsf{T}\mathbf{x}_n - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n^\mathsf{T}\mathbf{w} - y_n)^2 \\
&= \frac{1}{N}\left\| \begin{bmatrix} \mathbf{x}_1^\mathsf{T}\mathbf{w} - y_1 \\ \mathbf{x}_2^\mathsf{T}\mathbf{w} - y_2 \\ \cdots \\ \mathbf{x}_N^\mathsf{T}\mathbf{w} - y_N \end{bmatrix} \right\|^2 \\
&= \frac{1}{N}\left\| \begin{bmatrix} -- \mathbf{x}_1^\mathsf{T} -- \\ -- \mathbf{x}_2^\mathsf{T} -- \\ \cdots \\ -- \mathbf{x}_N^\mathsf{T} -- \end{bmatrix} \mathbf{w} - \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix} \right\|^2 \\
&= \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2
\end{aligned}$$

最后得到的各矩阵/向量维度为：

- X：$N \times (d+1)$
- $\mathbf{w}$：$(d+1)1$
- $\mathbf{y}$：$N \times 1$

由于E_{in}中只有$\mathbf{w}$是未知数，而经过证明可以发现，$E_{\text{in}}(\mathbf{w})$是连续且可微的凸函数。凸函数的一个重要性之就是，其在极值点的梯度为0，而梯度的定义是把函数在每个方向上做偏微分，因此在极值点梯度的每个分量都是0。即最优的$\mathbf{w}_{\text{LIN}}$满足$\nabla E_{\text{in}}(\mathbf{w}_{\text{LIN}}) = \mathbf{0}$

将E_{in}的表达式展开并由以下两条矩阵求导规则（A是矩阵）

$$\begin{aligned}
\frac{\partial(\mathbf{x}^\mathsf{T}\mathrm{A}\mathbf{x})}{\partial\mathbf{x}} &= 2\mathrm{A}\mathbf{x} \\
\frac{\partial(\mathbf{x}^\mathsf{T}\mathbf{a})}{\partial\mathbf{x}} &= \mathbf{a}
\end{aligned}$$

可知

$$\begin{aligned}
E_{\text{in}}(\mathbf{w}) &= \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2 = \frac{1}{N}(\mathrm{X}\mathbf{w} - \mathbf{y})^\mathsf{T}(\mathrm{X}\mathbf{w} - \mathbf{y}) && \text{definition of norm} \\
&= \frac{1}{N}(\mathbf{w}^\mathsf{T}\mathrm{X}^\mathsf{T} - \mathbf{y}^\mathsf{T})(\mathrm{X}\mathbf{w} - \mathbf{y}) && \text{rule of transpose} \\
&= \frac{1}{N}(\mathbf{w}^\mathsf{T}\mathrm{X}^\mathsf{T}\mathrm{X}\mathbf{w} - \mathbf{w}^\mathsf{T}\mathrm{X}^\mathsf{T}\mathbf{y} - \mathbf{y}^\mathsf{T}\mathrm{X}\mathbf{w} + \mathbf{y}^\mathsf{T}\mathbf{y}) \\
&= \frac{1}{N}(\mathbf{w}^\mathsf{T}\mathrm{X}^\mathsf{T}\mathrm{X}\mathbf{w} - 2\mathbf{w}^\mathsf{T}\mathrm{X}^\mathsf{T}\mathbf{y} + \mathbf{y}^\mathsf{T}\mathbf{y}) && \mathbf{w}^\mathsf{T}\mathrm{X}^\mathsf{T}\mathbf{y} \text{ is a scalar, } a^\mathsf{T} = a \\
\nabla E_{\text{in}}(\mathbf{w}) &= \frac{1}{N}(2\mathrm{X}\mathbf{w} - 2\mathrm{X}^\mathsf{T}\mathbf{y}) = \frac{2}{N}(\mathrm{X}^\mathsf{T}\mathrm{X}\mathbf{w} - \mathrm{X}^\mathsf{T}\mathbf{y})
\end{aligned}$$

将$\nabla_{\text{in}}(\mathbf{w})$置为0，如果$\mathrm{X}^\mathsf{T}\mathrm{X}$是可逆的，那么可以容易解出

$$\mathbf{w}_{\text{LIN}} = (\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}\mathbf{y}$$

其中$(\mathrm{X}^\mathsf{T}\mathrm{X})^{-1}\mathrm{X}^\mathsf{T}$称为X的**伪逆**，记做$\mathrm{X}^\dagger$。由于通常有$N >> d+1$，因此$X^\dagger$通常都会存在

但是有少部分情况有$N < d+1$，这样会有很多组解。不过即便如此，使用其它定义伪逆矩阵的方法，仍然可以得到其中的一个解有$\mathbf{w}_{\text{LIN}} = \mathrm{X}^{\dagger}\mathbf{y}$

实践中如果需要计算$\mathbf{w}_{\text{LIN}}$，一般都直接用计算$\mathrm{X}^{\dagger}$的函数，而不是手动计算$(\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}$

因此线性回归算法可以定义如下：

> 1). 从数据集$\mathcal{D}$中构造输入矩阵X和输出向量$\mathbf{y}$：
>
> $$\mathrm{X} = \underbrace{\begin{bmatrix} -- & \mathbf{x}_1^{\mathsf{T}} & -- \\ -- & \mathbf{x}_2^{\mathsf{T}} & -- \\ & \cdots & \\ -- & \mathbf{x}_N^{\mathsf{T}} & -- \end{bmatrix}}_{N\times(d+1)} \quad \mathbf{y} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix}}_{N\times 1}$$
>
> 2). 计算伪逆 $\underbrace{\mathrm{X}^{\dagger}}_{(d+1)\times N}$
>
> 3). 计算最优权重 $\underbrace{\mathbf{w}_{\text{LIN}}}_{(d+1)\times 1} = \mathrm{X}^{\dagger}\mathbf{y}$

一般化问题

所以线性回归问题真的是一种机器学习问题，上述的算法真的是一种机器学习算法么？看起来最优权重的值一步就能算出来（这种解称为**解析解**或者**闭合形式解**），在计算过程中也没有迭代地减小E_{in}和E_{out}

实际上，这是一种机器学习算法，因为

- 它有好的E_{in}
- 变量有限，E_{out}是有限的
- 伪逆的计算过程是迭代的过程，其过程也可以算是在减小E_{in}

所谓“黑猫白猫，抓得住老鼠就是好猫”，只要$E_{\text{out}}(\mathbf{w}_{\text{LIN}})$足够小，这个算法就可以看作是一种好的算法。下面试着证明$\overline{E_{\text{in}}}$足够小，但是同样的思想可以用来证明E_{out}也足够小

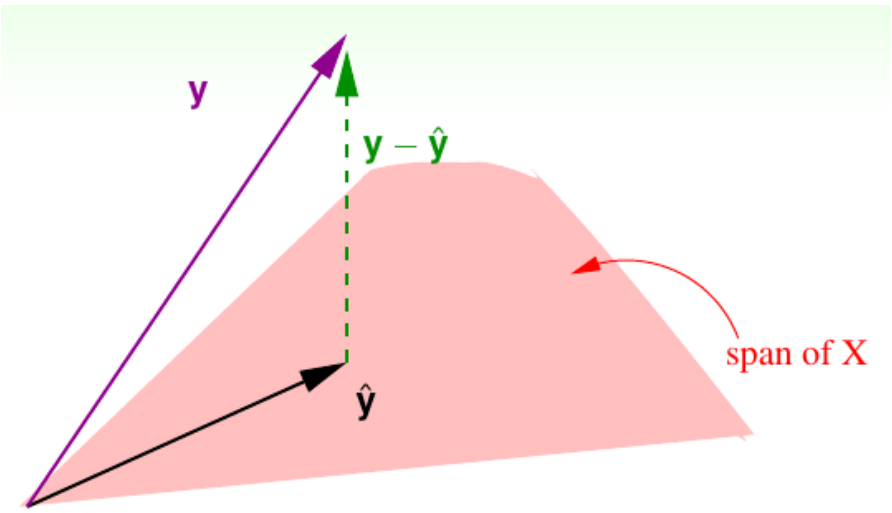
由于$\overline{E_{\text{in}}}$是对所有可能数据集E_{in}的平均值，而这些数据都来自于相同的分布，因此可以表示为

$$\overline{E_{\text{in}}} = \mathcal{E}_{\mathcal{D}\sim P^N}\{E_{\text{in}}(\mathbf{w}_{\text{LIN}} \text{ w.r.t } \mathcal{D})\}$$

其中每个E_{in}又可以如下表示

$$\begin{aligned} E_{\text{in}}(\mathbf{w}_{\text{LIN}}) &= \frac{1}{N}\|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \frac{1}{N}\|\mathbf{y} - \mathrm{X}\mathbf{w}_{\text{LIN}}\|^2 \\ &= \frac{1}{N}\|\mathbf{y} - \mathrm{X}\mathrm{X}^{\dagger}\mathbf{y}\|^2 = \frac{1}{N}\|(\mathrm{I} - \mathrm{X}\mathrm{X}^{\dagger})\mathbf{y}\|^2 \end{aligned}$$

称$\mathrm{X}\mathrm{X}^{\dagger}$为“帽子矩阵”，记为$\mathrm{H}$，因为真实值$\mathbf{y}$与之相乘会得到预测值$\hat{\mathbf{y}}$。帽子矩阵的几何意义可以通过下图解释



帽子矩阵的几何意义

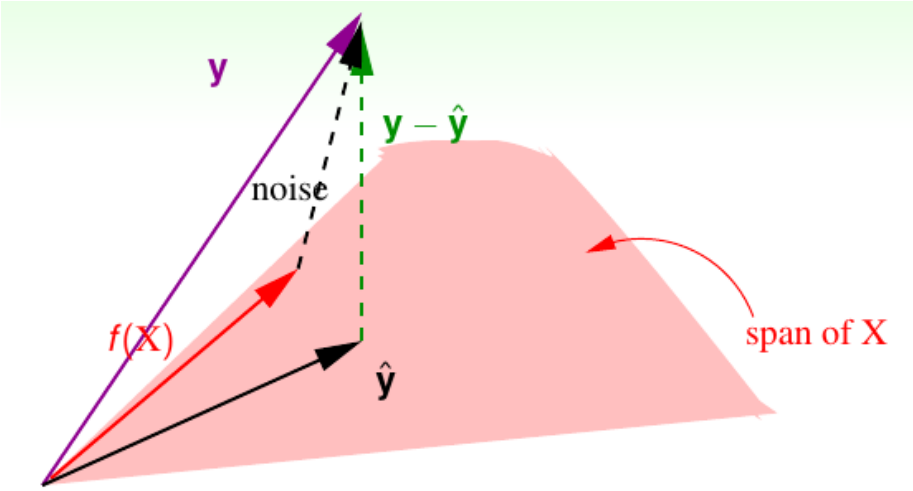
该图中，$\mathbf{y} \in \mathbb{R}^N$，而$\hat{\mathbf{y}}$是$\mathrm{X}$中每个列的线性组合，因此$\hat{\mathbf{y}}$属于$\mathrm{X}$的列空间。线性回归要让$\mathbf{y}$和$\hat{\mathbf{y}}$尽可能小，而$\hat{\mathbf{y}}$实际上是$\mathbf{y}$在$\mathrm{X}$列空间（下面简记为span ）中的投影，为了让两者的残差$\mathbf{y}-\hat{\mathbf{y}}$尽可能小，即让$\mathbf{y}$到span的距离最短，因此只能有$\mathbf{y}-\hat{\mathbf{y}}$ 垂直于 span。所以H起到的就是投影的作用，将任意向量$\mathbf{y}$投影到X的列空间中，投影成$\hat{\mathbf{y}}$，而$\mathrm{I}-\mathrm{H}$则是计算残差$\mathbf{y}-\hat{\mathbf{y}}$，它有一个很好的性质：

$$\operatorname{trace}(\mathrm{I}-\mathrm{H})=N-(d+1)$$

这条性质的物理意义在于，自由度为N的向量被投影到$d+1$维空间以后，残差的自由度最多只有$N-(d+1)$。证明如下（简记trace为tr，并记I_d是d维单位矩阵）：

$$\begin{aligned}
\operatorname{tr}(\mathrm{I}-\mathrm{H}) &= \operatorname{tr}(\mathrm{I}_n)-\operatorname{tr}(\mathrm{H}) && \operatorname{tr}(\mathrm{A}+\mathrm{B})=\operatorname{tr}(\mathrm{A})+\operatorname{tr}(\mathrm{B}) \\
&= N-\operatorname{tr}(\mathrm{H}) && \\
\operatorname{tr}(\mathrm{H}) &= \operatorname{tr}(\mathrm{X}\mathrm{X}^{\dagger}) && \\
&= \operatorname{tr}(\mathrm{X}(\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}) && \text{definition of } \mathrm{X}^{\dagger} \\
&= \operatorname{tr}((\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}\mathrm{X}) && \operatorname{tr}(\mathrm{AB})=\operatorname{tr}(\mathrm{BA}) \\
&= \operatorname{tr}(\mathrm{I}_{d+1}) && \\
&= d+1 && \\
\therefore \operatorname{tr}(\mathrm{I}-\mathrm{H}) &= N-(d+1) && \blacksquare
\end{aligned}$$

对$\mathbf{y}$引入误差，情况会略有变化，见下图



线性回归训练集内误差均值的计算

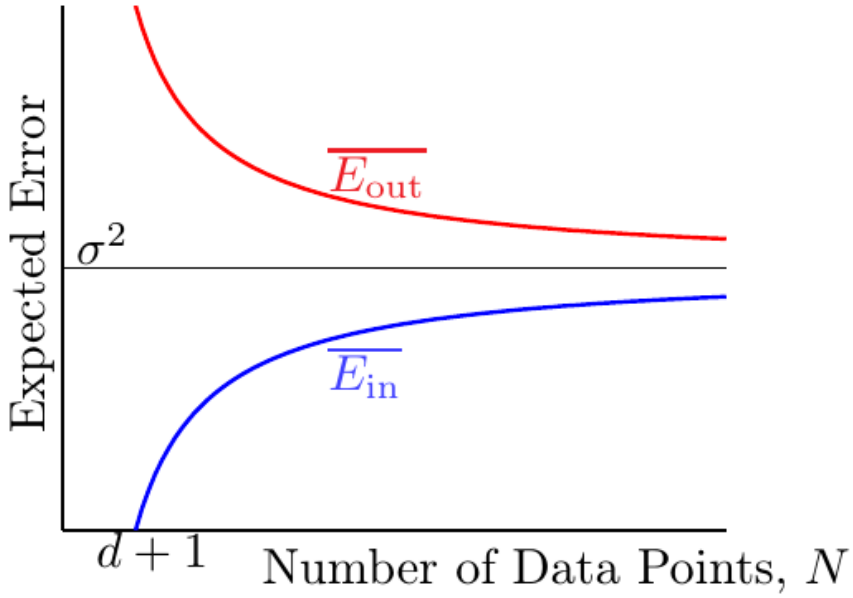
假设现在有一个理想的目标函数f，那么$f(\mathrm{X})$肯定会落在X的列空间中。现在，可以把观察到的$\mathbf{y}$看作是理想目标函数值与噪声的加和，即$\mathbf{y}=f(\mathrm{X})+\text{noise}$。由于要考察的残差仍然是$\mathbf{y}-\hat{\mathbf{y}}$，因此这个残差可以看作是噪声经过矩阵$\mathrm{I}-\mathrm{H}$的转换，即$E_{\text{in}}$就是把$\mathrm{I}-\mathrm{H}$用在噪声上。将这个关系代入到残差的表达式中，有：

$$\begin{aligned}
E_{\text{in}}(\mathbf{w}_{\text{LIN}}) = \frac{1}{N}\|\mathbf{y}-\hat{\mathbf{y}}\|^2 &= \frac{1}{N}\|(\mathrm{I}-\mathrm{H})\text{noise}\|^2 \\
&= \frac{1}{N}(N-(d+1))\|\text{noise}\|^2
\end{aligned}$$

因此可以得到E_{in}和E_{out}的均值（不过后者的计算更加复杂）

$$\begin{aligned}
\overline{E_{\text{in}}} &= \text{noise level}\cdot\left(1-\frac{d+1}{N}\right) \\
\overline{E_{\text{out}}} &= \text{noise level}\cdot\left(1+\frac{d+1}{N}\right)
\end{aligned}$$

哲学上的意义是，噪声的存在使得模型的学习过程中也对噪声数据进行了学习，学到的$\mathbf{w}$可能偏向了某个方向。由于测试数据是新鲜的，没看到过的，没用在学习过程中的数据，因此此消彼长，误差对偏向的方向惩罚更多。这样可以得到一个学习曲线图



学习曲线

可以看到，随着样本数量N的增大，E_{in}增大，E_{out}减小。但是最后随着$N \to \infty$，两者都会收敛到σ^2，即noise level。因此一般化误差（即E_{in}与E_{out}平均情况下相差多少）为$\frac{2(d+1)}{N}$

综上所述，当N尽可能大，噪声不影响时，学习是已经发生了的

用于二元分类的线性回归

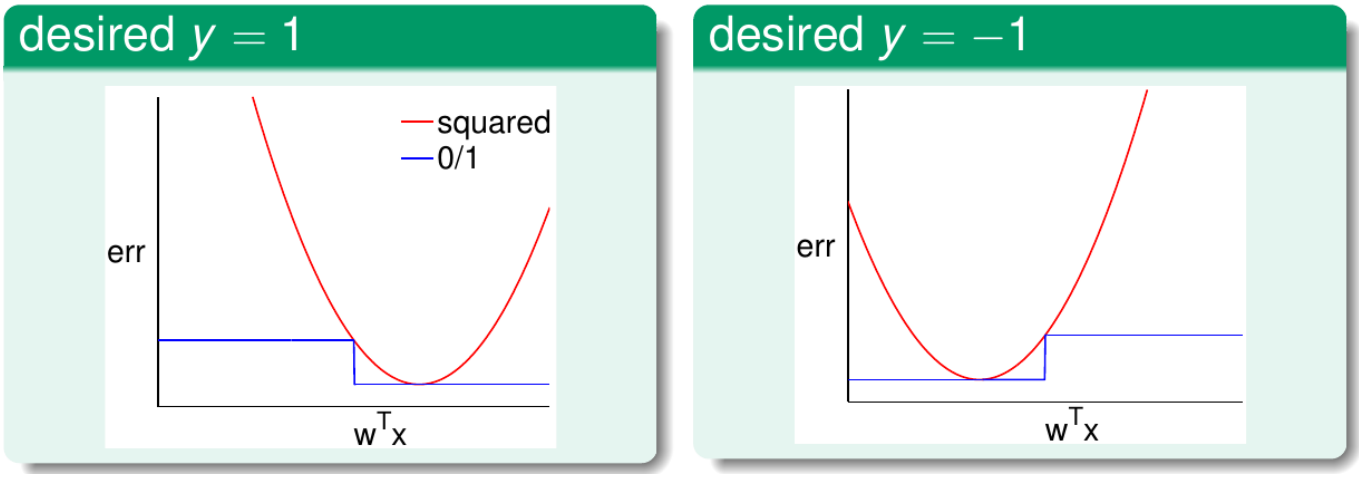
考虑之前提到的线性分类问题，已经被证明是NP难的。而线性回归却有非常有效的解析解可以一步得出结果。既然分类问题的结果$\{-1,+1\} \in \mathbb{R}$，也在线性回归的解空间里，那么可不可以用线性回归的思想求解分类问题呢？有人提出了一种做法：

1. 在数据集$\mathcal{D}$上运行线性回归算法，求出$\mathbf{w}_{\text{LIN}}$
2. 返回$g(\mathbf{x}) = \text{sign}(\mathbf{w}_{\text{LIN}}^{\mathsf{T}}\mathbf{x})$

如何说明这个做法是可行的呢？注意两个问题最大的区别是两者的误差函数定义不同：

- 0/1分类问题的损失函数为$\text{err}_{0/1} = [\![\text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x}) \neq y]\!]$
- 回归问题的损失函数为最小平方误差$\text{err}_{\text{sqr}} = \left(\mathbf{w}^{\mathsf{T}}\mathbf{x} - y\right)^2$

可以画出两个函数的曲线（横轴为$\mathbf{w}^{\mathsf{T}}\mathbf{x}$，纵轴为err），分情况讨论（期望$y=1$还是$y=-1$）



0/1误差函数与最小平方误差函数的图像比较。左为期望样本为正例的情况，右为期望样本为负例的情况

先看左边数据为正例的情况。由于此时数据为正例，因此其标签为1。最小平方误差函数以$\mathbf{w}^{\mathsf{T}}\mathbf{x}$为自变量时，函数图像是一个抛物线，在$\mathbf{w}^{\mathsf{T}}\mathbf{x} = y = 1$时取得最低点0。此时0/1分类误差函数也取得0，因为此时$\mathbf{w}^{\mathsf{T}}\mathbf{x}$的符号与$y$相同，不等号不成立，因此两函数交于$(1,0)$点。此外，当$\mathbf{w}^{\mathsf{T}}\mathbf{x} = 0$时，该点正好处在0/1分类误差函数变化的阶梯点上。但是由于$\mathbf{w}^{\mathsf{T}}\mathbf{x}$的符号跟$y$的符号不一致，因此0/1误差函数取1。而最小平方差函数在此时的值也为1（$(0-1)^2 = 1$），因此两函数还交于$(0,1)$点。画出所有图像以后，可以发现在$\mathbb{R}$上总有$\text{err}_{\text{sqrt}} \geq \text{err}_{0/1}$。同样的结论也可以用在右边数据为负例的情况。因此，由VC维的理论可以有

$$\begin{aligned}\text{classification } E_{\text{out}}(\mathbf{w}) &\leq \text{classification } E_{\text{in}}(\mathbf{w}) + \sqrt{\cdots\cdots} \\ &\leq \text{regression } E_{\text{in}}(\mathbf{w}) + \sqrt{\cdots\cdots}\end{aligned}$$

即回归问题的样本内误差加上一个常量，可以是分类问题样本外误差的一个上界。所以最优化的$\mathbf{w}_{\text{LIN}}$也可以解决分类问题。实践中，$\mathbf{w}_{\text{LIN}}$是一个不错的基线分类器，也可以用来做PLA算法或口袋算法的初始权重

NTUML 10. Logistic回归

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/08/30/NTUML-10-Logistic-Regression/
- **版权声明：** 本博客所有文章除特别声明外，均采用 CC BY-NC-SA 3.0 许可协议。转载请注明出处！

Logistic回归问题

二元分类问题，由于误差函数的定义，只关注样本**是/否**属于某个类别，即$\mathcal{Y} \in \{-1, +1\}$，理想的假设函数的形式也是
$f(\mathbf{x}) = \text{sign}(P(+1|\mathbf{x}) - \frac{1}{2}) \in \{-1, +1\}$

我们还会关心一个类似的问题。以医院中的例子为例，给定一个病人，可能不再只关心ta是否会得心脏病，而是还关注ta得心脏病的概率/风险/可能性（例如80%）。对这样的问题，要关心的是一个落在0到1之间的值。因此这个问题与原来的二元分类问题有所差别，被称作“软二元分类”问题。求解此问题得到的值越高，说明越有可能是○，否则越有可能是×。再次强调一下，这里的结果不再是非黑即白的二分法，而是一种可能性。此时目标函数会变化成$f(\mathbf{x}) = P(+1|\mathbf{x}) \in [0, 1]$的形式

理想状况下，要解决这样的问题，我们希望能看到给定$\mathbf{x}$数据是正例的概率，即类似如下的条目：

$$\begin{aligned}(\mathbf{x}_1, y_1') &= 0.9 = P(+1|\mathbf{x}_1) \\ (\mathbf{x}_2, y_2') &= 0.2 = P(+1|\mathbf{x}_2) \\ &\vdots \\ (\mathbf{x}_N, y_N') &= 0.6 = P(+1|\mathbf{x}_N)\end{aligned}$$

这样，只需要让g在数据上对概率的预测与实际概率差别越小越好。但是实际上，这样的资料是基本上拿不到的，上帝给的概率并不能看到，我们只能看到“是”或“否”，即真实数据是有噪声的版本，如下（尤其看最后一条）

$$\begin{aligned}(\mathbf{x}_1, y_1') &= 1 = [\![\circ \sim^? P(y|\mathbf{x}_1)]\!] \\ (\mathbf{x}_2, y_2') &= 0 = [\![\circ \sim^? P(y|\mathbf{x}_2)]\!] \\ &\vdots \\ (\mathbf{x}_N, y_N') &= 0 = [\![\circ \sim^? P(y|\mathbf{x}_N)]\!]\end{aligned}$$

因此，可以收集到的数据跟之前解二元分类时可以拿到的数据是一样的，只不过目标函数不同。因此，需要考察不同的假设函数集合

要确定假设函数的形式，第一件事还是要确定输入的形式和特征是什么。和之前一样，输入仍然可以表示为
$\mathbf{x} = (x_0, x_1, x_2, \ldots, x_d)$。一种自然的想法是，仍然计算其加权分数，即

$$s = \sum_{i=0}^{d} w_i x_i$$

但是值得关注的并不是这个加权分数，而是根据“分数越高概率越高，分数越低概率越低”这个设定以及“最后要得到一个概率值”这个要求，关注如何将结果缩放到[0, 1]这个区间。通常使用**Logistic函数**（记为θ）来将这个分数映射到$[0, 1]$区间，因此这个问题称为Logistic回归问题，假设也被称为Logistic假设。其形式为

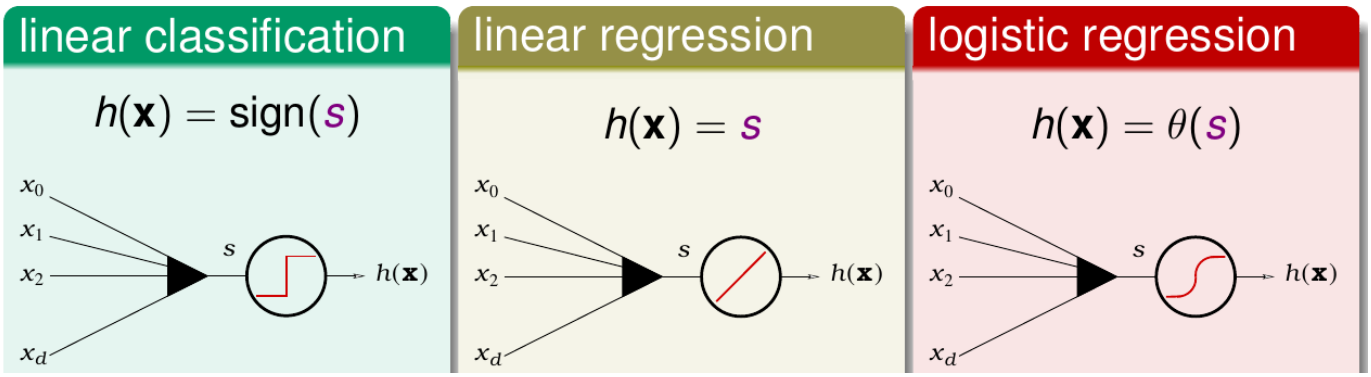
$$h(\mathbf{x}) = \theta(\mathbf{w}^{\mathsf{T}}\mathbf{x})$$

其中Logistic函数的定义为

$$\theta(s) = \frac{e^s}{1 + e^s} = \frac{1}{1 + e^{-s}}$$

Logistic回归误差

Logistic回归和之前提到的线性分类算法和线性回归算法有类似的地方，即对得到的特征都会做一个加权打分的操作。只不过三者在打分完以后做的事情有不同。可以参看下图



三种线性模型的比较

- 对于线性分类问题，算完得分以后，只根据得分的符号来做分类的动作。这里使用的是最简单的误差函数$\text{err}_{0/1}$
- 对于线性回归问题，算完得分以后，直接输出得分。误差函数使用平方误差，因为很容易最小化
- 对于Logistic回归问题，算完得分以后，再做一个logistic变换得到最后的结果。那么这里很自然需要关心的问题是，Logistic回归所使用的误差函数是什么?

考虑我们的目标函数为$f(\mathbf{x}) = P(+1|\mathbf{x})$，可以稍作变化得到给定$\mathbf{x}$时$y$的概率，即

$$P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases}$$

假设现在得到的数据集为

$$\mathcal{D} = \{(\mathbf{x}_1, \circ), (\mathbf{x}_2, \times), \ldots, (\mathbf{x}_N, \times)\}$$

根据前面的设定和贝叶斯定理，可以推出$\mathcal{D}$由f产生的概率，为

$$P(\mathbf{x}_1)P(\circ|\mathbf{x}_1) \times P(\mathbf{x}_2)P(\times|\mathbf{x}_2) \times \ldots \times P(\mathbf{x}_N)P(\times|\mathbf{x}_N)$$

将上式中的$P(y|\mathbf{x})$换成f，可以得到下面等价的式子

$$P(\mathbf{x}_1)f(\mathbf{x}_1) \times P(\mathbf{x}_2)(1 - f(\mathbf{x}_2)) \times \ldots \times P(\mathbf{x}_N)(1 - f(\mathbf{x}_N))$$

我们无法直接得到f，但是可以通过学习得到一个假设h，因此可以计算h生成同样数据集的概率。这个概率一般称之为**似然**。可以立即得到这个似然值为

$$P(\mathbf{x}_1)h(\mathbf{x}_1) \times P(\mathbf{x}_2)(1 - h(\mathbf{x}_2)) \times \ldots \times P(\mathbf{x}_N)(1 - h(\mathbf{x}_N))$$

如果有$h \approx f$，则h对$\mathcal{D}$的似然应该近似于f产生$\mathcal{D}$的概率。而通常f产生$\mathcal{D}$的概率应该非常大，就有

$$\text{likelihood}(h) \approx (\text{probability using } f) \approx \text{large}$$

因此可以认为，$\mathcal{H}$中最好的g应该是似然最大的那一个，即

$$g = \arg\max_h \text{likelihood}(h)$$

又因为logistic函数有对称性，即有

$$1 - h(\mathbf{x}) = h(-\mathbf{x})$$

因此

$$\text{likelihood}(\text{logistic } h) \propto \prod_{n=1}^{N} h(y_n \mathbf{x}_n)$$

Logistic回归算法的目的就是找到一个h使得上式最大化。同样由于每个h都关联一个$\mathbf{w}$，因此代入h的定义，目标可以写为

$$\max_{\mathbf{w}} \text{likelihood}(\mathbf{w}) \propto \prod_{n=1}^{N} \theta(y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n)$$

又因为$\log(ab) = \log a + \log b$以及对已有函数取对数不会改变函数的单调性，有

$$
\begin{aligned}
&\max_{\mathbf{w}} \prod_{n=1}^{N} \theta(y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n) \\
\Leftrightarrow &\max_{\mathbf{w}} \ln \prod_{n=1}^{N} \theta(y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n) \\
\Leftrightarrow &\max_{\mathbf{w}} \sum_{n=1}^{N} \ln \theta(y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n) \\
\Leftrightarrow &\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} -\ln \theta(y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n) \\
\Leftrightarrow &\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \ln(1+\exp(-y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n)) \qquad (\text{definition of logistic function})
\end{aligned}
$$

定义

$$
\mathrm{err}(\mathbf{w}, \mathbf{x}, y) = \ln(1+\exp(-y\mathbf{w}^{\mathsf{T}}\mathbf{x}))
$$

记为交叉熵误差函数，那么$\sum_{n=1}^{N} \mathrm{err}(\mathbf{w}, \mathbf{x}_n, y_n)$就是Logistic回归问题中的$E_{\mathrm{in}}$

Logistic回归误差的梯度

有了误差函数，接下来的问题自然就是怎么求出最优的$\mathbf{w}$。经过分析可以发现，交叉熵误差函数也是连续、可微、二次可微的凸函数，因此第一个想法就是试着推导$\nabla E_{\mathrm{in}}(\mathbf{w})$。求导有如下结果

$$
\begin{aligned}
\frac{\partial E_{\mathrm{in}}(\mathbf{w})}{\partial \mathbf{w}} &= \frac{1}{N} \sum_{n=1}^{N} \frac{1}{1+\exp(-y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n)} \cdot \exp(-y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n) \cdot (-y_n \mathbf{x}_n) \\
&= \frac{1}{N} \sum_{n=1}^{N} \theta(-y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n)(-y_n \mathbf{x}_n)
\end{aligned}
$$

接下来试着求解$\nabla E_{\mathrm{in}}(\mathbf{w}) = 0$。注意梯度是用$\theta$加权$y_n \mathbf{x}_n$的，由于梯度是一个求和项，那么如果所有$\theta$都是0，那么求和项就会为0。而当每个$y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n \to +\infty$时，$\theta(-y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n)$才会趋近于0。由于此时每个$y_n \mathbf{w}^{\mathsf{T}} \mathbf{x}_n$都是一个巨大的正数，因此有之前感知机部分的知识可以得知，每条数据都被正确分类，这意味着数据集$\mathcal{D}$是线性可分的。而大部分情况下，这个条件不能满足，这就要求我们要想办法去解这个方程。而这个式子并不是一个线性的式子。实际上$\nabla E_{\mathrm{in}}(\mathbf{w}) = 0$并没有解析解

所以如何破？回顾之前的PLA算法，它是迭代的，发现算法犯错的时候会修正$\mathbf{w}$。为了简单起见，可以把“找出使$\mathbf{w}_t$犯错的数据$(\mathbf{x}_{n(t)}, y_{n(t)})$”这一步和“改正$\mathbf{w}_t$这一步合并，连写为

$$
\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + [\![\mathrm{sign}(\mathbf{w}_t^{\mathsf{T}} \mathbf{x}_n) \neq y_n]\!] y_n \mathbf{x}_n
$$

这里其实有两样东西，其一是更新的方向（这里是$[\![\mathrm{sign}(\mathbf{w}_t^{\mathsf{T}} \mathbf{x}_n) \neq y_n]\!] y_n \mathbf{x}_n$），另一是更新的步长（这里就是1，没有在上式写出）。如果在这里要对更新的方向$\mathbf{v}$和更新的步长η做出改变，就会得到不同的机器学习算法。这样的算法一般称为**迭代优化方法**。

梯度下降

迭代优化方法的思想可以如下表示。其中$\mathbf{v}$代表更新的方向，η代表步长

> 对$t = 0, 1, \ldots$
>
> $$
> \mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \mathbf{v}
> $$
>
> 算法终止时，将最后得到的$\mathbf{w}$返回为g

PLA里通过修正错误来得到$\mathbf{v}$。而Logistic回归，前面可以看到，其损失函数是一个光滑的曲线，这能带来什么样的启发？想象将一个球放在山坡的某个地方，它自然会滚下去，到谷底的时候停住，而停住的点就是梯度为0的点。因此，解Logistic回归问题时，所选的$\mathbf{v}$就是这里球滚下去的方向，其中为了推导的方便，假设$\mathbf{v}$的长度是1。比较贪心的想法是，希望小球滚下去时每一步的跨越最大（E_{in}下降最多），往陡的方向滚。假设限制了每一步只能走η这么大，那么问题就化为找到满足下式的$\mathbf{v}$

$$
\min_{|\mathbf{v}|=1} E_{\mathrm{in}}(\underbrace{\mathbf{w}_t + \eta \mathbf{v}}_{\mathbf{w}_{t+1}})
$$

也就是如何找到一个最好的方向。然而这个问题仍然是一个不好解的问题：仍然是要优化一个非线性函数，而且更丧心病狂的是居然还加了一个条件。因此，这个问题并不比$\min_{\mathbf{w}} E_{\mathrm{in}}(\mathbf{w})$好解。考虑到目前学过的容易的问题都跟线性函数有关，有没有办法把上面的式

子变成一个关于$\mathbf{v}$的线性函数?

答案是可以的。任何曲线如果只观察其非常微小的一段，这一段都可以认为是一条长度很短的直线。因此在η特别小时，使用泰勒展开，有

$$E_{\text{in}}(\mathbf{w}_t + \eta\mathbf{v}) \approx E_{\text{in}}(\mathbf{w}_t) + \eta\mathbf{v}^{\mathsf{T}}\nabla E_{\text{in}}(\mathbf{w}_t)$$

因此问题转化为，当η很小时，求

$$\min_{|\mathbf{v}|=1} E_{\text{in}}(\mathbf{w}_t) + \eta\mathbf{v}^{\mathsf{T}}\nabla E_{\text{in}}(\mathbf{w}_t)$$

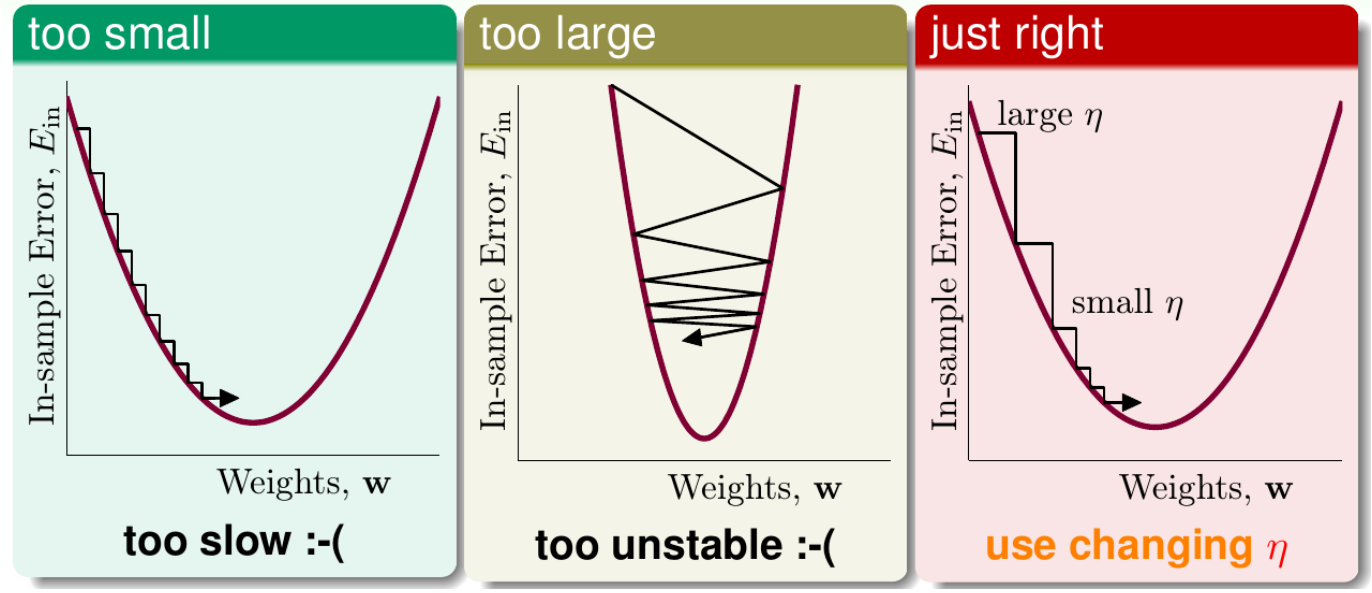
由于E_{in}是常数，因此最好的$\mathbf{v}$与梯度方向相反就可以。又因为$\mathbf{v}$是单位向量，因此

$$\mathbf{v} = \frac{-\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$$

所以理想的更新方法（称为**梯度下降**）为

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta\frac{\nabla E_{\text{in}}(\mathbf{w}_t)}{\|\nabla E_{\text{in}}(\mathbf{w}_t)\|}$$

注意这里η不能太小也不能太大。如果η太小，收敛会很慢；如果η太大，曲线的这一小段就不能看作是一段很短的直线，逼近不准，因此收敛不稳定。具体示例可见下图



eta取不同值时w的收敛状况

最好的方法是η要经常改变。梯度越大，η越大；梯度越小，η越小，即$\eta \propto \|\nabla E_{\text{in}}(\mathbf{w}_t)\|$。这个关系也可以表示为$\eta = \eta_0 \cdot \|\nabla E_{\text{in}}(\mathbf{w}_t)\|$，而稍作变换就有$\eta_0 = \eta/\|\nabla E_{\text{in}}(\mathbf{w}_t)\|$。如果把$\eta_0$用一个新的$\eta$替换，上式就可以写为

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta\nabla E_{\text{in}}(\mathbf{w}_t)$$

称此时的η为**固定的学习率**，上式也被称为固定学习率的梯度下降

因此，Logistic回归算法可以表示如下：

> 初始化$\mathbf{w}_0$
>
> 当$t = 0, 1, \ldots$时
>
> 计算
>
> $$\nabla E_{\text{in}}(\mathbf{w}_t) = \frac{1}{N}\sum_{n=1}^{N}\theta(-y_n\mathbf{w}_t^{\mathsf{T}}\mathbf{x}_n)(-y_n\mathbf{x}_n)$$
>
> 更新权重
>
> $$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \eta\nabla E_{\text{in}}(\mathbf{w}_t)$$

> 直到$\nabla E_{\text{in}}(\mathbf{w}_{t+1}) = 0$或迭代次数足够多
>
> 将最后得到的$\mathbf{w}_{t+1}$返回为g

可以看到，Logistic回归算法的时间复杂度与口袋算法每次迭代的时间复杂度相似

NTUML 11. 用于分类问题的线性模型

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/09/01/NTUML-11-Linear-Models-for-Classification/

用于二元分类的线性模型

之前提到的三种线性模型都使用了$s = \mathbf{w}^{\mathsf{T}}\mathbf{x}$这个函数来计算得分

- 线性分类直接对这个分值取正负号，使用0/1误差来做误差函数。$E_{\text{in}}(\mathbf{w})$是离散的，优化是NP-hard问题
- 线性回归对这个分值不作处理直接输出，因此值域是$\mathbb{R}$，误差函数是平方误差。这个$E_{\text{in}}(\mathbf{w})$优化起来比较友好，能直接得到一个闭合解
- Logistic回归对分值做一个logistic变换$\theta(s)$，值域是$[0, 1]$，使用交叉熵函数做误差函数。这个$E_{\text{in}}(\mathbf{w})$可以用梯度下降方法求解

接下来首先探讨一个问题：线性回归或Logistic回归可以用来求解线性分类问题吗？首先要整合一下这些误差函数，将关于$h, \mathbf{x}, y$的函数转化为关于s, y的函数。其中$s = \mathbf{w}^{\mathsf{T}}\mathbf{x}, y \in \{-1, +1\}$

对线性分类问题，有

$$
\begin{aligned}
& & h(\mathbf{x}) &= \text{sign}(s) \\
& & \text{err}(h, \mathbf{x}, y) &= [\![h(\mathbf{x}) \neq y]\!] \\
\Rightarrow & & \text{err}_{0/1}(s, y) &= [\![\text{sign}(s) \neq y]\!] \\
& & &= [\![\text{sign}(ys) \neq 1]\!]
\end{aligned}
$$

对线性回归问题，有

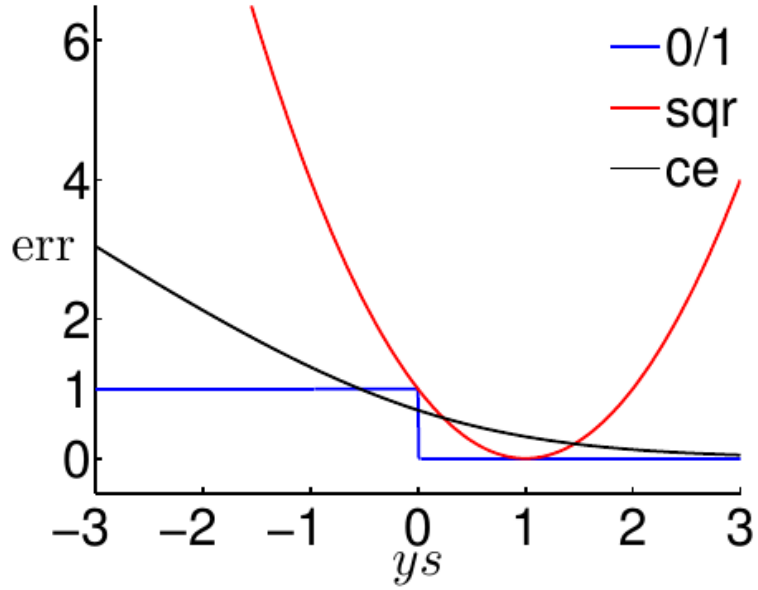
$$
\begin{aligned}
& & h(\mathbf{x}) &= s \\
& & \text{err}(h, \mathbf{x}, y) &= (h(\mathbf{x}) - y)^2 \\
\Rightarrow & & \text{err}_{\text{SQR}}(s, y) &= (s - y)^2 \\
& & &= (ys - 1)^2
\end{aligned}
$$

对Logistic回归问题，有

$$
\begin{aligned}
& & h(\mathbf{x}) &= \theta(s) \\
& & \text{err}(h, \mathbf{x}, y) &= -\ln h(y\mathbf{x}) \\
\Rightarrow & & \text{err}_{\text{CE}}(s, y) &= \ln(1 + \exp(-ys))
\end{aligned}
$$

可以看到，这三个误差函数里都包括了ys这一项。那么该项的物理意义是什么？这里y是正确度，s是得分，因此两者相乘可以看作是“分类的正确度得分”：两者相乘得到的结果越大，说明越正确；否则说明越不正确。这个做法实际很符合逻辑：假设$\mathbf{w}^{\mathsf{T}}\mathbf{x}$是一个比较大的正数，那么由Logistic回归的物理意义这说明$y = 1$的概率会非常大。假设实际上$y = -1$，那么两者相乘就是一个比较小的负数，说明是有问题的

将上面得到的三个函数图像作到一个坐标系里，可以得到如下的图



三种误差函数的图象

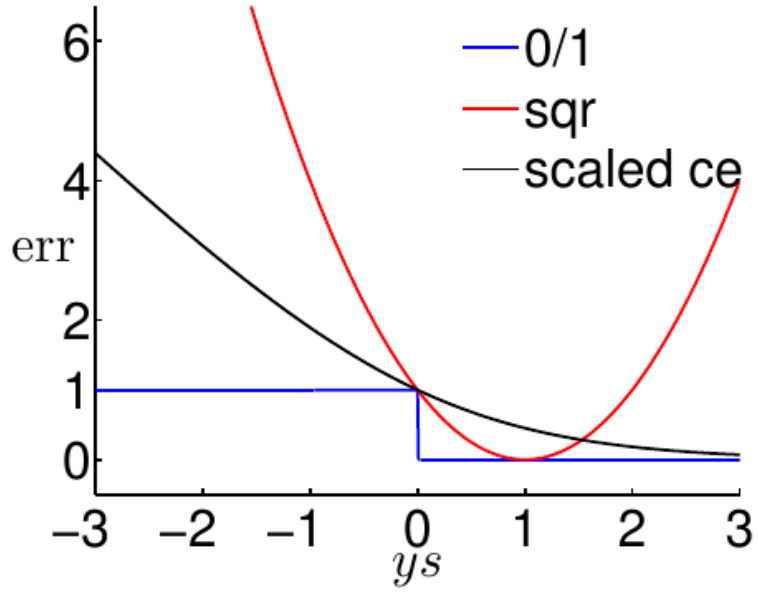
可以看到一些现象：

- 对于0/1误差函数，如果$ys \leq 0$，说明y与s异号，发生了分类错误，因此$\mathrm{err}_{0/1}$为1。如果$ys > 0$，说明y与s同号，分类是正确的，$\mathrm{err}_{0/1}$为0
- 对于平方误差函数，可以看到如果$ys \ll 1$则误差函数会得到一个很大的值，即对严重的错分现象施加了比较大的惩罚。但是，由于二次函数抛物线的性质，可以看到如果$ys \gg 1$，即分类非常正确时，误差函数也会加以很大的惩罚，所以误差函数的表现并不总是尽人意。不过无论如何，如果平方误差小，一定会说明0/1误差小
- 对于交叉熵误差函数，可以看到它是关于ys单调递减的。交叉熵误差越小，说明0/1误差越小，反之亦然。

为了推导的方便，可以对交叉熵误差函数做一个缩放（实际上就是换了一个底）。缩放后的交叉熵误差函数修正如下：

$$\mathrm{err}_{\mathrm{SCE}}(s, y) = \log_2(1 + \exp(-ys))$$

修改后的交叉熵误差函数能保证当$ys = 0$时交叉熵误差函数和0/1误差函数相切，即图像变为下图



对交叉熵误差函数换底以后，三个误差函数图像的比较

可以看到，换底以后，交叉熵误差函数成为了0/1误差函数严格的上界。也就是说，对任意ys（其中$s = \mathbf{w}^{\mathrm{T}}\mathbf{x}$），有

$$\begin{aligned} \mathrm{err}_{0/1}(s, y) \leq \mathrm{err}_{\mathrm{SCE}}(s, y) &= \frac{1}{\ln 2}\mathrm{err}_{\mathrm{CE}}(s, y) \\ \Longrightarrow E_{\mathrm{in}}^{0/1}(\mathbf{w}) \leq E_{\mathrm{in}}^{\mathrm{SCE}}(\mathbf{w}) &= \frac{1}{\ln 2}E_{\mathrm{in}}^{\mathrm{CE}}(\mathbf{w}) \\ E_{\mathrm{out}}^{0/1}(\mathbf{w}) \leq E_{\mathrm{out}}^{\mathrm{SCE}}(\mathbf{w}) &= \frac{1}{\ln 2}E_{\mathrm{out}}^{\mathrm{CE}}(\mathbf{w}) \end{aligned}$$

套用之前的VC维理论，有

$$\begin{aligned} E_{\text{out}}^{0/1}(\mathbf{w}) &\leq E_{\text{in}}^{0/1}(\mathbf{w}) + \Omega^{0/1} \\ &\leq \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w}) + \Omega^{0/1} \end{aligned}$$

或者从另一个方向，有

$$\begin{aligned} E_{\text{out}}^{0/1}(\mathbf{w}) &\leq \frac{1}{\ln 2} E_{\text{out}}^{\text{CE}}(\mathbf{w}) \\ &\leq \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w}) + \frac{1}{\ln 2} \Omega^{CE} \end{aligned}$$

无论从哪个方向，都可以得出结论

$$\text{small } E_{\text{in}}^{\text{CE}}(\mathbf{w}) \Rightarrow \text{small } E_{\text{out}}^{0/1}(\mathbf{w})$$

平方误差同理。因此，这证明，Logistic回归和线性回归都可以用在线性分类问题上，算法为

1. 在$y_n \in \{-1, +1\}$的$\mathcal{D}$上运行Logistic回归或线性回归算法得到$\mathbf{w}_{\text{REG}}$
2. 返回$g(\mathbf{x}) = \text{sign}(\mathbf{w}_{\text{REG}}^{\mathsf{T}} \mathbf{x})$

使用线性回归的好处是优化变得容易，但是对$|ys|$比较大的情况，其得到的误差比0/1误差宽松太多。使用Logistic回归的好处是优化也比较容易，但是类似于线性回归，它对$ys \ll 0$的情况得到了太宽松的上界。另一方面，在数据集线性可分时，PLA非常有效而且效果很强，但是对线性不可分的情况只能使用启发式的口袋算法

在实践中，通常把线性回归得到的权重用来初始化PLA/口袋算法/Logistic回归的$\mathbf{w}_0$。另外，大部分人会使用Logistic回归，而少用口袋算法

随机梯度下降

PLA、口袋算法和Logistic回归算法都使用了迭代优化的方法，权重$\mathbf{w}$会一步步地越变越好。在PLA算法中，每一个错误点的发现都会导致权重的更改，每次迭代的时间复杂度仅为$O(1)$。然而，Logistic回归的每次迭代会看输入数据集$\mathcal{D}$上的每个点，将所有数据点的梯度算出来，求和以后算出总梯度，再去产生$\mathbf{w}_{t+1}$。因此，Logistic回归每次迭代的时间复杂度为$O(N)$

有没有可能提高效率，让Logistic回归每次迭代的时间复杂度也变为$O(1)$？回顾Logistic回归的迭代方法

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \underbrace{\frac{1}{N} \sum_{n=1}^{N} \theta(-y_n \mathbf{w}_t^{\mathsf{T}} \mathbf{x}_n)(y_n \mathbf{x}_n)}_{-\nabla E_{\text{in}}(\mathbf{w}_t)}$$

要让每次迭代更新的时间复杂度为$O(1)$，需要同时满足两个要求：首先，方向$\mathbf{v}$不应该变化太多，还应该有$\mathbf{v} \approx -\nabla E_{\text{in}}(\mathbf{w}_t)$。这是因为由前面的推导，这个方向梯度的减小是最快的。其次，希望计算$\mathbf{v}$时只用到一个点$(\mathbf{x}_n, y_n)$，因为只有这样才能保证时间复杂度。所以，原式中的求和项一定要想办法去掉

如何去掉这个求和项？注意到式子中有一个$\frac{1}{N} \sum_{n=1}^{N}$。可以很自然地将其与期望$\mathcal{E}$的概念联系起来：如果随机且均匀地选取了$n$个点求它们梯度的期望，就应该是这样计算。假如每次只随机取一个点，将这个点的梯度看作是梯度的期望，就可以避免求和的过程。这种做法叫做**随机梯度下降**（SGD），将整体的梯度看作是选出的若干点（“若干”可以为1）梯度的期望，即

$$\nabla_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) = \mathcal{E}_{\text{random } n} \nabla_{\mathbf{w}} \text{err}(\mathbf{w}, \mathbf{x}_n, y_n)$$

SGD的Logistic回归迭代为：

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \underbrace{\theta(-y_n \mathbf{w}_t^{\mathsf{T}} \mathbf{x}_n)(y_n \mathbf{x}_n)}_{-\nabla \text{err}(\mathbf{w}_t, \mathbf{x}_n, y_n)}$$

可以将随机梯度看作是真实梯度加上了一些期望为0的噪声：虽然总体上还是沿真实梯度方向前进，但是每一步都会向随机的某个方向偏一点。SGD的做法就是把原来算法中真实的梯度换成随机的梯度。由于噪声的期望为0，因此经过足够多的迭代以后，真实梯度的平均也约等于随机梯度的平均。这样的好处是，计算量减小了很多，尤其适合于大数据环境或在线学习的状况。但是，SGD会比较不稳定（尤其是η很大的情况下）

回顾PLA的迭代操作

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + 1 \cdot [\![y_n \neq \text{sign}(\mathbf{w}_t^{\mathsf{T}} \mathbf{x}_n)]\!] (y_n \mathbf{x}_n)$$

跟前面SGD Logistic回归相比，可以把SGD Logistic回归看作是“软化的”PLA：不再非黑即白地根据其对错来更新，而是根据错误程度更新：错得多，更新得多；错得少，更新得也少。反过来看，当$\mathbf{w}_t^{\mathsf{T}}\mathbf{x}_n$很大时，PLA可以看作是$\eta = 1$的SGD Logistic回归

真正使用SGD Logistic回归时，有两条实践经验：

- 停止条件一般是设定迭代次数t足够大即可
- η一般设置为0.1

使用Logistic回归做多元分类

已有的这些分类算法不仅可以做“判断题”（二元分类），也可以做“选择题”（多元分类）。假设要分的类不再是○和×两类，而是□、◇、△和⋆四类，应该怎么做？可以使用线性分类器逐个看样本是否属于这四类，例如，判断样本是否为□时，可以划归到所有□是○，剩下三类都是×的情况，以此类推

在大部分情况下，这种方法都能做出不错的选择。但是有两种情况可能需要费一些周章：其一，对某些样本，可能有两个（或以上）分类器都认为它应该属于自己的类别；另一种情况是，对某些样本，可能有两个（或以上）分类器都认为它不应该属于自己的类别

因此，可以使用Logistic回归来“软化”，不再计算每个样本是否属于某个类别，而是计算每个样本属于每个类别的概率。仍然以判断样本是否为□时的状况为例，此时还是所有□都是○，剩下三类样本都是×，但是这时计算的是$P(\square|\mathbf{x})$，最好的假设函数为

$$g(\mathbf{x}) = \arg\max_{k\in\mathcal{Y}} \theta\left(\mathbf{w}_{[k]}^{\mathsf{T}}\mathbf{x}\right)$$

因此多元分类的一种算法如下（笔记注：讲义上写的算法名称叫做OVA（One-Versus-All），但是sklearn包偏爱的称法为OVR（One-Versus-Rest））

> 对每个$k \in \mathcal{Y}$
>
> 　　将数据集转化为如下形式，然后运行Logistic回归算法以获取$\mathbf{w}_{[k]}$
>
> $$\mathcal{D}_{[k]} = \{(\mathbf{x}_n, y_n' = 2[\![y_n = k]\!] - 1)\}_{n=1}^N$$
>
> 返回$g(\mathbf{x}) = \arg\max_{k\in\mathcal{Y}} \theta\left(\mathbf{w}_{[k]}^{\mathsf{T}}\mathbf{x}\right)$

这个方法的好处是很有效（只需要运行k次），而且可以适用于任何类似于Logistic回归的算法（只要这个算法可以输出概率）。但是如果类别数很多，会导致**不平衡类别**问题的出现，即正负样本的比例会差得比较多。在这种情况下，所有Logistic回归分类器都倾向于猜×（笔记注：实际是给出一个特别小的概率）。尽管也可以在这些小概率中选择最大的概率作为样本所属类别，但是在这种情况下Logistic回归的表现一般不会太好。统计学中还有一种对Logistic回归的扩展，称为multinomial logistic regression，不过这里没有做太多介绍

尽管存在着问题，OVA算法仍然是一种简单但是特别有效的，可适用于多类别分类的基础算法

使用二元分类做多元分类

前面说到，OVA面临的最大问题是，当类别数很多时，问题会变成不平衡样本分类问题。其原因是将每个类别视作正例时，都是所有样本参与分类过程。如果有K个类，属于每个类的数据量都近似，那么每次分类时正负样本数量比都是$1:K-1$。为了解决这个问题，可以考虑如何在分类时仍然让正负样本数接近1比1。因此，可以使用一对一（OVO, One-Versus-One）的分类法：每次取出K个类别中属于某两类的样本进行比较，其它样本都不参与分类。这样，总共有$\binom{K}{2}$个分类步骤，得到$\binom{K}{2}$个分类器。对于任何一条数据，看这$\binom{K}{2}$个分类器各自认为它属于哪个类别，最后哪个类别收到的票数多，哪个类别获胜。可以形象地认为，K个类别就这个数据两两作赛，谁赢的次数多谁就是冠军。因此OVO分类法返回的假设函数为

$$g(\mathbf{x}) = \text{tournament champion}\ \{\mathbf{w}_{[k,\ell]}^{\mathsf{T}}\mathbf{x}\}$$

多元分类的OVO算法总体过程如下

> 对每个类别对$(k,\ell) \in \mathcal{Y}\times\mathcal{Y}$
>
> 　　在如下数据集上运行线性二元分类器算法，获得$\mathbf{w}_{[k,\ell]}$
>
> $$\mathcal{D}_{[k,\ell]} = \{(\mathbf{x}_n, y_n' = 2[\![y_n = k]\!] - 1) : y_n = k \text{ or } y_n = \ell\}$$
>
> 返回$g(\mathbf{x}) = \text{tournament champion}\ \{\mathbf{w}_{[k,\ell]}^{\mathsf{T}}\mathbf{x}\}$

这个算法是每次分类过程效率比较高（因为只使用了少量样本）、稳定，而且也可以适用于任何二元分类算法。但是，算法需要更多空间（存储更多的$\mathbf{w}$），预测更慢（因为要跑$\binom{K}{2}$个分类过程，训练更多

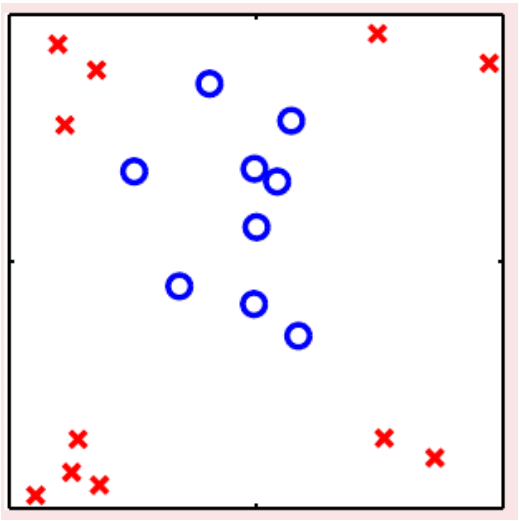
同样，OVO算法仍然是一种简单但是特别有效的，可适用于多类别分类的基础算法

NTUML 12. 非线性变换

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/09/04/NTUML-12-Nonlinear-Transformation/

二次函数假设

二维平面上，解决二元分类问题的分类器可以看做是一条直线（在更高维空间上，是一个超平面），其核心思想是使用权重向量$\mathbf{w}$对输入$\mathbf{x}$算一个分数$s = \mathbf{w}^{\mathsf{T}}\mathbf{x}$，然后对得到的分数做进一步处理。这种做法的好处是其VC维能够受到控制，因此E_{in}和E_{out}不会差太远。但是对于某些数据$\mathcal{D}$，例如下图中给出的这种，可以发现无论怎么画线，都很难很好地将数据集分开。也就是说，不论怎么画线，E_{in}都会非常大，因此E_{out}也会很大。这样，线性模型在这样的数据集上无论如何都学不好。基于这样的状况，如何突破线性模型的限制成为了必须要解决的问题



非线性可分数据集的例子

经过观察，可以发现，尽管上图中的$\mathcal{D}$并不是线性可分，但是可以用一个圆（半径为$\sqrt{0.6}$）将这些数据都分开，因此，可以用假设函数

$$h_{\text{SEP}}(\mathbf{x}) = \text{sign}(-x_1^2 - x_2^2 + 0.6)$$

来对这个数据集分类。但是我们需要重头推导一遍圆形可分的数据集的算法吗？并不用。如果将上式中的未知数和变量写明确，可以转化为如下形式

$$h(\mathbf{x}) = \text{sign}\left(\underbrace{0.6}_{\tilde{w}0}\cdot\underbrace{1}_{z0} + \underbrace{(-1)}_{\tilde{w}1}\cdot\underbrace{x_1^2}_{z1} + \underbrace{(-1)}_{\tilde{w}2}\cdot\underbrace{x_2^2}_{z2}\right)$$

如上式所示，将x_1^2视作新变量z_1，x_2^2视作新变量z_2，则上式可以简写为

$$h(\mathbf{z}) = \text{sign}(\tilde{\mathbf{w}}^{\mathsf{T}}\mathbf{z})$$

这意味着，使用上面的变换方法，在$\{(\mathbf{x}_n, y_n)\}$上圆形可分的数据，在$\{(\mathbf{z}_n, y_n)\}$上会变得线性可分。这种将$\mathcal{X}$的点变换为$\mathcal{Z}$中点的方法（记为$\mathbf{\Phi}$）

$$\mathbf{x} \in \mathcal{X} \xrightarrow{\mathbf{\Phi}} \mathbf{z} \in \mathcal{Z}$$

称为一个**（非线性）特征变换**

既然$\mathcal{X}$上的圆形分类器都可以用$\mathcal{Z}$上的一条直线来表示，那么反过来是不是对的？即$\mathcal{Z}$上的一条直线是不是对应了$\mathcal{X}$上的一个圆？按照之前$\mathbf{\Phi}$的做法，有

$$(z_0, z_1, z_2) = \mathbf{z} = \mathbf{\Phi}(\mathbf{x}) = (1, x_1^2, x_2^2)$$

那么

$$h(\mathbf{x}) = \tilde{h}(\mathbf{z}) = \text{sign}(\tilde{\mathbf{w}}^\mathsf{T}\mathbf{\Phi}(\mathbf{x})) = \text{sign}(\tilde{w}_0 + \tilde{w}_1 x_1^2 + \tilde{w}_2 x_2^2)$$

根据解析几何的知识，对$\tilde{\mathbf{w}} = (\tilde{w}_0, \tilde{w}_1, \tilde{w}_2)$，尽管$\mathcal{Z}$中是一条直线，但根据权重每个分量的取值不同，对应到$\mathcal{X}$上可能是不同的二次曲线，例如

- $(0.6, -1, -1)$对应了一个圆（将圆内的点判断为正例）
- $(-0.6, +1, +1)$对应了一个圆（将圆外的点判断为正例）
- $(-0.6, -1, -2)$对应了一个椭圆
- $(-0.6, -1, +2)$对应了一个双曲线
- $(0.6, +1, +2)$这个分类器会将所有样本都分为正例

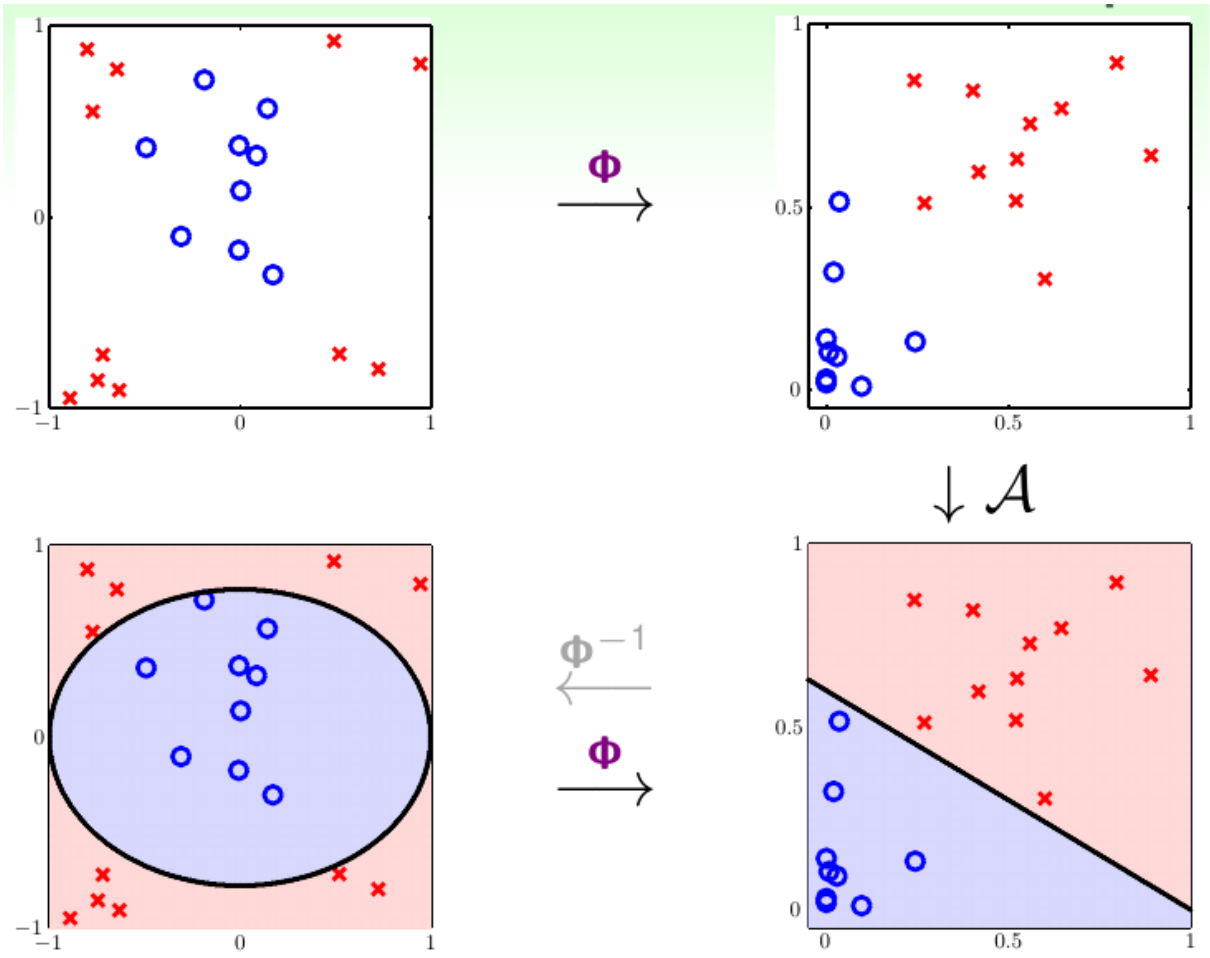
注意这里尽管得到的二次曲线不同，但是它们会有一些共同的限制。例如，当$w_1 = w_2$且与w_0异号时，得到的圆虽然半径不同，但是其圆心始终会在原点上。要打破这样的限制，就需要$\mathbf{\Phi}$做的变换包含了x_1与x_2二次（及更低次）组合的各种形式，即$\mathbf{\Phi}_2(\mathbf{x}) = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$。重新记$\mathcal{X}$经$\mathbf{\Phi}_2$变换后得到的空间为$\mathcal{Z}$，则假设集合$\mathcal{H}_{\mathbf{\Phi}_2} = \{h(\mathbf{x}) : h(\mathbf{x}) = \tilde{h}(\mathbf{\Phi}_2(\mathbf{x})) \text{ for some linear } \tilde{h} \text{ on } \mathcal{Z}\}$就可以实现$\mathcal{X}$中的所有二次曲线，包括直线等退化情况

非线性变换

之前说到，$\mathcal{X}$空间中的非线性分类问题，在找到一个合适的从$\mathcal{X}$到$\mathcal{Z}$上的非线性变换$\mathbf{\Phi}$以后，可以用$\mathcal{Z}$空间上的感知机来做分类。如何训练$\mathcal{Z}$上的感知机？参考之前使用数据$\{(\mathbf{x}_n, y_n)\}$训练$\mathcal{X}$上感知机的方法，用经过非线性变换得到的新数据$\{(\mathbf{z}_n = \mathbf{\Phi}(\mathbf{x}_n), y_n)\}$进行训练即可。整个步骤描述如下

> 1. 将原始数据$\{(\mathbf{x}_n, y_n)\}$使用$\mathbf{\Phi}$变换为$\{(\mathbf{z}_n = \mathbf{\Phi}(\mathbf{x}_n), y_n)\}$
> 2. 使用$\{(\mathbf{z}_n, y_n)\}$和你擅长的线性分类算法得到$\mathcal{A}$得到一个好的权重$\tilde{\mathbf{w}}$
> 3. 返回$g(\mathbf{x}) = \text{sign}(\tilde{\mathbf{w}}^\mathsf{T}\mathbf{\Phi}(\mathbf{x}))$

也可参考如下图例。注意使用非线性变换时一般没有$\mathbf{\Phi}^{-1}$的那一步，图中加上这一步只是为了加深理解（实际上，$\mathbf{\Phi}$的反函数是否存在也未可知）



非线性变换过程图解

非线性变换是一个独立的提取特征的操作，因此它不一定只能用在二元分类问题上，而是可以和所有其它线性模型相结合。需要注意的是，非线性变换只是得到新特征的一种方法，而特征工程是解决机器学习问题时最重要的“原力”之一

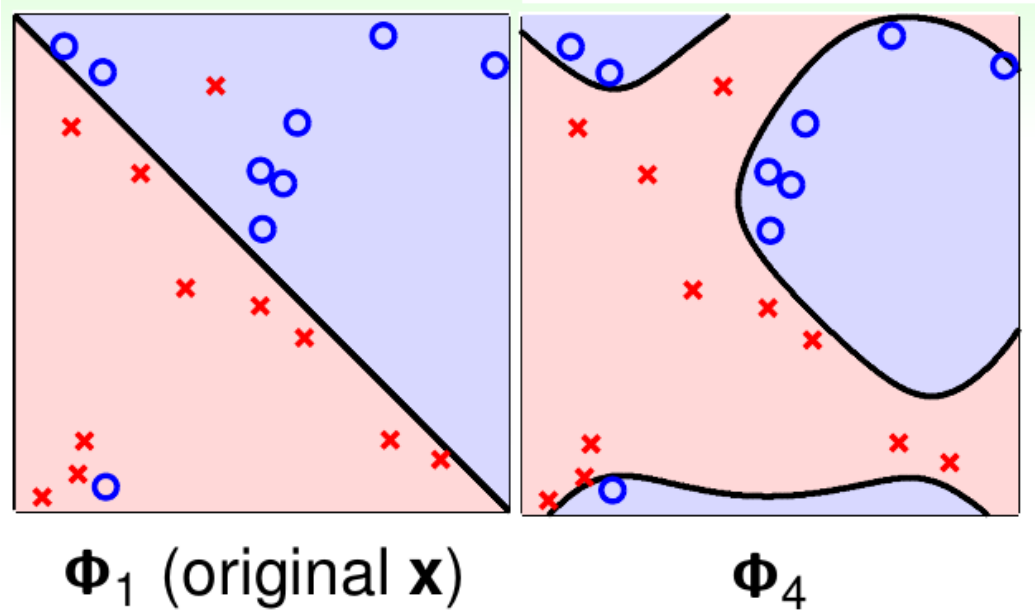
非线性变换的代价

计算/存储代价

假设原始的变量集合$\mathbf{x} \in \mathcal{X}$都是$d$维的，即$\mathbf{x} \in \mathbb{R}^d$，要做一个完全的二次变换，即其包含$(x_1, x_2, \dots, x_d)$形成的所有二次项、一次项和常数项，得到的$\mathbf{z} = \mathbf{\Phi}_2(\mathbf{x})$的维度是多少？所有二次项的数目是$\binom{d}{2} + d$（$x_i x_j, i \neq j$的项和$x_i^2$的项），一次项的数目是$d$，常数项数目是1，因此最后$\mathbf{z}$的维度是$\frac{d^2}{2} + \frac{3d}{2} + 1$。推广这个结论，假设源数据维度为$d$，做一个完全的$Q$次变换，得到的新数据维度$\tilde{d}$为$\binom{Q+d}{Q} = \binom{Q+d}{d}$，复杂度大概为$O(Q^d)$。这也是计算和存储$\mathbf{z} = \mathbf{\Phi}_Q(\mathbf{x})$和$\tilde{\mathbf{w}}$的代价

模型复杂度代价

另一方面，特征变换以后，新的权重$\tilde{\mathbf{w}}$有$\tilde{d} + 1$个自由变量。由前面提到的VC维和自由变量数之间的关系，$\mathcal{H}_{\mathbf{\Phi}_Q}$的VC维大概就是$\tilde{d} + 1$。这意味着当$Q$变大时，模型的VC维也会变大。这会带来什么问题？考虑下面这个带噪声的数据集，左边使用线性特征做分类，右边使用四次特征做分类



使用低维特征和高维特征对有噪声样本分类结果的比较

尽管$\mathbf{\Phi}_4$转换以后分类器可以做到E_{in}为0，但是*看上去*$\mathbf{\Phi}_1$更符合直觉。这又带来了一个均衡问题：低维的$\tilde{d}\,(Q)$可以使E_{out}与E_{in}足够接近，但是不能得到足够小的E_{in}；高维的$\tilde{d}\,(Q)$可以得到足够小的E_{in}，但是不能让E_{in}足够接近于E_{out}

那么如何选择这个合适的Q呢？上面这个例子里，可以通过用眼看来选择。但是用眼看总是一个很好的方法吗？先不说当$d = 10$时如何用眼偷看资料，仅考虑最开始举的那个例子

- 如果什么都不看，做一个$\mathbf{\Phi}_2$变换，VC维是6
- 如果看了数据，可以将$\mathbf{x}$转化为$\mathbf{z} = (1, x_1^2, x_2^2)$，这样VC维是3
- 甚至更聪明一点，可以做转化$\mathbf{z} = (1, x_1^2 + x_2^2)$，VC维是2

因此需要意识到一个问题：后来做的这些转换，其VC维降低是因为大脑做了分析而造成的功劳，因此判断VC维的时候得综合考虑，不能忘记人为分析造成的VC维减小

（讲义里最后还有句话：为了能安全地估计VC维，不能先“偷看”数据再决定做什么特征变换$\mathbf{\Phi}$）

结构化假设集

多项式变换可以递归定义：

$$
\begin{aligned}
\mathbf{\Phi}_0(\mathbf{x}) &= (1) \\
\mathbf{\Phi}_1(\mathbf{x}) &= (\mathbf{\Phi}_0(\mathbf{x}), x_1, x_2, \ldots, x_d) \\
\mathbf{\Phi}_2(\mathbf{x}) &= (\mathbf{\Phi}_1(\mathbf{x}), x_1^2, x_1x_2, \ldots, x_d^2) \\
\mathbf{\Phi}_3(\mathbf{x}) &= (\mathbf{\Phi}_2(\mathbf{x}), x_1^3, x_1^2x_2, \ldots, x_d^3) \\
&\cdots \\
\mathbf{\Phi}_Q(\mathbf{x}) &= (\mathbf{\Phi}_{Q-1}(\mathbf{x}), x_1^Q, x_1^{Q-1}x_2, \ldots, x_d^Q)
\end{aligned}
$$

如果记$\mathcal{H}_{\mathbf{\Phi}_i}$为$\mathcal{H}_i$，由上面的递归定义有

$$\mathcal{H}_0 \subset \mathcal{H}_1 \subset \mathcal{H}_2 \subset \mathcal{H}_3 \subset \ldots \subset \mathcal{H}_Q$$

由于越复杂的变换得到的假设函数能够打散的点越多，因此

$$d_{\text{VC}}(\mathcal{H}_0) \leq d_{\text{VC}}(\mathcal{H}_1) \leq d_{\text{VC}}(\mathcal{H}_2) \leq d_{\text{VC}}(\mathcal{H}_3) \leq \ldots$$

如果设$g_i = \arg\min_{h \in \mathcal{H}_i} E_{\text{in}}(h)$，则因为越复杂的变换可选的$h$越多，有

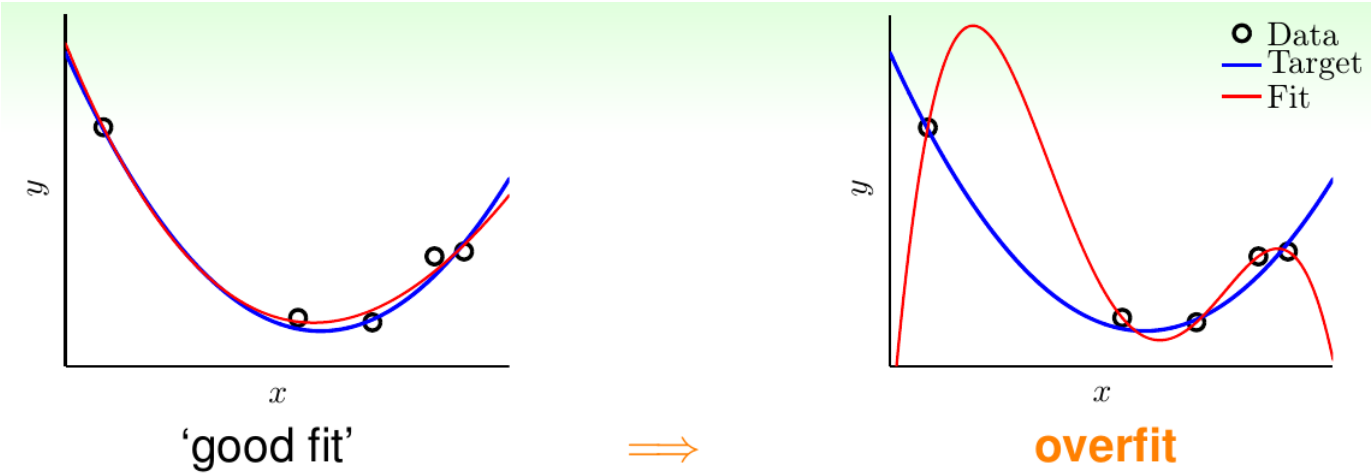
$$E_{\text{in}}(g_0) \geq E_{\text{in}}(g_1) \geq E_{\text{in}}(g_2) \geq E_{\text{in}}(g_3) \geq \ldots$$

所以是否有Q越高越好？且慢！第七讲中的图列出了VC维与误差之间的关系。尽管随着VC维的增大，样本内误差会单调减小，但是模型复杂度会增高。当模型的VC维超过最优的d_{VC}^*以后，样本外误差会不降反升。如果上来设计一个很高维度的转换，尽管能做到不错的E_{in}，但是遇到不同的样本模型性能会很差，而且很难做进一步改进。非线性变换（这里主要是多项式变换）的正确打开方式应该是，先从$\mathcal{H}_1$试起。如果成功了，那么由于其本身的性质，E_{out}应该也不会差。即便是效果不好，也可以再逐个尝试$\mathcal{H}_2$等等。这样做失去的只有用简单特征训练测试模型时的一点时间，得到的确实比较保险稳妥的结果——而且，大多数情况下，线性模型可能还真挺有效可用的。

NTUML 13. 过拟合的危害

什么是过拟合

假设现在要做一个一维数据集的回归分析，数据集一共有5个点，目标函数是一个二次函数，所有的$x_i, i = 1, 2, 3, 4, 5$都是随机产生，其对应的标签y_i是对应的目标函数值加上一点微小的噪声。由于拿到数据集以后并不知道目标函数的形式，因此一种思路是使用前面提到的多项式变换，引入一些四维特征。这样，拟合出的四维函数g可以完美地穿过这五个点，做到$E_{\text{in}}(g) = 0$。但是显而易见，这种情况下$E_{\text{out}}(g)$会非常大，导致了比较差的泛化结果。前面曾经提到过，当h的VC维特别大时，就会导致这种情况发生。只有当h的VC维取到最优情况d_{VC}^*时，才能使得E_{in}和E_{out}都取到最小。当h的VC维很大，超过d_{VC}^*时，随着VC维的升高，E_{in}仍然会降低，但是E_{out}会不降反增。称这种现象为**过拟合**。对应地，当h的VC维比较小，没有达到d_{VC}^*时，随着VC维的减小，E_{in}和E_{out}都会增加。称这种现象为**欠拟合**。下图给出了一个关于过拟合的状况示例



过拟合的例子。左图是使用复杂度恰到好处的模型拟合的结果，右边是使用过于复杂的模型拟合的结果

以上所讲的概念可以用开车的过程做类比——

- 过拟合，类似于出车祸

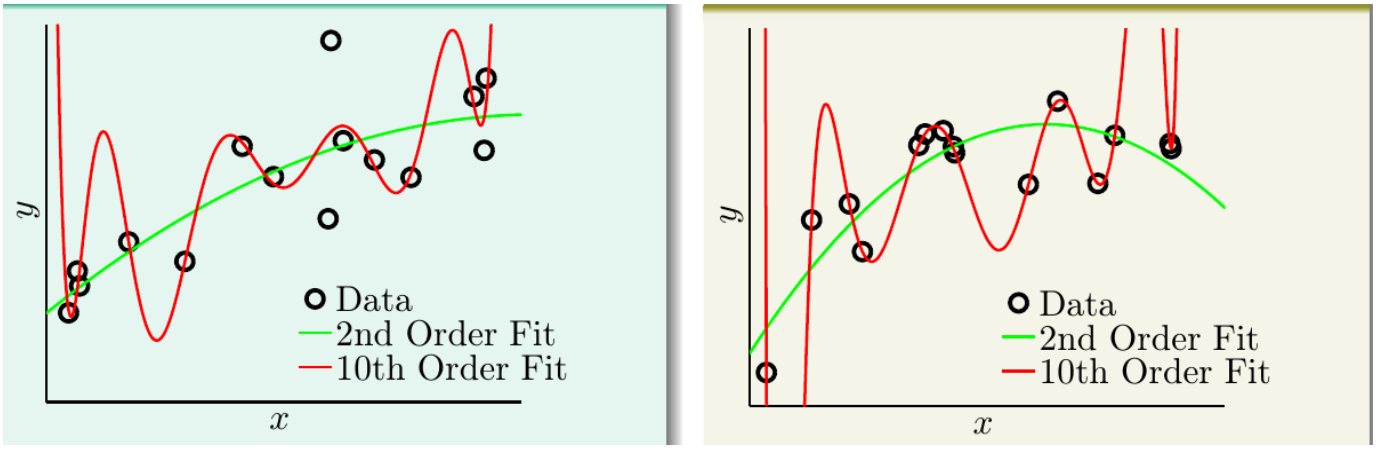
- 使用太复杂，d_{VC}太大的模型，类似于开太快
- 数据中噪声比较多，类似于路面情况不平
- 数据集大小N比较小，类似于司机对路面状况观察比较有限，对路况不熟

噪声与数据集大小的角色

接下来进行更多的实验来探索过拟合这一现象。我们设计了两个不同的目标函数和数据集，其分别为

- 目标函数是某个十次多项式，数据产生时加入一些噪声，产生15个数据点
- 目标函数是某个50次多项式，数据产生时不加入噪声，产生15个数据点

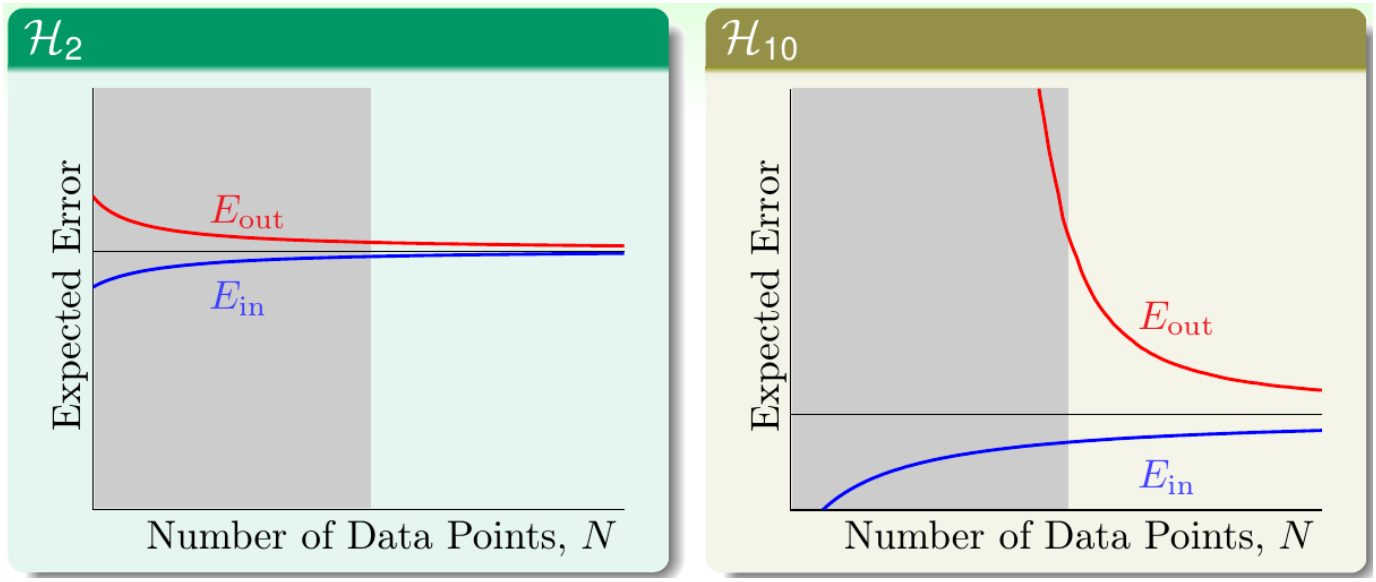
然后，对这两个数据集，都使用二次多项式和十次多项式拟合，观察这两个模型的效果。记二次多项式模型为$g_2 \in \mathcal{H}_2$，十次多项式模型为$g_{10} \in \mathcal{H}_{10}$。下图给出了两组方法在两组数据集上的效果比较。绿色为g_2，红色为g_{10}



二次多项式模型与十次多项式模型在两数据集上效果的比较。左边数据集由某个十次多项式产生，噪声适中；右边数据集由某个50次多项式产生，噪声非常小

对于左边的数据集，g_2的E_{in}为0.050，g_{10}的E_{in}为0.034，后者稍胜。但是前者的E_{out}为0.127，后者达到了9。对于右边的数据集，g_2的E_{in}为0.029，g_{10}的E_{in}为0.00001。而E_{out}的差距更多：前者0.120，后者居然达到了7680！可以看到，在这样两个由高次多项式产生的数据集上，使用高次多项式拟合出来的模型仍然发生了严重的过拟合现象。尤其是对于由十次多项式产生的数据集，按照道理，二次多项式模型应该没有能力在其上获得很好的效果，但是其E_{out}却的确取得了一个比较令人满意的结果。应该如何解释这种“以退为进”的现象？

这里回顾一下对不同的假设集合，其E_{in}/E_{out}与数据集大小N之间的关系，可见下图



如图所示，所有的二次多项式假设集$\mathcal{H}_2$和所有的十次多项式假设集$\mathcal{H}_{10}$都遵循了同样的规律：

- 随着数据量N的增长，E_{out}会下降，E_{in}会上升。最后当$N \to \infty$时，$|E_{\text{in}} - E_{\text{out}}| \to 0$
- E_{in}永远小于E_{out}。因为模型在学习数据时，总会学习到其所看见的数据中的噪声，并对其进行拟合。而对看不见的数据就难以对噪声拟合很好。由于假设数据集噪声的期望为0，因此当偏向一面时，就会远离另一面
- $\mathcal{H}_{10}$的E_{in}永远低于$\mathcal{H}_2$的

除此以外，当$N \to \infty$时，更复杂的$\mathcal{H}_{10}$的确会得到更好的E_{out}。因为模型VC维更大，能力更强。但是当N不够大时，$\mathcal{H}_{10}$的E_{out}会非常大。实际上，图中显示，当N比较小时（落在灰色区域），$E_{\text{out}}(\mathcal{H}_{10}) > E_{\text{out}}(\mathcal{H}_2)$，表现出了过拟合（“聪明反被聪明误”）。因此，**数据量不够大时，总应该使用更简单的模型！**

但是这里有一个问题：对于第二个数据集，明明数据集里没有噪声，为什么简单的模型表现也会好？实际上，模型复杂度提升以后，其效果可以看做是类似于往数据集里添加了噪声。这一情况将在下节继续探讨

确定性噪声

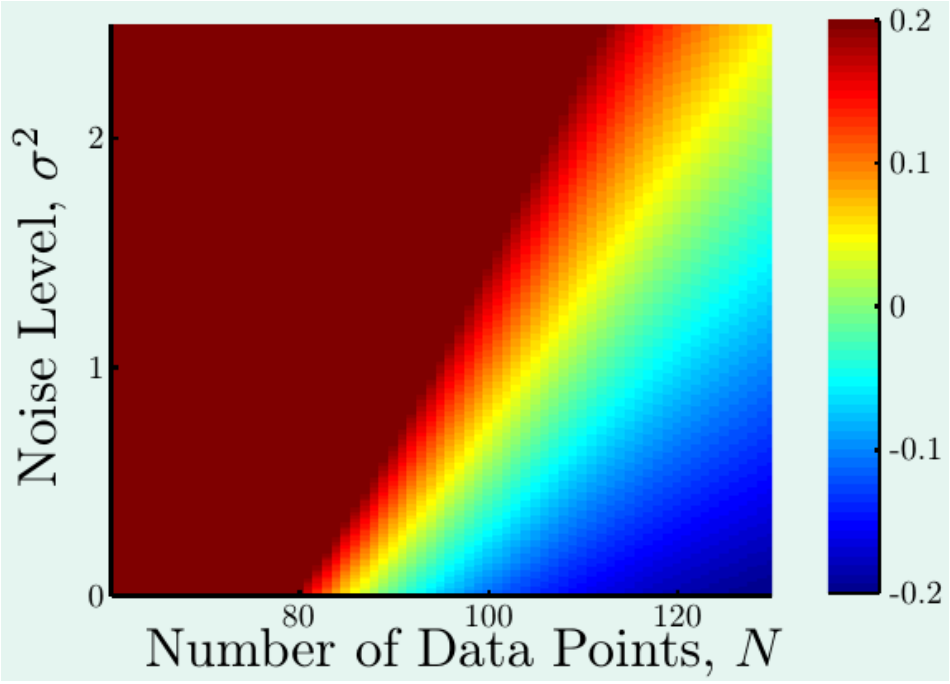
接下来看一个更细节的实验，来观察何时会发生过拟合。想象数据产生的过程为目标函数加上一些随机噪声，即$y = f(x) + \epsilon$。其中噪声的分布为方差为σ^2的高斯分布。因此有

$$y \sim \text{Gaussian}\left(\underbrace{\sum_{q=0}^{Q_t} \alpha_q x^q}_{f(x)}, \sigma^2\right)$$

这里，σ^2控制了噪声的强度，$f(x)$控制了模型的复杂度。记$f(x)$的复杂度为Q_f，其与多项式的最高次数挂钩：如果$f(x)$是10次多项式，$Q_f = 10$；如果$f(x)$是50次多项式，$Q_f = 50$。记实验使用的数据量为N，这里要探索不同的(N, σ^2)和不同的(N, Q_f)对过拟合的影响

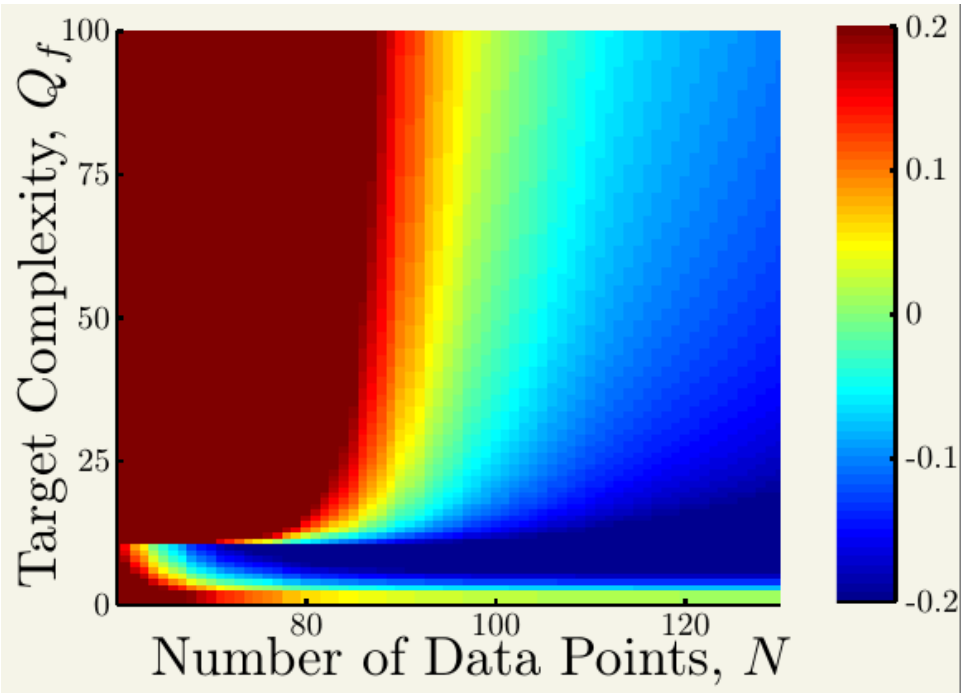
这里，候选的假设集合仍然和前面的设置一样，分别是$g_2 \in \mathcal{H}_2$和$g_{10} \in \mathcal{H}_{10}$。由前面的分析，$g_{10}$过拟合的概率比较大，那么这个过拟合的程度如何衡量？这里使用的方法是$E_{\text{out}}(g_{10}) - E_{\text{out}}(g_2)$

首先看固定模型复杂度$Q_f = 20$以后，过拟合度与N和σ^2的关系。这里颜色越深，过拟合程度越严重。显见当样本量小，误差大的时候，过拟合比较严重；而样本量大，误差小的时候，不太容易过拟合



模型复杂度固定时，样本量与噪声强度对过拟合的影响

接着看固定噪声强度$\sigma^2 = 1$以后，过拟合度与Q_f和N的关系。



噪声强度固定时，样本量与模型复杂度对过拟合的影响

大部分情况下，这种情况与前述实验类似，也是左上角容易过拟合，右下角不容易过拟合。这再次说明模型复杂度可以扮演噪声的角色。称前面施加的正态噪声为**随机噪声**，模型复杂度带来的噪声为**确定性噪声**，可以看出造成过拟合的因素有如下几条

- 数据量太小
- 随机噪声太大
- 确定性噪声太大
- 使用的模型比目标函数更复杂（注意第二幅图左下角）

过拟合常常存在，并很容易发生

可以从另一个角度去理解确定性噪声：当目标函数f过于复杂，$f \notin \mathcal{H}$时，f的一些性质不能被$\mathcal{H}$中的任意假设所描述（例如一个10次多项式，它的一些弯折是无法被二次多项式所描述的）。因此$\mathcal{H}$中最好的假设h^*和f之间的差距就是确定性噪声。实际上，确定性噪声与随机噪声差别不大（参考伪随机数生成器的原理），不过它并不是完全随机的，依赖于$\mathcal{H}$和给定的$\mathbf{x}$

应对过拟合

在知道了过拟合的概念和产生原因以后，就可以设计对过拟合的应对方案。仍以开车作比喻，有

- 先使用简单模型训练，类似于开慢车
- 对数据做一些清洗，类似于使用更准确的路况
- 观察数据，根据对数据的理解，按照相同规律产生一些新的数据（data hinting），类似于多看一些路况
- 使用正则化方法，类似于踩刹车
- 验证模型，类似于常看仪表盘

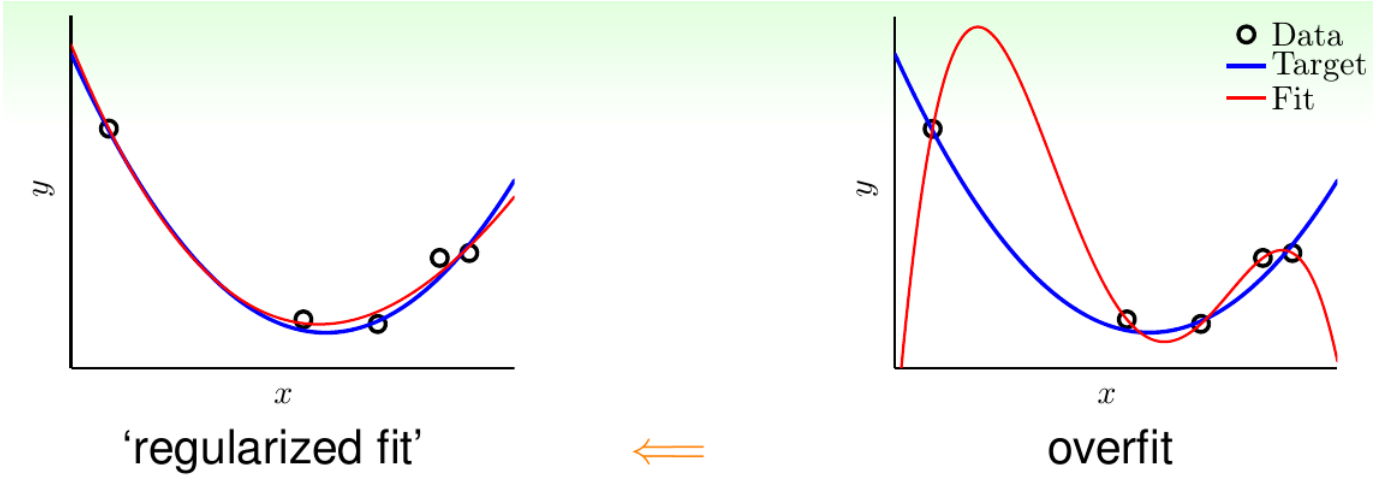
其中正则化和模型验证在之后讲解。数据清洗包括了两种方法，一种是矫正数据的标签，另一种是直接把脏数据移除。不过这种方法对模型的影响不定，可能非常有限

对于data hinting，一种很经典的例子就是在手写数字识别问题中，对已有的手写数字图像做翻转或者倾斜，产生一些新的例子（称作虚拟的例子）。这里要注意，这种方法产生的数据已经不是$\overset{\text{i.i.d.}}{\sim} P(\mathbf{x}, y)$，因此加进来的数据要有道理，而且不能离原有数据太远

NTUML 14. 正则化

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/09/10/NTUML-14-Regularization/

正则化的假设集



过拟合的正则化

如上图所示，右边给出的就是在上讲中经常提到的过拟合现象，可以看到拟合的曲线（红线）与目标函数图像（蓝线）差别特别大。**正则化**的目的是对复杂的模型进行调整，使其也可以逼近比较简单的模型。上图左边给出的就是正则化的效果示意，可以发现经过正则化以后，函数光滑了许多，也与目标函数接近了许多。其思想是，从高次多项式假设集合逐步“走回”低次多项式假设集合。“正则化”这个概念来自于早期函数逼近这个领域，以解决其中ill-posed问题

那么如何完成这个“走回”的操作？回想之前给出的对$x \in \mathbb{R}$做Q次多项式变换的方法$\mathbf{\Phi}_Q(x) = (1, x, x^2, \dots, x^Q)$，为了方便起见，这里不再将在转换后空间$\mathcal{Z}$中训练得到的权重记为$\tilde{\mathbf{w}}$，而是直接记为$\mathbf{w}$。由前面的介绍，$\mathcal{H}_{10}$中的$\mathbf{w}$的形式和$\mathcal{H}_2$中的$\mathbf{w}$的形式可以各自表示为：

$$\begin{aligned}\mathcal{H}_{10} &: w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_{10} x^{10} \\ \mathcal{H}_2 &: w_0 + w_1 x + w_2 x^2\end{aligned}$$

从这个角度看，$\mathcal{H}_2$可以看作是$\mathcal{H}_{10}$加上了一些限制$w_3 = \ldots = w_{10} = 0$而成。因此，前面的这个“走回”操作，其实就是加入一些限制。从最优化问题的角度来看，使用$\mathcal{H}_{10}$求解回归问题，其对应的最优化表达法为

$$\min_{\mathbf{w} \in \mathbb{R}^{10+1}} E_{\text{in}}(\mathbf{w})$$

而使用$\mathcal{H}_2$求解回归问题，从某种意义上讲，就是

$$\begin{aligned}\min_{\mathbf{w} \in \mathbb{R}^{10+1}} \quad & E_{\text{in}}(\mathbf{w}) \\ \text{s.t.} \quad & w_3 = w_4 = \ldots w_{10} = 0\end{aligned}$$

注意到$\mathcal{H}_2$是让$\mathcal{H}_{10}$的8个高次项系数为0。可以不这么教条，对这个限制条件做一点放松，使得其不是所有高次项全为0，而是有任意至少8个项系数为0（即最多有三个项系数不为0）。这样得到的新假设集合记做$\mathcal{H}_2'$，则其对应的最优化问题为

$$\begin{aligned}\min_{\mathbf{w} \in \mathbb{R}^{10+1}} \quad & E_{\text{in}}(\mathbf{w}) \\ \text{s.t.} \quad & \sum_{q=0}^{10} [\![w_q \neq 0]\!] \leq 3\end{aligned}$$

这三个假设集之间的关系是$\mathcal{H}_2 \subset \mathcal{H}_2' \subset \mathcal{H}_{10}$，因此新得到的这个假设集合表示能力强于$\mathcal{H}_2$，而且风险程度又弱于$\mathcal{H}_{10}$，看上去很不错。但是这里最关键的问题是，这个问题是一个NP难问题！

但是退而求其次，可以提出一个类似的最优化问题，即不再限制（至多）有多少个项不为0，而是限制这些系数的平方和小于某个常数。这种提法是有意义的，因为在原始问题里，如果某一项系数为0（或者很接近0），它对系数总体平方和的贡献也有限。因此，上面这个问题可以近似于（非等价于！）如下最优化问题

$$\begin{aligned}\min_{\mathbf{w} \in \mathbb{R}^{10+1}} \quad & E_{\text{in}}(\mathbf{w}) \\ \text{s.t.} \quad & \sum_{q=0}^{10} w_q^2 \leq C\end{aligned}$$

记该问题对应的假设集合为$\mathcal{H}(C)$，注意$\mathcal{H}(C)$和$\mathcal{H}_2'$有交叉，但是不完全相等。而且对不同的$C \geq 0$，$\mathcal{H}(C)$也有重叠结构：

$$\mathcal{H}(0) \subset \mathcal{H}(1.126) \subset \ldots \subset \mathcal{H}(1126) \subset \ldots \subset \mathcal{H}(\infty) = \mathcal{H}_{10}$$

称$\mathcal{H}(C)$这样的假设集合为正则化的假设集合，其最优解为正则化的假设，记为$\mathbf{w}_{\text{REG}}$

权重衰减的正则化方法

如何解这个加了限制的最优化问题？首先可以把问题转化为向量和矩阵形式。向量形式为

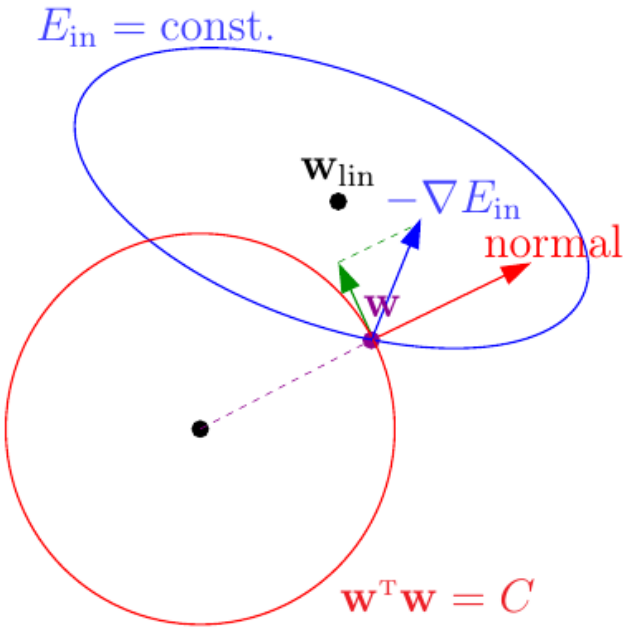
$$\min_{\mathbf{w}\in\mathbb{R}^{Q+1}} \ E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n - y_n)^2$$
$$\text{s.t.} \quad \sum_{q=0}^{Q} w_q^2 \le C$$

对应的矩阵形式为

$$\min_{\mathbf{w}\in\mathbb{R}^{Q+1}} \ E_{\text{in}}(\mathbf{w}) = \frac{1}{N}(\mathrm{Z}\mathbf{w}-\mathbf{y})^{\mathsf{T}}(\mathrm{Z}\mathbf{w}-\mathbf{y})$$
$$\text{s.t.} \quad \mathbf{w}^{\mathsf{T}}\mathbf{w} \le C$$

从几何意义上讲，这个限制条件是把$\mathbf{w}$限制在了一个球心为原点，半径为$\sqrt{C}$的超球体里

如何求解这个问题？如果没有条件限制，直接让$\mathbf{w}$沿着$-\nabla E_{\text{in}}(\mathbf{w})$的方向滚下去，滚到谷底就得到了$\mathbf{w}_{\text{LIN}}$



施加正则化以后权重w的优化

如上图，如果没有正则化项，蓝线就是（某一时刻）E_{in}的等高线，蓝色的箭头是$\mathbf{w}$变化的方向。加入正则化条件以后，$\mathbf{w}$只能在红圈里（或边上），因此大部分情况下$\mathbf{w}$应该都在红圈上。假设此时$\mathbf{w}$已经在红圈上，判断其是否达到最优解的方法就是看它是否还能沿着$-\nabla E_{\text{in}}$的方向往下滚。但是限制条件又规定$\mathbf{w}$不能滚出球（即沿着红色箭头的方向移动），只能沿着垂直于球法向量的方向移动（即绿色箭头的方向）。这样一来，如果$-\nabla E_{\text{in}}$这个方向可以沿着绿色箭头的方向分出一个分量，$\mathbf{w}$沿着该分量方向移动的话，就可以同时满足 i) 受条件$\mathbf{w}^{\mathsf{T}}\mathbf{w} = C$这个条件限制 ii) 减小$E_{\text{in}}$ 这两个要求。因此，最优的$\mathbf{w}_{\text{REG}}$肯定与$-\nabla E_{\text{in}}(\mathbf{w}_{\text{REG}})$方向平行（否则还可以继续往下滚），即

$$-\nabla E_{\text{in}}(\mathbf{w}_{\text{REG}}) \propto \mathbf{w}_{\text{REG}}$$

将这个比值设为$\frac{2\lambda}{N}$，则加入了正则化项以后，最优化问题实际上就是要求解$\mathbf{w}_{\text{REG}}$满足

$$\nabla E_{\text{in}}(\mathbf{w}_{\text{REG}}) - \frac{2\lambda}{N}\mathbf{w}_{\text{REG}} = 0$$

这里$\lambda > 0$实际上是拉格朗日乘子，上面实际上给出了拉格朗日乘子的几何意义。假设λ已知，上面这个式子其实就是关于$\mathbf{w}_{\text{REG}}$的一个线性方程。代入梯度的定义，有

$$\frac{2}{N}(\mathrm{Z}^{\mathsf{T}}\mathrm{Z}\mathbf{w}_{\text{REG}} - \mathrm{Z}^{\mathsf{T}}\mathbf{y}) + \frac{2\lambda}{N}\mathbf{w}_{\text{REG}} = 0$$

求解可得

$$\mathbf{w}_{\text{REG}} = (\mathrm{Z}^{\mathsf{T}}\mathrm{Z} + \lambda\mathrm{I})^{-1}\mathrm{Z}^{\mathsf{T}}\mathbf{y}$$

给定$\lambda > 0$，由于$\mathrm{Z}^{\mathrm{T}}\mathrm{Z}$是半正定矩阵，$\lambda I$是正定矩阵，两者相加也是正定矩阵，因此$\mathrm{Z}^{\mathrm{T}}\mathrm{Z} + \lambda\mathrm{I}$永远存在逆矩阵，上式总可求解

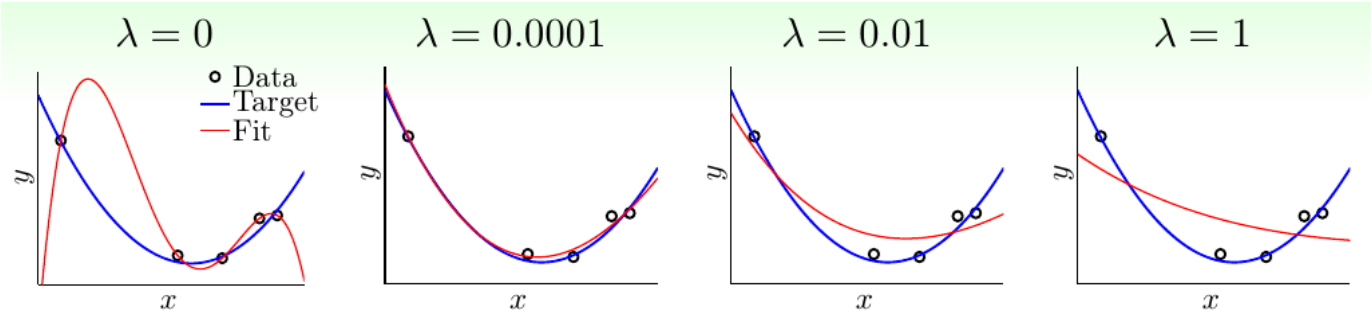
这个回归在统计上被称为是**岭回归**

岭回归还可以从另一个角度推导：求解$\nabla E_{\text{in}} = 0$对应的是要最小化$E_{\text{in}}(\mathbf{w})$，如果加上限制条件，那么对限制条件求积分，其实就是为了最小化$E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^{\mathrm{T}}\mathbf{w}$。其中加上的这一项称为**正则化项**，加上正则化项以后的误差称为**增强误差**，记为$E_{\text{aug}}(\mathbf{w})$。增强误差项包含了限制条件，又没有显式给出限制条件，比较容易求解。因此，正则化的过程也可以看作是最小化增强误差的过程，即对给定的$\lambda \geq 0$求解

$$\mathbf{w}_{\text{REG}} \leftarrow \arg\min_{\mathbf{w}} E_{\text{aug}}(\mathbf{w})$$

（笔记注：这块感觉讲得有点太细致了，如果知道拉格朗日乘子的意义，这种转化是显然的）

下图中给出了正则化项系数λ对模型效果的影响。可以看出，λ越大，正则得越狠，模型越容易欠拟合。实际使用时，一点点正则化项就可以起到很大的作用



正则化项系数lambda对模型效果的影响

进一步地，称$+\frac{\lambda}{N}\mathbf{w}^{\mathrm{T}}\mathbf{w}$为权重衰减的正则化方法。因为正则化项系数$\lambda$越大，$\mathbf{w}$就会更小，其范数$C$也会越小

这种方法可以和任意特征变换方式和线性模型搭配使用，例如跟Logistic回归搭配使用

第12页幻灯片给出了一个多项式特征变换的细节：如果按照普通的$1, x, x^2, \ldots$这样变换，当$x \in [-1, 1]$时，其高次项的值会特别小，因此对应的权重系数会比较大。为了避免这种现象出现，讲义中使用了$1, L_1(x), L_2(x), \ldots$的变换方式，其中$L_i(x)$对应为勒让德多项式中的第$i$个表达式。勒让德多项式的各个组成元素也是多项式空间的一组正交基

正则化与VC维

一个很自然的问题就是，正则化方法和之前介绍的VC维有什么关系。我们的出发点是要解一个受限制的E_{in}的问题，而由于λ与C有对应关系，因此解带有限制的最优化问题其实就等价于解一个无限制，但是相对E_{in}更大的E_{aug}。而原始问题对应的VC保证是，$E_{\text{out}}(\mathbf{w})$会比$E_{\text{in}}(\mathbf{w})$加上一个对复杂性的惩罚项$\Omega(\mathcal{H}(C))$，即$E_{\text{out}}(\mathbf{w}) \leq E_{\text{in}}(\mathbf{w}) + \Omega(\mathcal{H}(C))$。最小化$E_{\text{aug}}(\mathbf{w})$的过程也是在间接地满足这个VC保证（因为每个$\lambda$都有对应的$C$），而且由于去掉了限制条件，还考察了所有可能的$\mathbf{w}$

因此，被放大的误差与VC上界存在一定的相似性：被放大的误差是向E_{in}加上了一项，VC上界也是向E_{in}加上了一项。被放大的误差所加的一项$\frac{\lambda}{N}\mathbf{w}^{\mathrm{T}}\mathbf{w}$说明了某个假设有多复杂（这个假设越复杂，正则项就会越大，从直觉来看，曲线就越弯曲），而VC上界加的项$\Omega(\mathcal{H})$说的是整个假设集合有多复杂。如果将$\mathbf{w}^{\mathrm{T}}\mathbf{w}$记为$\Omega(\mathbf{w})$，将其看作是某个模型的复杂度，同时考虑一个前提：假设某个假设很复杂，那么它肯定处在一个复杂的假设集合中。这样一来，$\frac{\lambda}{N}\Omega(\mathbf{w})$就是$\Omega(\mathcal{H})$的一个缩影，而$E_{\text{aug}}$比起$E_{\text{in}}$也是$E_{\text{out}}$更好的代理。所以，最小化$E_{\text{aug}}$是最小化$E_{\text{out}}$的一个更好的启发式方法，这种做法给予我们更自由在$\mathcal{H}$上选择$\mathbf{w}$的能力

再从另一个角度去思考所得到模型的复杂性。对优化问题

$$\min_{\mathbf{w}\in\mathbb{R}^{\tilde{d}+1}} E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\Omega(\mathbf{w})$$

名义上模型的复杂度$d_{\text{VC}}(\mathcal{H}) = \tilde{d} + 1$，因为考虑了所有$\mathbf{w}$。但是实际上由于$\lambda$和$C$隐含的对应性，只有一少部分属于$\mathcal{H}(C)$的$\mathbf{w}$才会被考虑，因此模型的实际复杂度只有$d_{\text{VC}}(\mathcal{H}(C))$。这样，可以引入模型的“有效VC维”这个概念，记为$d_{\text{EFF}}(\mathcal{H}, \mathcal{A})$。其中$\mathcal{H}$说明候选的假设集仍然是完整的假设集，而$\mathcal{A}$则代表了算法选择模型的方法。所以即便$d_{\text{VC}}(\mathcal{H})$很大，如果用了正则化的算法$\mathcal{A}$，实际的$d_{\text{EFF}}(\mathcal{H}, \mathcal{A})$也会很小

常见的正则项

前面介绍的方法里，正则项用的都是$\mathbf{w}^{\mathrm{T}}\mathbf{w}$。那么正则项有没有可能有别的选择？$\mathbf{w}$有没有别的形式的条件？由于目标函数的不可预知性，我们无法提前给出最好的假设会是什么样。不过退而求其次，可以给出目标函数应该处于什么方向上。总体来讲，设计正则项有如

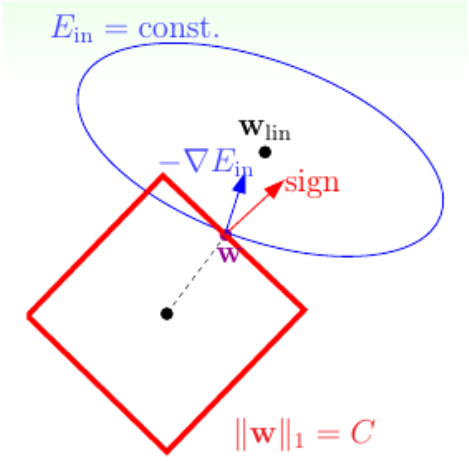
下指导性原则

- 正则项的设计应该与目标相关。例如假设要让正则项对称，可以设为$\sum[\![q \text{ is odd}]\!]w_q^2$
- 正则项的设计应该看上去合理。由于无论是随机误差还是决定性误差都不平滑，因此正则项应该使最后的$\mathbf{w}$更平滑或更简单
- 正则项的设计应该比较友好，易于优化

当然，即便正则项设计得不好，也可以用λ来控制，至少可以保证模型不受正则项影响

可以看到，正则项设计的三项原则和误差函数设计的三项原则异曲同工，都包括了目标相关性、合理性和友好性这三个点。它们也是设计机器学习算法时需要注意的关键

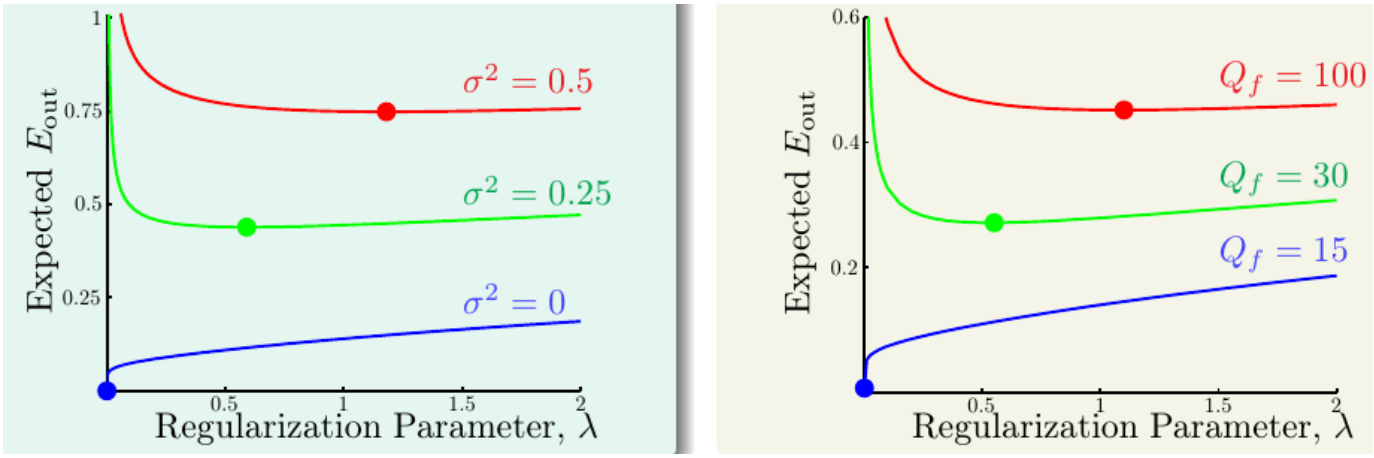
之前介绍过L2正则化项，其形式为$\Omega(\mathbf{w}) = \sum_{q=0}^{Q} w_q^2 = \|\mathbf{w}\|_2^2$。这是一个凸函数，而且处处可微，因此易于优化。另一种常见的正则化项称为L1正则化项：前面L2正则化项用的是$\mathbf{w}$的L2范数，同理L1正则化项用的是$\mathbf{w}$的L1范数，即$\Omega(\mathbf{w}) = \sum_{q=0}^{Q} |w_q| = \|\mathbf{w}\|_1$。注意这个函数仍然是凸函数，但是在$w_i = 0$时是不可微的，因此优化起来要稍微麻烦一点。那为什么它还广受青睐？参考下图



L1正则化示意图

类似之前提到的L2正则化的原理，只有当$\mathbf{w}$的移动方向$-\nabla E_{\text{in}}$（这里是蓝色箭头）与法向量（这里是红色箭头）平行时，$\mathbf{w}$才达到最优解。当$\mathbf{w}$在正方形的边上时，这种状态一般不会发生，因此总会有$\mathbf{w}$的一个分量拉着$\mathbf{w}$前进，直到到达正方形的顶点处。而在顶点时，说明有一些w_i其值为0，这意味着使用L1正则项会得到比较稀疏的解，计算效率更高

最后，下图给出了当噪声不同的时候，最优正则项系数的取值



噪声不同时，lambda与Eout期望值的关系。左为随机噪声，右为决定性噪声。最优lambda使用大实心点标出

可以看到，无论是随机噪声还是决定性噪声，噪声越大，λ就应该越大。但是我们并不知道噪声有多大，那么λ应该如何选择？这个问题留待下讲作答

NTUML 15. 验证

模型选择问题

即便是对二元分类问题，前面也介绍了很多种机器学习算法。这些算法（和算法所需要设置的参数）包括了

- 算法$\mathcal{A}$本身，例如PLA、口袋算法、线性回归、Logistic回归
- 算法迭代的步数T。是100，1000还是10000
- 算法的学习率η。是1，0.1还是0.01
- 特征转换$\mathbf{\Phi}$。是原始的特征，线性的特征，10次特征还是勒让德多项式10次特征
- 正则项$\Omega(\mathbf{w})$。是L2正则项，L1正则项还是其它正则项
- 正则项系数λ。是0，0.01还是1

既然面对了这么多选择，应该如何做决策就成了一个问题，这种问题称为模型选择问题。其可以进一步描述为，给定M个模型$\mathcal{H}_1,\mathcal{H}_2,\dots,\mathcal{H}_M$，每个模型对应的算法为$\mathcal{A}_1,\mathcal{A}_2,\dots,\mathcal{A}_M$，选出一个模型$\mathcal{H}_{m^*}$使得$g_{m^*}=\mathcal{A}_{m^*}(\mathcal{D})$有比较低的$E_{\text{out}}(g_{m^*})$。这个问题的难点在于，由于无法知道$P(\mathbf{x})$和$P(y|\mathbf{x})$，因此$E_{\text{out}}$是未知的；而选择又是不可避免的，因此这个问题是非常有实际意义的

通过视觉方法做选择已经在第12讲被否定了。那么是否可以根据各模型的E_{in}来判断？即

$$m^*=\mathop{\arg\min}_{1\le m\le M}(E_m=E_{\text{in}}(\mathcal{A}_m(\mathcal{D})))$$

并不可以。回顾之前的各种实验数据，如果采用了这样的设定，$\mathbf{\Phi}_{1126}$一定好于$\mathbf{\Phi}_1$，$\lambda=0$肯定比$\lambda=0.1$好，因此会非常容易过拟合。此外，假设$\mathcal{A}_1$是在$\mathcal{H}_1$上最小化了E_{in}，$\mathcal{A}_2$是在$\mathcal{H}_2$上最小化了E_{in}，那么最优的g_{m^*}就会在$\mathcal{H}_1\cup\mathcal{H}_2$上最小化$E_{\text{in}}$，模型选择和学习的过程VC维就是$d_{\text{VC}}(\mathcal{H}_1\cup\mathcal{H}_2)$。VC维变大，就会导致一般化问题

综上所述，使用E_{in}选择模型是非常危险的

或者可以根据各模型在测试集上的表现E_{test}来判断？即

$$m^*=\mathop{\arg\min}_{1\le m\le M}(E_m=E_{\text{test}}(\mathcal{A}_m(\mathcal{D})))$$

这样做可以避免一般化问题，因为根据有限的Hoeffding不等式，有

$$E_{\text{out}}(g_{m^*})\le E_{\text{test}}(g_{m^*})+O\left(\sqrt{\frac{\log M}{N_{\text{test}}}}\right)$$

但这么做的问题是，测试数据应该是找不到的，或者它应该在你老大那里！所以，这种方法是不可行的，至少是自欺欺人的

既然两种方法都不可行，有没有可能找到一份数据，它能同时满足以下三个要求：

- 与样本$\mathcal{D}$中数据独立同分布
- 方便获得
- 干净，没有被$\mathcal{A}_m$用来选择g_m（没有被候选算法偷看过）

答案是肯定的。具体做法是从原始数据中划出一部分数据作为**验证数据**（或者称为**验证集**），记为$\mathcal{D}_{\text{val}}$。然后，将算法用在$\mathcal{D}_{\text{val}}$上，根据它们在这些数据上的错误E_{val}来选择模型（有点像是合法地作弊）

验证

前面说到，可以从原始数据$\mathcal{D}$中取一部分数据作为验证数据$\mathcal{D}_{\text{val}}$，即有$\mathcal{D}_{\text{val}}\subset\mathcal{D}$，用它来模拟测试集。假设验证集大小为$K$，为了让验证误差$E_{\text{val}}$能够逼近$E_{\text{out}}$，这$K$个数据需要随机从$\mathcal{D}$中抽取。这样，原始大小为$N$的数据集被分解成了互不相交的两个集合：大小为$N-K$，专门用来做训练的数据集$\mathcal{D}_{\text{train}}$和大小为$K$，专门用来做验证的验证集$\mathcal{D}_{\text{val}}$。为了保证$\mathcal{D}_{\text{val}}$的纯洁性，只能把$\mathcal{D}_{\text{train}}$送进算法$\mathcal{A}_m$去训练模型和做模型选择。记通过这种方法得到的最优假设为g_m^-，类比前面给出的不等式，有

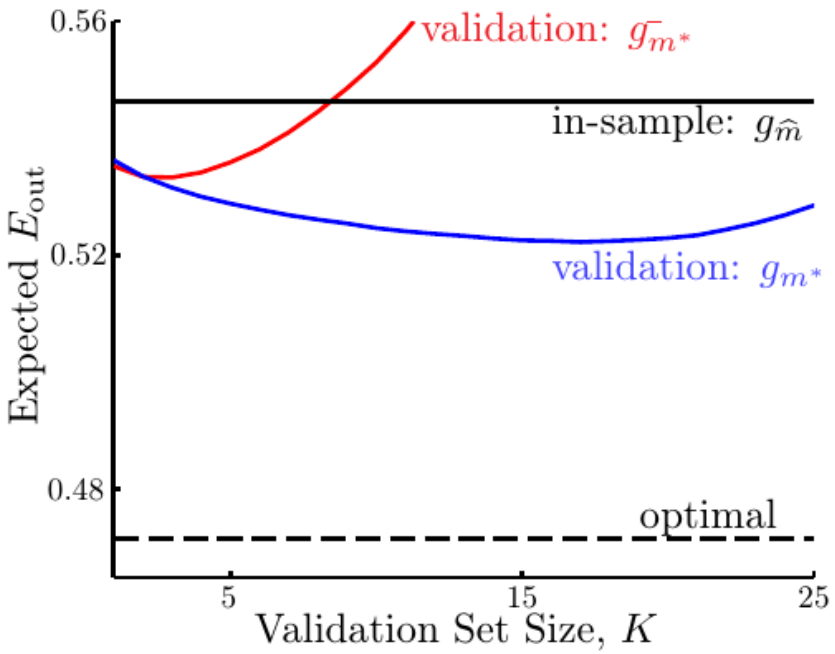
$$E_{\text{out}}(g_m^-)\le E_{\text{val}}(g_m^-)+O\left(\sqrt{\frac{\log M}{K}}\right)$$

因此，模型选择的方法是

$$m^*=\mathop{\arg\min}_{1\le m\le M}(E_m=E_{\text{val}}(\mathcal{A}_m(\mathcal{D}_{\text{train}})))$$

但是这里有一点需要注意：验证集的分出意味着与原来相比训练集变小。由之前的学习曲线可知，使用更少数据训练出来的模型，其E_{out}会大。因此，合理做法是记录m^*对应模型的超参数（就是前面提到的$\lambda,\mathbf{\Phi}$等等），然后用**整个**$\mathcal{D}$再把该模型训练一遍

可以使用验证的方法来在五次多项式特征变换假设集$\mathcal{H}_{\Phi_5}$和十次多项式特征变换假设集$\mathcal{H}_{\Phi_{10}}$之间做选择。得到的图像如下图所示



验证集大小与期望Eout之间的关系

这里四条线分别为：

- 黑实线，表示使用E_{in}来做选择，得到的最优模型的E_{out}。记该模型为$g_{\hat{m}}$
- 黑虚线（很靠下可能容易被忽略掉），表示使用E_{test}来做选择，得到的最优模型的E_{out}（实际上是作弊了）
- 红实线，表示使用$\mathcal{D}_{\text{train}}$训练，$\mathcal{D}_{\text{val}}$做模型选择得到的最优模型的$E_{\text{out}}$。记该模型为$g_{m^*}^-$
- 蓝实线，表示使用$\mathcal{D}_{\text{train}}$训练，$D_{\text{val}}$做模型选择，然后再把最优模型用$\mathcal{D}$训练一遍得到的模型的$E_{\text{out}}$。记该模型为$g_m^*$

可见蓝实线总是最低的，因此验证法真的在实际工程中可用

注意红线不如黑线的状况是因为K比较大，因此用来训练模型的数据量变小，所以模型效果就不好了。这也说明在选择验证集大小的时候其实有个两难境地：如果要使$E_{\text{out}}(g) \approx E_{\text{out}}(g^-)$，就需要$K$比较小才好，因为有足够多的数据训练模型；而要想$E_{\text{out}}(g^-) \approx E_{\text{val}}(g^-)$，就需要$K$比较大才好，因为这样才能足够模拟未知情况。实际中，常常设$K = N/5$

留一交叉验证

对于上面的验证过程，考虑一个极端情况：$K = 1$。这样，g^-和g会非常接近，但是$E_{\text{out}}(g^-) \approx E_{\text{val}}(g^-)$就不能满足了，只有一条数据能够用来验证模型。假设这条数据是第n条数据，则

$$\mathcal{D}_{\text{val}}^{(n)} = \{(\mathbf{x}_n, y_n)\}$$
$$E_{\text{val}}^{(n)}(g_n^-) = \text{err}(g_n^-(\mathbf{x}_n), y_n) = e_n$$

这一条e_n显然不能管中窥豹，很好地近似$E_{\text{out}}(g)$（事实上，如果要解决的是一个二元分类问题，使用一条数据得到的误差值只可能是0或者1，不可能是一个浮点数）。但是，如果验证的过程重复n遍，依次取出一条数据作为验证集，将它们的e_n记录下来最后做平均，得到的平均误差可能就是对E_{out}一个很好的近似。记这个过程为留一交叉验证，则该过程对E_{out}的近似为

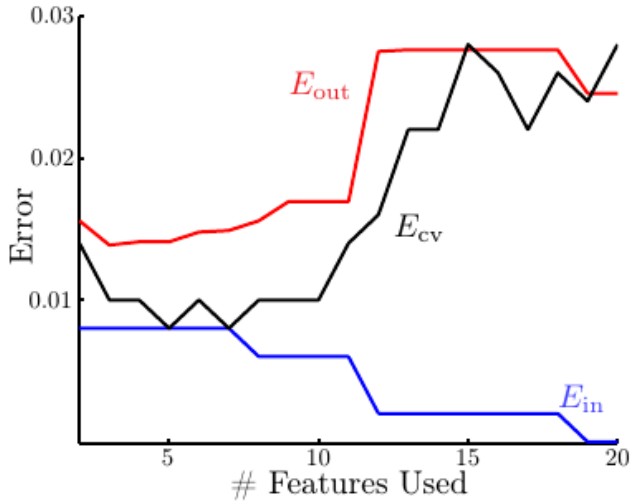
$$E_{\text{loocv}}(\mathcal{H}, \mathcal{A}) = \frac{1}{N}\sum_{n=1}^{N} e_n = \frac{1}{N}\sum_{n=1}^{N} \text{err}(g_n^-(\mathbf{x}_n), y_n)$$

接下来就是要证明，确实有$E_{\text{loocv}}(\mathcal{H}, \mathcal{A}) \approx E_{\text{out}}(g)$。设假设集$\mathcal{H}$和算法$\mathcal{A}$在任意数据集上都用过，所有这些数据集的期望留一交叉验证误差为$\mathcal{E}_{\mathcal{D}} E_{\text{loocv}}(\mathcal{H}, \mathcal{A})$，则

$$\begin{aligned}\mathcal{E}_{\mathcal{D}}E_{\text{loocv}}(\mathcal{H},\mathcal{A}) = \mathcal{E}_{\mathcal{D}}\frac{1}{N}\sum_{n=1}^{N}e_n &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}_{\mathcal{D}}e_n \\ &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}_{\mathcal{D}_n}\mathcal{E}_{(\mathbf{x}_n,y_n)}\text{err}(g_n^-(\mathbf{x}_n),y_n) \\ &= \frac{1}{N}\sum_{n=1}^{N}\mathcal{E}_{\mathcal{D}_n}E_{\text{out}}(g_n^-) \\ &= \frac{1}{N}\sum_{n=1}^{N}\overline{E_{\text{out}}}(N-1) = \overline{E_{\text{out}}}(N-1)\end{aligned}$$

即留一交叉验证误差的期望值与$E_{\text{out}}(g^-)$的期望值的确是近似的，可以看作是$E_{\text{out}}(g)$几乎无偏差的估计

下图给出了留一交叉验证的一个示例



留一交叉验证的示例。横轴为特征数目，纵轴为误差值

可以看出，交叉验证的误差曲线和E_{out}非常接近，即在实践中它的确是对E_{out}一个很好的估计。这张图再次证明，过多的特征会导致过拟合，特征数适量最好

V折交叉验证

留一交叉验证虽然可以很好地估计E_{out}，但是耗费的时间太多：每次使用$N-1$条数据训练，重复N次。因此除非有个别情况（例如有解析解的情况），否则留一交叉验证不太可行。此外，这种方法还有稳定性的问题。可以看到前一节的图中E_{cv}曲线有波动，而我们都希望选择的模型是比较可靠的，并不是因为某个实验的波动才导致其E_{cv}变小

思考留一交叉验证的过程可以发现，计算量变大的主要原因是该方法把数据集切得太碎，验证集太小造成了计算量的增加。因此一个改进的方法就是，只把原始数据集“斩成几大块”。这种方法叫做V折交叉验证：原始数据集$\mathcal{D}$被随机分成V份，仍然是每次使用$V-1$份数据训练模型，1份数据用来验证。这里交叉验证的误差为

$$E_{\text{cv}}(\mathcal{H},\mathcal{A}) = \frac{1}{V}\sum_{v=1}^{V}E_{\text{val}}^{(v)}(g_v^-)$$

选择时，仍然是

$$m^* = \underset{1\le m\le M}{\arg\min}(E_m = E_{\text{cv}}(\mathcal{H}_m,\mathcal{A}_m))$$

实践中，V通常设为10

总结

实践中，如果计算允许的话，一般还是使用V折交叉验证，而不使用普通的单次验证。而且5折交叉验证或者10折交叉验证效果就不错了，没必要做留一交叉验证

训练模型的过程是在整个假设集上做选择，有点像初赛；而验证的过程是对初赛中选出来的模型再做一次选择，有点像复赛。需要注意的是，只要有选择就会有误差，因此验证的结果还是会比测试结果更乐观。**测试结果才是最重要的，不要用验证结果自欺欺人**

NTUML 16. 三条锦囊妙计

- **本文作者：** Tingxun Shi

- **本文链接：** http://txshi-mt.com/2017/09/14/NTUML-16-Three-Learning-Principles/

奥卡姆剃刀

> Entia non sunt multiplicanda praeter necessitatem
>
> —-William of Occam (1287-1347)

该拉丁格言的意译为“如无必要，勿增实体”。用在机器学习的领域，意味着**对数据最简单的拟合往往也是最好的。**不过这里有两个问题：1. 说一个模型“简单”是什么意思，2. 我们怎么知道越简单就越好

简单的模型

之前其实提到过什么是“简单的假设”。如果一个假设h简单，通常它的复杂度$\Omega(h)$不高，需要定义的参数少。另外，也提到过什么是“简单的假设集”。如果一个假设集简单，它包含的有效假设不多，成长函数增长比较慢。这两个定义并不冲突：假设一个假设集$\mathcal{H}$只有2^ℓ个假设，那么属于它的每个假设h只需要ℓ个比特就能描述。因此，这两者之间的关系为，

$$\text{small } \Omega(h) \Leftarrow \text{small } \Omega(\mathcal{H})$$

因此，或者使用简单的模型，或者使用正则化得到简单的假设，这样就会对数据集有简单的解释

简单就好

除去数学证明，还可以通过一些富有哲学意味的推演来说明简单的就是好的。假设使用了一个简单的模型$\mathcal{H}$，那么它的成长函数$m_{\mathcal{H}}(N)$就会比较小，成长比较慢。这样，如果给模型随便给出N条数据（数据标签随机产生），那么这个模型很不太可能完美拟合数据（概率大概是$\frac{m_{\mathcal{H}}(N)}{2^N}$）。反过来，如果模型对数据的拟合非常好，则说明这个拟合的结果是有显著性的，数据应该有迹可循。而这个推论对复杂的模型是不能成立的，即对于复杂数据，即是其可以对数据表现良好，也不敢说这个结果是有显著性的

因此，在实操时，首先就应该使用线性模型，而且要时时思考有没有过度使用模型的现象

取样偏差

先来看个故事。1948年美国总统候选，民主党的候选人是杜鲁门，共和党则是杜威。投票结束后，某报纸进行了一次电话民调，抽样估计谁会赢。抽样结束以后，写下了《杜威击败杜鲁门》的报道

然而，实际上，获胜的是杜鲁门！造成这个反转的原因，不是因为编辑弄错，也不是运气不好，而是因为电话很贵，所以抽样到的都是有钱人，当年正好有钱人是杜威的票仓，因此这样做很容易得出杜威获胜的结论。这告诉我们

> 如果数据抽样是有偏的，学习得到的结果也是有偏的

回想当初推导VC上界时，少数几条限制条件之一就是训练集和测试集数据应该来自同一个分布。假设训练集数据来自$P_1(\mathbf{x}, y)$而测试集数据来自于$P_2 \neq P_1$，VC理论就不成立了。就像问已知一个学生刻苦学习了数学，问他在英语考试中会取得什么样的成绩一样——谁知道呢

另一个例子来源于林轩田老师本人的经历：在做Netflix的电影推荐系统时，主办方称如果准确率能提升10%就有100万奖金可以拿，而林老师提的第一个模型在验证集上就提升了13%的准确率。这里有什么问题？问题在于他构建验证集是使用传统的**随机**取样法，而实际比赛要求的是根据某个人前几次的观影记录来推测其对**后几部**电影的评分，即测试集和验证集的分布还是不同的。这个隐蔽的错误再次证明，**要了解测试环境，进而让训练环境与测试环境尽量接近。**对于刚才的例子，训练时就可以给时间轴上靠后的数据权重更大，并使用后面的数据做验证集

之前介绍的信用卡批准这一案例实际上也有同样的问题：我们拿到的信用卡使用数据其实都是来自于银行已经核发了信用卡的顾客，即顾客群体已经被做了一部分筛查。如果这些数据不进行调整就用来训练模型，可能会造成问题，因为它面向的顾客群体与之前相比多了一些会被原系统毙掉的顾客，而模型对这些顾客的表现很可能与预期不符合

数据窥探

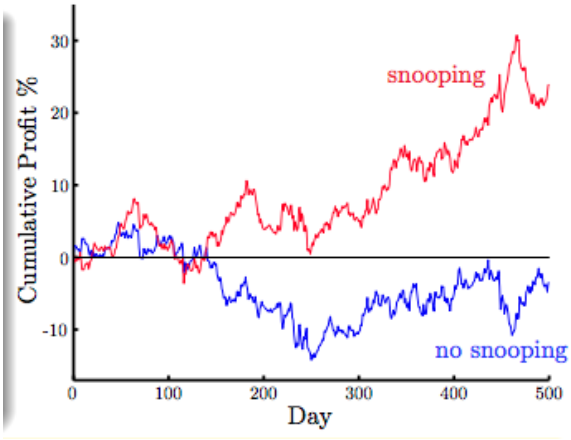
前面曾经说过，如果肉眼偷看了数据，尽管设计出来的模型看似VC维很低，但是实际上如果考虑到人脑“附加的模型复杂度”，VC维仍然会很高。因此，在做多项式变换Φ时，不应该提前窥探数据

然而数据窥探的概念其实远比前面所说的过程要宽泛。严格来讲，需要记住

> 无论数据集影响了学习过程中的哪一步，学习得到的模型效果都要被打个问号

假设拿到了某种货币8年的汇率情况，希望对汇率的走势进行预测（假设是用过去20天的汇率情况预测第21天的汇率情况）。一种自然的对数据集的划分是把前6年的数据作为训练集，后2年的数据作为测试集。可以看出来，这是一个回归问题

处理该问题时，为了避免原始数据取值范围太大对模型造成影响，通常要对数据做一个缩放（scale）。正常的情况下，应该是将训练数据和测试数据分别作缩放操作。但是如果将这两组数据放在一起缩放，缩放缩放完以后再划分，会是什么结果？



数据窥探对模型造成的影响

可以看出，尽管后两年的数据没有直接参与训练，但是仍然对模型效果产生了影响。因为在缩放的时候，训练数据已经隐含地知道了测试数据的一些统计信息（例如最大值、最小值、均值、标准差等等）

学术研究中也有类似数据窥探的现象。一般研究时，都会有一个标准数据集$\mathcal{D}$。假设研究机构A提出了一个模型$\mathcal{H}_1$在$\mathcal{D}$上表现良好，那么就会发表文章。后人如果做出了改进，提出的$\mathcal{H}_2$比$\mathcal{H}_1$更好，也会发表文章。假设某个人做出了m个模型，实际上该模型的一般化能力需要被怀疑一下，因为此时该模型的VC维已经是$d_{\text{VC}}(\bigcup_m \mathcal{H}_m)$。因此越晚发表的论文其实受数据窥探的影响越严重

> “如果在一个犯人身上拷问越久，ta就越容易招供”

然而，完全避免数据窥探是非常难的，只能尽力做好以下几点

- 不到必须的时候，不使用测试集。或者小心地使用验证集
- 不在看过数据以后再决定用什么特征
- 时刻保持一颗怀疑之心

三生三事

本课到此为止（也就是机器学习基石课）讲的东西基本都跟“三”有关，最后来一起梳理一下

- 三项跟机器学习有关的概念：数据挖掘、人工智能和统计学
- 三条理论保证：Hoeffding, Multi-Bin Hoeffding, VC
- 三种线性模型：PLA/pocket（使用0/1误差）、线性回归（使用平方误差）和Logistic回归（使用交叉熵误差）
- 三种重要的工具：特征变换、正则化、验证
- 三条锦囊妙计：奥卡姆剃刀、抽样偏差、数据窥测
- 三条未来的发展方向（机器学习技法课的内容）：更强力的变换方法、更丰富的正则化方法、更少标签时的处理方式

NTUML 17. 线性支持向量机

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/09/18/NTUML-17-Linear-SVM/

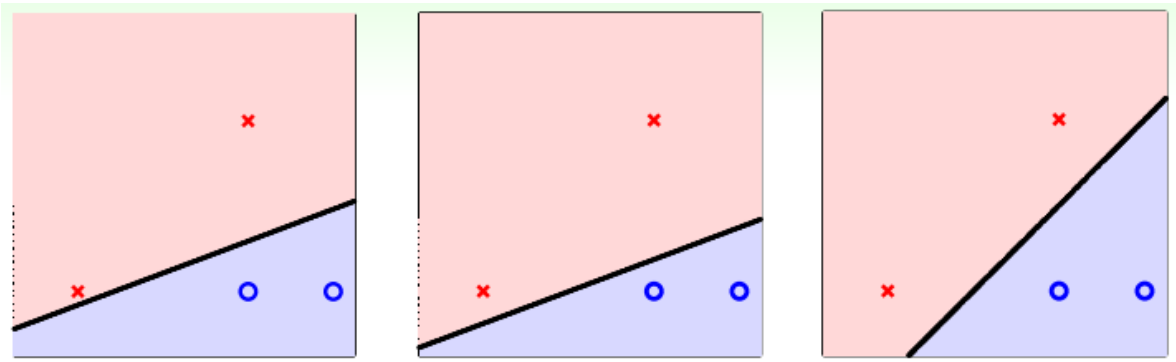
最大间隔分离超平面

前面在介绍感知机时，曾经提到线性分类器的表现形式为

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x})$$

它在二维空间表现为一条直线，在三维空间表现为一个平面，在n维空间其维度总为$n-1$，称作**超平面**

感知机算法对一个线性可分的数据集，总能找到一个超平面将其分开。然而，绝大多数情况下，这个超平面都有很多种可能性。例如下图所示



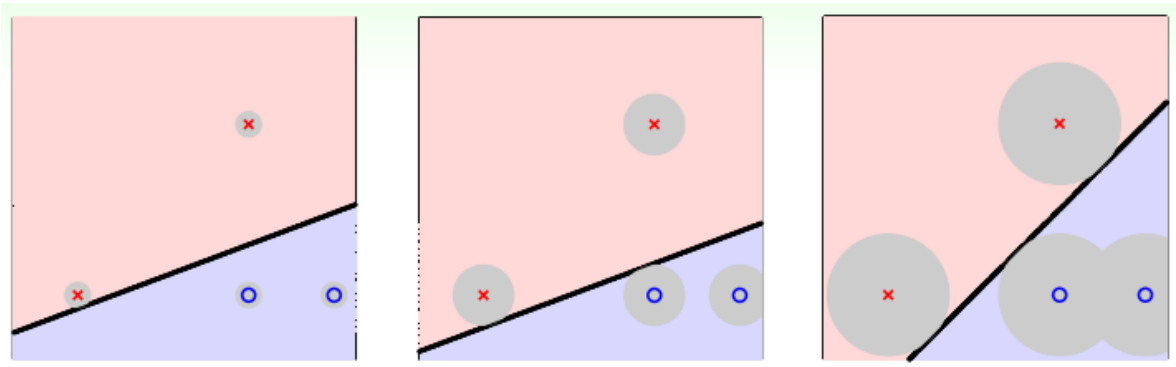
对同样的数据集，通常可以学习出多个超平面将其分开

对图中的四个点，至少有三条直线可以将它们分开。PLA算法的选择是随机的，所以它们都有可能被选到。而且就VC维来看，这三条线也没什么差别。因为有

$$E_{\text{out}}(\mathbf{w}) \leq E_{\text{in}}(\mathbf{w}) + \Omega(\mathcal{H})$$

对这三条直线，都有$E_{\text{in}}(\mathbf{w}) = 0$。而感知机的假设集合前面已经证明过其VC维为$d_{\text{vc}} = d + 1$。所以它们的$E_{\text{out}}$上限是相等的

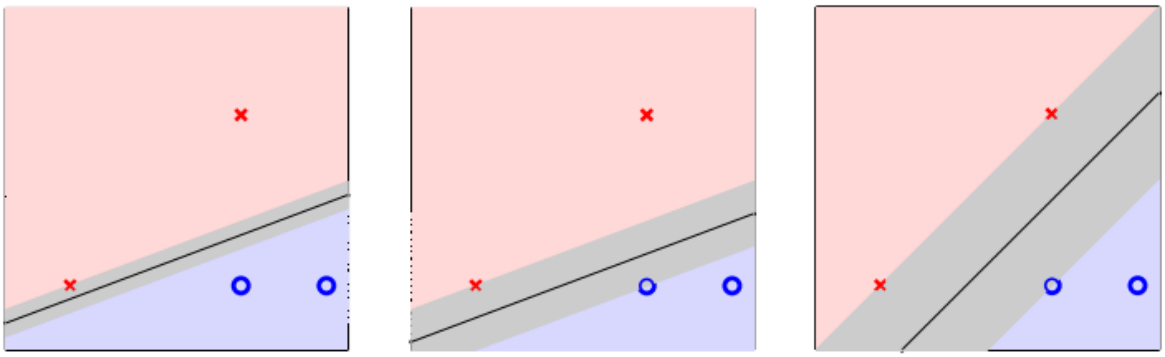
不过，对大部分人来说，可能都会选择第三幅小图中的那一个分类器。从直觉来说，假设原始数据为$\mathbf{x}_n$，测试数据为$\mathbf{x}$，由于误差的存在，有$\mathbf{x} \approx \mathbf{x}_n$。学习的目标当然是希望对于学习出来的分类器能将这两条数据归为一类。对于图中所给出的第三个分类器，因为它离所有四个数据点都很远，因此即便这个误差比较大，它也能做出正确的分类。对每条数据，以其为圆心，用灰色标记出分类器可允许的最大误差范围（即测试点落在灰圈之内就可以得到正确的标记），三条直线各自的允许误差如下图所示



分类器所允许的最大误差范围，以灰圈表示

也就是说，左边分类器与最右边分类器的最大区别，就是对误差的容忍度有不同。数据$\mathbf{x}_n$离分类器越远，模型对误差的容忍度越高。而由前面的讲述，测量误差是噪声的一种，因此模型对误差的容忍度越高，意味着模型对噪声的容忍度越高。又因为噪声的存在是导致过拟合现象的一大诱因，所以模型对噪声的容忍度越高，其本身鲁棒性越强，越不容易过拟合。上面的推理过程反过来也成立，所以最右分类器最优的原因是它最鲁棒，因为它离与它最近的数据最远

前面的讲述是从“点到直线距离”这个角度出发。可以换个角度，再看看从“直线到点距离”这个角度出发有什么结论。如果直线到点的距离越大，那么将直线像两边样本方向移动（直到碰到最接近的正/负例样本），其可移动的距离就越大，可以说这条直线越“胖”。三条直线各自的“胖度”如下图所示



前述三个分类器的“胖度”

因此，分类器“越胖”，它与离它最近的数据点$\mathbf{x}_n$距离越大，鲁棒性越好。我们的目标也就变成了要寻找最胖的分离超平面。假设数据集是线性可分的，那么要解决的最优化问题可以使用如下不严格的语言描述

$$\begin{aligned} \max_{\mathbf{w}} \quad & \text{fatness}(\mathbf{w}) \\ \text{subject to} \quad & \mathbf{w} \text{ classifies every } (\mathbf{x}_n, y_n) \text{ correctly} \\ & \text{fatness}(\mathbf{w}) = \min_{n=1,\dots,N} \text{distance}(\mathbf{x}_n, \mathbf{w}) \end{aligned}$$

文献中，这个“胖度”（fatness）通常称为**间隔**（margin）（有点像排版中“留白”的意味）。另外，“分类正确”意味着$\mathbf{w}^{\mathsf{T}}\mathbf{x}_n$与$y_n$同号，也就是对每个样本$n$有$y_n\mathbf{w}^{\mathsf{T}}\mathbf{x}_n > 0$。因此上面的最优化问题可以写成

$$\begin{aligned} \max_{\mathbf{w}} \quad & \text{margin}(\mathbf{w}) \\ \text{subject to} \quad & \text{every } y_n\mathbf{w}^{\mathsf{T}}\mathbf{x}_n > 0 \\ & \text{margin}(\mathbf{w}) = \min_{n=1,\dots,N} \text{distance}(\mathbf{x}_n, \mathbf{w}) \end{aligned}$$

问题的目标就变成了：**要找到最大间隔分离超平面**

标准的最大间隔问题

由上一节最优化问题的定义，可以看到里面有一项是要计算点$\mathbf{x}_n$与$\mathbf{w}$之间的距离。所以首先要解决的问题是，这个距离应该如何计算。注意这里算距离时截距项的系数w_0与其它特征项的系数$w_1, \dots, w_d$是分开计算的，即

$$b = w_0$$

$$\begin{bmatrix} | \\ \mathbf{w} \\ | \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}; \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

本讲中的假设函数形式也变为了

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{x} + b)$$

因此要求解的问题可以定义为

> 给定超平面$\mathbf{w}^{\mathsf{T}}\mathbf{x}' + b = 0$，求距离$\text{distance}(\mathbf{x}, b, \mathbf{w})$

假设$\mathbf{x}', \mathbf{x}''$是超平面上的两个点，由上面的式子，显然有$\mathbf{w}^{\mathsf{T}}\mathbf{x}' = -b, \mathbf{w}^{\mathsf{T}}\mathbf{x}'' = -b$。两者相减，有$\mathbf{w}^{\mathsf{T}}(\mathbf{x}'' - \mathbf{x}') = 0$。而$\mathbf{x}'' - \mathbf{x}'$是超平面上的一个向量，$\mathbf{w}$与这个向量内积为0，说明$\mathbf{w}$与这个超平面正交。因此空间内任意一点$\mathbf{x}$到该超平面的距离就是$\mathbf{x} - \mathbf{x}'$往$\mathbf{w}$方向的投影，即

$$\text{distance}(\mathbf{x}, b, \mathbf{w}) = \left| \frac{\mathbf{w}^{\mathsf{T}}}{\|\mathbf{w}\|}(\mathbf{x} - \mathbf{x}') \right| = \frac{1}{\|\mathbf{w}\|}|\mathbf{w}^{\mathsf{T}}\mathbf{x} + b|$$

又因为现在研究的超平面是分离超平面，前面提到过，在数据集线性可分的情况下，分离超平面对数据集中每个点n都有$y_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) > 0$且$y_n = \pm 1$。利用这个性质，距离的计算公式又可以写为

$$\text{distance}(\mathbf{x_n}, b, \mathbf{w}) = \frac{1}{\|\mathbf{w}\|} y_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b)$$

又考虑到平面上同一条线有不同的表示方式，$\mathbf{w}^{\mathsf{T}}\mathbf{x} + b = 0$和$3\mathbf{w}^{\mathsf{T}}\mathbf{x} + 3b = 0$，因此系数可以放缩。如果将其放缩到满足$\min_{n=1,\dots,N} y_n(\mathbf{w}^{\mathsf{T}}\mathbf{x}_n + b) = 1$，代入到最开始要求解的最优化问题中的限制条件，就有$\text{margin}(b, \mathbf{w}) = \frac{1}{|\mathbf{w}|}$。因此该最优化问

题就可以写为

$$\begin{aligned} \max_{\mathbf{w},b} \quad & \frac{1}{\|\mathbf{w}\|} \\ \text{subject to} \quad & \text{every } y_n \mathbf{w}^\mathsf{T} \mathbf{x}_n > 0 \\ & \min_{n=1,\dots,N} y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) = 1 \end{aligned}$$

而且再仔细分析，两个限制条件中第二个条件比第一个严格，所以可以继续化简

$$\begin{aligned} \max_{\mathbf{w},b} \quad & \frac{1}{\|\mathbf{w}\|} \\ \text{subject to} \quad & \min_{n=1,\dots,N} y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) = 1 \end{aligned}$$

这个问题是求最大值，而限制条件里又有最小值，还是不太好解。但是限制条件可以进一步放宽，只需要满足$y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1$就可以。这里不用担心出现所有点都$y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) > 1$的情况，因为如果这种情况发生，假设有$y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq C, C > 1$，那么对$(b, \mathbf{w})$缩放，得到$(b/C, \mathbf{w}/C)$，则还可以满足$y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1$，但是因为$\mathbf{w}$被除了一个大于1的正常数，因此$\frac{1}{|w|}$会变大，新的解比$(b, \mathbf{w})$取得的最优解更好，造成矛盾。最后，再做一点简单的变换，可以得到最终要优化的问题为

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \\ \text{subject to} \quad & y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \geq 1 \text{ for all } n \end{aligned}$$

支持向量机（SVM）

上面得到的这个问题通常称为**标准问题**，解这个问题返回的g通常称为**支持向量机**（Support Vector Machine, SVM）。SVM的来源是这样的：通过分析第一节中的例子，可以发现最大间隔分离超平面完全由离该超平面最近的点定义，其它点如果从数据集中删除掉，对最优超平面的选择没有任何影响。这些决定了最大间隔分离超平面的点通常被称为**支持向量**，因此SVM的意思就是使用支持向量学习出最优的超平面

SVM并不是一个很容易手工解出的问题。由于限制条件的存在，使用梯度下降也不太方便。不过这个问题要最小化的是一个$\mathbf{w}$的二次函数，限制条件又都是$\mathbf{w}$和b的一次式，因此这个问题其实是凸优化问题里的二次规划问题（简称QP问题）。接下来的问题就是，如何把这个最优化问题改写成二次规划问题的标准形式。假设变量为$\mathbf{u}$，其二次项系数为Q，一次项系数为$\mathbf{p}^\mathsf{T}$，限制条件的每一项系数为$\mathbf{a}_m^\mathsf{T}$，常数为c_m，则二次规划问题的标准形式就可以写为

$$\begin{aligned} & \text{optimal } \mathbf{u} \leftarrow \mathrm{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c}) \\ \min_{\mathbf{u}} \quad & \frac{1}{2}\mathbf{u}^\mathsf{T}\mathrm{Q}\mathbf{u} + \mathbf{p}^\mathsf{T}\mathbf{u} \\ \text{subject to} \quad & \mathbf{a}_m^\mathsf{T}\mathbf{u} \geq c_m, \text{for } m = 1, 2, \dots, M \end{aligned}$$

记$\mathbf{w}$的维数为d，则对应关系如下

$$\begin{aligned} \mathbf{u} &= \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix} \\ \mathrm{Q} &= \begin{bmatrix} 0 & \mathbf{0}_d^\mathsf{T} \\ \mathbf{0}_d & \mathrm{I}_d \end{bmatrix} \\ \mathbf{p} &= \mathbf{0}_{d+1} \\ \mathbf{a}_n^\mathsf{T} &= y_n \begin{bmatrix} 1 & \mathbf{x}_n^\mathsf{T} \end{bmatrix} \\ c_n &= 1 \\ M &= N \end{aligned}$$

因此可以得到线性硬间隔支持向量机算法为

> 1. 设置$\mathrm{Q} = \begin{bmatrix} 0 & \mathbf{0}_d^\mathsf{T} \\ \mathbf{0}_d & \mathrm{I}_d \end{bmatrix}; \mathbf{p} = \mathbf{0}_{d+1}; \mathbf{a}_n^\mathsf{T} = y_n \begin{bmatrix} 1 & \mathbf{x}_n^\mathsf{T} \end{bmatrix}; c_n = 1$
> 2. $\begin{bmatrix} b \\ \mathbf{w} \end{bmatrix} \leftarrow \mathrm{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$
> 3. 将b和$\mathbf{w}$返回为g_{SVM}

这里“硬间隔”意味着没有点会在“胖的边界”里或者甚至越过分离超平面，而“线性”意味着用的是原始的$\mathbf{x}$。如果要做非线性的SVM，只需要做非线性转换$\mathbf{z}_n = \mathbf{\Phi}(\mathbf{x}_n)$就可以

最大间隔分离超平面背后的原理

最后，来看一下SVM模型可以做得好的理论解释：SVM可以跟正则化做连接：正则化最小化的是E_{in}，但是限制了$\mathbf{w}^{\mathsf{T}}\mathbf{w} \leq C$，而SVM最小化的是$\mathbf{w}^{\mathsf{T}}\mathbf{w}$，限制条件是$E_{\text{in}} = 0$（当然还有其它条件，但是这里没有包括进去）。尽管两者的最小化目标和限制条件有所交换，但是可以认为两者做的事情是相同的，实际上就是想把两件事都考虑进去，即可以把SVM看作是一种正则化

另一种解释是从VC的角度看。假设现在使用一种“大间隔算法”$\mathcal{A}_\rho$，如果存在某个g有$\text{margin}(g) \geq \rho$，则这个算法返回$g$，否则返回空。$\mathcal{A}_0$（实际上就是PLA）可以打散三个元素，但是如果$\rho$比较大，能被$\mathcal{A}_0$打散的数据就不一定能被$\mathcal{A}_\rho$打散。即SVM的VC维更小，一般化能力更强

可以从概念上推导$\mathcal{A}_\rho$的VC维，注意此时VC维依赖于数据。这里假设$\mathcal{X}$是$\mathbb{R}^2$上的单位圆，如果$\rho = 0$，那算法退化成感知机，$d_{\text{VC}} = 3$。但是当$\rho > \frac{\sqrt{3}}{2}$时，算法不能打散任何3个输入，$d_{\text{VC}} < 3$。通常来讲，假设$\mathcal{X}$在一个半径为$R$的超球体里，有

$$d_{\text{VC}}(\mathcal{A}_\rho) \leq \min\left(\frac{R^2}{\rho^2}, d\right) + 1 \leq d + 1$$

之前课程讲述过超平面和特征变换这两种工具。超平面的假设集不大，但是边界比较简单；而带了特征变换的超平面的假设集变得非常大，不过边界也更加复杂。这两者颇有点鱼和熊掌的意味：假设集不大是件好事，因为d_{VC}不会太大，有利于一般化；边界复杂也是好事，因为这会使E_{in}变小。在之前，这两种优点不可兼得，但是SVM的引入使得这两个好处有可能被一个模型同时获得（因为SVM的假设集更小）。这种模型称为非线性SVM，其使用的还是SVM的思想，但是会对数据做一些特征变换$\mathbf{\Phi}$

NTUML 18. 对偶支持向量机

提出对偶SVM的动机

上一讲提到的线性SVM可以很容易地被延展到非线性的情况，只需要引入非线性特征变换$\mathbf{\Phi}$就可以。引入以后的问题描述为

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} \\ \text{subject to} \quad & y_n(\mathbf{w}^{\mathsf{T}} \underbrace{\mathbf{z}_n}_{\mathbf{\Phi}(\mathbf{x}_n)} + b) \geq 1 \text{ for } n = 1, 2, \ldots, N \end{aligned}$$

非线性硬间隔SVM的算法就可以被修改为

> 1. 设置$\mathrm{Q} = \begin{bmatrix} 0 & \mathbf{0}_{\tilde{d}}^{\mathsf{T}} \\ \mathbf{0}_{\tilde{d}} & \mathrm{I}_{\tilde{d}} \end{bmatrix}; \mathbf{p} = \mathbf{0}_{\tilde{d}+1}; \mathbf{a}_n^{\mathsf{T}} = y_n \begin{bmatrix} 1 & \mathbf{z}_n^{\mathsf{T}} \end{bmatrix}; c_n = 1$
> 2. $\begin{bmatrix} b \\ \mathbf{w} \end{bmatrix} \leftarrow \mathrm{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$
> 3. 将$b \in \mathbb{R}$和$\mathbf{w} \in \mathbb{R}^{\tilde{d}}$返回为$g_{\text{SVM}}(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x}) + b)$

非线性硬间隔SVM的思路是将SVM的优点（间隔大，有效VC维低，鲁棒性强）和特征变换的优点（能够形成非线性的边界）结合起来。但是这里有个问题：$\mathbf{x}$经过非线性特征转换以后，映射到新空间里得到的$\mathbf{z}$通常维度比较高。记映射后向量的维度为$\tilde{d}$，则求解QP时需要面对$\tilde{d} + 1$个变量和N个限制条件。因此，需要探索一种方法使得SVM不去依赖$\tilde{d}$

这种方法的核心思路为，将原来的二次规划问题（有$\tilde{d} + 1$个变量和N个限制条件）转换为一个等价的二次规划问题（有N个变量和$N + 1$个限制条件）。这种新的问题被称为**原始SVM的对偶问题**

回顾第14讲介绍正则化时，要求解的原始问题也是带了一个限制条件。当时的做法是进行了一些变换，转而求解一个没有条件，称为“放大误差”的最优化问题。即这两个问题也存在一个等价关系：

$$\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) \text{ s.t. } \mathbf{w}^{\mathsf{T}}\mathbf{w} \leq C \Leftrightarrow \min_{\mathbf{w}} E_{\text{aug}}(\mathbf{w}) = E_{\text{in}}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^{\mathsf{T}}\mathbf{w}$$

这里用到的工具称为**拉格朗日乘子**，也就是λ。做法是把限制条件乘上一个λ，再加到要求解的式子里。这时，λ可以看做是限制条件中参数C的一个对应。同样的思想也可以用来求解前述的非线性硬间隔SVM，不过具体做法会稍微有点不同：此时这些拉格朗日乘子都会被看做是一个变量，去解跟这个变量有关的对偶问题。而每个限制条件都会对应一个拉格朗日乘子，因此最后会有N个未知的λ

所以，第一步要做的就是移除原有问题中的条件限制。这一步可以通过引入拉格朗日乘子（这里不再记为λ，而是记为α）和定义拉格朗日函数做到。这里的拉格朗日函数为

$$\mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) = \underbrace{\frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w}}_{\text{objective}} + \sum_{n=1}^{N} \alpha_n \underbrace{(1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b))}_{\text{constraint}}$$

为什么拉格朗日函数定义是合理的？下面试着进行说明：原有的问题其实等价于

$$\text{SVM} \equiv \min_{b,\mathbf{w}} \left(\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) \right)$$

这个式子的意思是

- 先考察所有b和$\mathbf{w}$。对于给定的b和$\mathbf{w}$，寻找所有可能的N个非负的α_n，满足$\mathcal{L}$最大的$\boldsymbol{\alpha}$是要找的$\boldsymbol{\alpha}$
- 这样，对所有b和$\mathbf{w}$，都能找到它们能做到的$\mathcal{L}$的最大值。在这些最大值中，求一个最小值。满足这个最小值的b和$\mathbf{w}$就是要找的b和$\mathbf{w}$

那么问题来了，为什么这个等价关系存在？事实上，对所有的b和$\mathbf{w}$，可以分为两组：

- “坏的”，也就是违反了（某一个）限制条件的b和$\mathbf{w}$（violate）。限制条件要求$y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b) \geq 1$，对应的就是$1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) \leq 0$。如果这组$b$和$\mathbf{w}$违反了限制条件，那么就会有$1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{x}_n + b) > 0$。将其带入到拉格朗日函数里，求和项中有至少一项的限制条件就会是个正数。前面说，要找到所有可能的N个非负的α_n，满足$\mathcal{L}$最大。由于目标函数是非负的，限制项中又有至少一个正数，因此只要让限制条件为正数的项的系数随意大就可以无限制地最大化$\mathcal{L}$。即在此情况下，$\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) \to +\infty$
- “好的”，也就是遵守全部限制条件的b和$\mathbf{w}$（feasible）。在这种情况下，拉格朗日函数中所有限制项都是非正数。为了让$\mathcal{L}$尽可能大，最方便的方法就是让所有$\alpha_n = 0$，这个时候$\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) = \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w}$。再对这些$b$和$\mathbf{w}$找到最小的$\mathcal{L}$，其实就是去找到最小的$\frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w}$，跟原来要求解的最优化问题一样

也就是说，原来的限制条件隐藏在了内层的最大化函数里。即

$$\text{SVM} \equiv \min_{b,\mathbf{w}} \left(\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) \right) = \min_{b,\mathbf{w}} \left(\infty \text{ if violate; } \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} \text{ if feasible} \right)$$

拉格朗日对偶SVM

现在的问题是，如何继续对问题做转换。由于内层是一个最大化函数，因此对任意给定的$\boldsymbol{\alpha}'$ $(\forall \alpha'_n \geq 0)$，肯定有

$$\min_{b,\mathbf{w}} \left(\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) \right) \geq \min_{b,\mathbf{w}} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}')$$

因为任何函数的最大值肯定比其任意给定参数对应的值要大。进一步地，使得上面不等式右侧式子取得最大值的$\boldsymbol{\alpha}'$，肯定还有（由于$\boldsymbol{\alpha}'$的任意性，可以把$\boldsymbol{\alpha}'$再用$\boldsymbol{\alpha}$代替）

$$\min_{b,\mathbf{w}} \left(\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) \right) \geq \max_{\text{all } \alpha_n \geq 0} \left(\min_{b,\mathbf{w}} \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha}) \right)$$

（课程中讲的意思大概是说，因为右边不等式是对任意可能的值求极值，肯定不如上来直接求到的极值大。不过这实际上是max-min不等式。化用wiki里的证明方式，可以做简单的证明

$$\begin{aligned}
&\text{Define } g(x) = \min_y f(x, y) \\
&\Rightarrow g(x) \leq f(x, y), \forall x, y \\
&\Rightarrow \max_x g(x) \leq \max_x f(x, y), \forall y \\
&\Rightarrow \max_x \min_y f(x, y) \leq \max_x f(x, y), \forall y \\
&\Rightarrow \max_x \min_y f(x, y) \leq \min_y \max_x f(x, y) \quad \blacksquare
\end{aligned}$$

也就对应了民间谚语“瘦死的骆驼比马大”）

这样做实际上交换了min和max的顺序，得到的问题称为是**拉格朗日对偶问题**，而对偶问题的解是原问题的解的下限。事实上，“$\geq$”是一个弱对偶关系，只有当两边的式子相等时，这两个问题才构成强对偶关系。对于二次规划问题，如果满足以下三个条件（统称为**约束规范**），则等号成立：

- 原问题是凸的
- 原问题是可解的
- 原问题的约束条件是线性的

原始SVM问题正好满足上述约束规范，因此等号成立（如果经过多项式变换$\mathbf{\Phi}$数据集在新的空间线性可分，那么原问题就是可解的）。在强对偶关系下，解左边的问题和解右边的问题一样，或者换句话说，使右式最优的$(b, \mathbf{w}, \boldsymbol{\alpha})$也可以使左式最优，反之亦然

这样，就可以写出对偶问题的完整形式

$$\max_{\text{all } \alpha_n \geq 0} \left(\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \sum_{n=1}^{N} \alpha_n (1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b)) \right)$$

现在，最小化的这个函数（也就是拉格朗日函数）没有任何限制条件，可以看看能不能化简。由于其没有限制条件，因此最优的b肯定使函数对其的偏导数为0，即

$$\frac{\partial \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha})}{\partial b} = 0 \Rightarrow -\sum_{n=1}^{N} \alpha_n y_n = 0$$

由于最优解必定导致如上的结论，因此把这个结论作为条件加上，并不会改变最优解。而且由于拉格朗日函数展开以后关于b的项是$-\sum_{n=1}^{N} \alpha_n y_n \cdot b = -b\sum_{n=1}^{N} \alpha_n y_n$，由于$\sum_{n=1}^{N} \alpha_n y_n = 0$，因此加上这个条件以后，可以直接把关于$b$的项去掉，即对偶问题可以化简为

$$\max_{\text{all } \alpha_n \geq 0, \sum y_n\alpha_n = 0} \left(\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \sum_{n=1}^{N} \alpha_n (1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n)) \right)$$

同样的道理，现在对$\mathbf{w}$求偏导数，有

$$\frac{\partial \mathcal{L}(b, \mathbf{w}, \boldsymbol{\alpha})}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n$$

将上面的式子作为约束条件，并将$\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n$代入式子，且考虑到$\mathbf{w}^\mathsf{T}\mathbf{w} = \|\mathbf{w}\|^2$，可以得到

$$\begin{aligned}
&\max_{\text{all } \alpha_n \geq 0, \sum y_n\alpha_n = 0} \left(\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \sum_{n=1}^{N} \alpha_n (1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n)) \right) \\
\Leftrightarrow &\max_{\text{all } \alpha_n \geq 0, \sum y_n\alpha_n = 0, \mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n} \left(\min_{\mathbf{w}} \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + \sum_{n=1}^{N} \alpha_n - \mathbf{w}^\mathsf{T}\mathbf{w} \right) \\
\Leftrightarrow &\max_{\text{all } \alpha_n \geq 0, \sum y_n\alpha_n = 0, \mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n} \left(-\frac{1}{2}\left\| \sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n \right\|^2 + \sum_{n=1}^{N} \alpha_n \right)
\end{aligned}$$

这就转化成了一个几乎只有$\boldsymbol{\alpha}$的最优化问题，实际上是简化版的拉格朗日对偶问题。也就是说，最优的$\boldsymbol{\alpha}$和最优的b和$\mathbf{w}$需要满足一些关系，它们才能同时是原问题和对偶问题的最优解。这些关系统称为KKT条件（Karush-Kuhn-Tucker conditions），包括

- $y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b) \geq 1$
- $\alpha_n \geq 0$
- $\sum y_n\alpha_n = 0; \mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n$
- $\alpha_n(1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b)) = 0$

之后，就要用KKT条件求解最优的$\boldsymbol{\alpha}$，进而求解最优的$(b, \mathbf{w})$

求解对偶SVM

将之前的对偶问题稍作如下变换

- 将求最大值问题转为求最小值问题（取相反数即可）
- 展开向量的内积

可以得到硬间隔SVM对偶问题的标准形式：

$$\begin{aligned}\min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\alpha_n\alpha_m y_n y_m \mathbf{z}_n^{\mathsf{T}}\mathbf{z}_m - \sum_{n=1}^{N}\alpha_n \\ \text{subject to} \quad & \sum_{n=1}^{N} y_n\alpha_n = 0; \\ & \alpha_n \geq 0, \text{for } n = 1, 2, \ldots, N\end{aligned}$$

这是一个有N个变量和$N+1$个约束条件的凸二次规划问题，可以将其转换成二次规划的标准形式

$$\begin{aligned}\text{optimal } \boldsymbol{\alpha} &\leftarrow \mathrm{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c}) \\ q_{n,m} &= y_n y_m \mathbf{z}_n^{\mathsf{T}}\mathbf{z}_m \\ \mathbf{p} &= -\mathbf{1}_N \\ \mathbf{a}_{\geq} &= \mathbf{y}, \mathbf{a}_{\leq} = -\mathbf{y}; \mathbf{a}_n^{\mathsf{T}} = n\text{th. unit direction} \\ c_{\geq} &= 0, c_{\leq} = 0; c_n = 0\end{aligned}$$

记对偶问题的Q为Q_{D}，由上面的定义可知Q是稠密矩阵。如果$N = 30000$，那Q可以占3G内存！所以要解对偶问题，需要一个特殊的求解器，它不存储整个Q_{D}或者使用某些特殊的限制，以应付N很大的情况

求出最优的$\boldsymbol{\alpha}$以后，可以根据KKT条件算出最优的b和$\mathbf{w}$。最优的$\mathbf{w}$很容易求，$\mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n$。而由$\alpha_n(1 - y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b)) = 0$，可知如果有任意$\alpha_n > 0$，就可以算出$b = y_n - \mathbf{w}^{\mathsf{T}}\mathbf{z}_n$（不同的$\alpha_n > 0$应该算出同样的$b$）。注意此时有$y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b) = 1$，即对任一点$n$如果有$\alpha_n > 0$，则说明这个点在“胖”的边界上

（这里其实说明了将原始问题转化成对偶问题的两个好处：

1. 只需要优化$\boldsymbol{\alpha}$而不是b和$\mathbf{w}$，减小了计算量，降低了算法的时间复杂度
2. 通过判断α_n是否为0可以找出支持向量）

对偶SVM传递的信息

在前一讲中曾经得到一个结论，即支持向量机只关注那些在“胖”边界上的点，而其它点则不需要。上一节又证明，如果某个点有$\alpha_n > 0$，则这个点一定在“胖”边界上。称所有$\alpha_n > 0$的样本$(\mathbf{z}_n, y_n)$为**支持向量**。也就是说有$\mathrm{SV}(\text{positive } \alpha_n) \subseteq \text{SV candidates (on boundary)}$。这意味着只需要用支持向量计算$\mathbf{w}$和$b$（对于那些不是支持向量的点，算出来的$\mathbf{w}$和$b$也都是0），而SVM也是通过对偶问题的最优解找出支持向量（即有用的向量），进而学到最大间隔超平面的算法

因此，SVM的权重可以写为$\mathbf{w}_{\mathrm{SVM}} = \sum_{n=1}^{N}\alpha_n(y_n\mathbf{z}_n)$，其中$\alpha_n$来自于对偶问题的解。这个形式其实跟之前PLA算法得到的权重类似：PLA的权重可以写为$\mathbf{w}_{\mathrm{PLA}} = \sum_{n=1}^{N}\beta_n(y_n\mathbf{z}_n)$，其中$\beta_n$是对错分样本进行纠正的次数。即SVM和PLA得到的解都是原始数据的线性组合。更广泛地讲，对基于梯度下降的算法，如果设初始的$\mathbf{w}_0 = 0$，那么最后求出的解还是原始数据的线性组合。称这种现象为**最优的权重$\mathbf{w}$可以被数据表示**。其中SVM更特殊一些，最优的权重只用支持向量就能表示（PLA是用错分点表示）

这样，到目前为止，一共介绍了两种SVM：

- 原始的硬间隔SVM，一共有$\tilde{d} + 1$个变量和N个限制条件。如果数据维度比较小，则适用。其物理意义是对$(b, \mathbf{w})$进行缩放，得到一个合适的值
- 对偶的硬间隔SVM，一共有N个变量和$N+1$个简单的限制条件。如果数据量比较小，则适用。其物理意义是找出起作用的支持向量和它们的α_n

无论使用哪种做法，都可以得到最优的$(b, \mathbf{w})$，进而得到最优的假设函数为

$$g_{\mathrm{SVM}}(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^{\mathsf{T}}\boldsymbol{\Phi}(\mathbf{x}) + b)$$

但是问题还没有结束。回想提SVM对偶问题的动机，是为了避免对数据维度$\tilde{d}$的依赖。尽管看上去最后求解时只依赖了数据量N（求解N个变量和$N+1$个限制条件），但是构造矩阵Q_{D}时还是在$\mathbb{R}^{\tilde{d}}$空间做内积。如果直接计算的话，运算的时间复杂度还是$O(\tilde{d})$。因此，之后要解决的问题就是如何避免直接计算内积

NTUML 19. 核支持向量机

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/09/23/NTUML-19-Kernel-SVM/

核技巧

前面提到，对偶问题最后也能写成二次规划问题的标准形式。乍一看，问题的变量数为N，条件数为$N+1$，跟数据的维度$\tilde{d}$无关，但是需要注意的是，$\mathrm{Q_D}$中的每一项计算方式为两个y相乘再乘上两个$\mathbf{z}$的内积（$q_{n,m}=y_n y_m \mathbf{z}_n^\mathsf{T}\mathbf{z}_m$}），这个内积就跟$\tilde{d}$有关了。如果数据维度很大，那么$\mathrm{Q_D}$的计算量就很大，成为了求解问题的瓶颈所在。如果使用某种技巧把$\mathbf{z}_n^\mathsf{T}\mathbf{z}_m$这个直接做内积步骤替换掉，那么计算效率就能得到比较大的提升

经过进一步分析可以发现，之前的讨论都是在$\mathcal{Z}$空间做的。但是$\mathbf{z}$实际上是原始数据$\mathbf{x}$经过非线性变换$\mathbf{\Phi}$得到，因此求$\mathbf{z}_n^\mathsf{T}\mathbf{z}_m$实际上是两步：先做变换，再内积。这两个步骤如果能合起来，是不是计算会快一点？

先从一个比较简单的例子入手：考虑之前提到的二次多项式变换

$$\mathbf{\Phi}_2(\mathbf{x})=(1,x_1,x_2,\dots,x_d,x_1^2,x_1x_2,\dots x_1x_d,x_2x_1,x_2^2,\dots,x_2x_d,\dots,x_d^2)$$

这里为了简单起见，将x_1x_2和x_2x_1都放了进来。接下来，假设有两个数据点$\mathbf{x}$和$\mathbf{x}'$，看一看先转换再做内积的结果，有

$$\begin{aligned}\mathbf{\Phi}_2(\mathbf{x})^\mathsf{T}\mathbf{\Phi}_2(\mathbf{x}')&=1+\sum_{i=1}^d x_ix_i'+\sum_{i=1}^d\sum_{j=1}^d x_ix_jx_i'x_j'\\&=1+\sum_{i=1}^d x_ix_i'+\sum_{i=1}^d x_ix_i'\sum_{j=1}^d x_jx_j'\\&=1+\mathbf{x}^\mathsf{T}\mathbf{x}'+(\mathbf{x}^\mathsf{T}\mathbf{x}')(\mathbf{x}^\mathsf{T}\mathbf{x}')\end{aligned}$$

化简以后，$\mathbf{\Phi}_2(\mathbf{x})^\mathsf{T}\mathbf{\Phi}_2(\mathbf{x}')$的计算复杂度从$O(d^2)$降到了$O(d)$。因此，在某些情况下，如果把变换和内积这两个步骤合起来，计算效率可以得到提高。这里可以把转换+内积这个合并步称为**核函数**，即对变换$\mathbf{\Phi}$有对应的核函数$K_{\mathbf{\Phi}}$满足

$$K_{\mathbf{\Phi}}(\mathbf{x},\mathbf{x}')\equiv\mathbf{\Phi}(\mathbf{x})^\mathsf{T}\mathbf{\Phi}(\mathbf{x}')$$

在本例中，有

$$\mathbf{\Phi}_2\Leftrightarrow K_{\mathbf{\Phi}_2}(\mathbf{x},\mathbf{x}')=1+(\mathbf{x}^\mathsf{T}\mathbf{x}')+(\mathbf{x}^\mathsf{T}\mathbf{x}')^2$$

有了核函数以后，原始对偶SVM的算法可以有如下几点修改

- 矩阵$\mathrm{Q_D}$中的每一项$q_{n,m}$的计算得到了简化：$q_{n,m}=y_ny_m\mathbf{z}_n^\mathsf{T}\mathbf{z}_m=y_ny_mK(\mathbf{x}_n,\mathbf{x}_m)$。注意这里给核函数传递的是在原始空间$\mathcal{X}$的向量，提出核函数也就是为了避免在高维空间的计算
- 计算最优的偏置b时，也会用到。具体为（注意$\mathbf{w}=\sum_{n=1}^N\alpha_ny_n\mathbf{z}_n$）

  $$b=y_s-\mathbf{w}^\mathsf{T}\mathbf{z}_s=y_s-\left(\sum_{n=1}^N\alpha_ny_n\mathbf{z}_n\right)^\mathsf{T}\mathbf{z}_s=y_s-\sum_{n=1}^N\alpha_ny_n\left(K(\mathbf{x}_n,\mathbf{x}_s)\right)$$

  这里$(\mathbf{x}_s,y_s)$是得到的任意一个支持向量
- 类似地，得到最优的$\mathbf{w}$和b以后，对新的测试输入$\mathbf{x}$，也可以引入核函数来做预测

  $$g_{\mathrm{SVM}}(\mathbf{x})=\mathrm{sign}(\mathbf{w}^\mathsf{T}\mathbf{\Phi}(\mathbf{x})+b)=\mathrm{sign}\left(\sum_{n=1}^N\alpha_ny_nK(\mathbf{x}_n,\mathbf{x})+b\right)$$

可以注意到，原有对偶问题求解的时候，里面所有要在$\tilde{d}$维空间$\mathcal{Z}$做的操作，最后都可以在d维空间$\mathcal{X}$通过核函数K得到相等的结果。这种方法也称为“核技巧”，即把所有转换-内积的步骤都换成计算核函数的值，因此在算法里就不会依赖$\tilde{d}$了

引入核函数以后，就可以得到核硬边界SVM算法（Kernel Hard-Margin SVM Algorithm）

> 1. $q_{n,m}=y_ny_mK(\mathbf{x}_n,\mathbf{x}_m);\mathbf{p}=-\mathbf{1}_N$（A和c还是对偶硬边界SVM算法中的对应值）
> 2. $\boldsymbol{\alpha}\leftarrow\mathrm{QP}(\mathrm{Q_D},\mathbf{p},\mathrm{A},\mathbf{c})$
> 3. 任选一个支持向量$(\mathbf{x}_s,y_s)$，计算$b\leftarrow(y_s-\sum_{\text{SV indices}}\alpha_ny_nK(\mathbf{x}_n,\mathbf{x}_s))$
> 4. 将所有支持向量、对应的α_n和b返回。对新的$\mathbf{x}$，计算
>
> $$g_{\mathrm{SVM}}(\mathbf{x})=\mathrm{sign}\left(\sum_{\text{SV indices}}\alpha_ny_nK(\mathbf{x}_n,\mathbf{x})+b\right)$$

假设核函数的计算时间为k，则算法中第一步的时间复杂度仅为$O(kN^2)$，第二步只有N个变量和N个限制条件，第三、四步的时间复杂度为$O(k\#\text{SV})$。因此可以看到，引入核函数以后，就可以完全避免在$\tilde{d}$维空间做计算，而且核SVM的预测还是只需要支持向量就可以

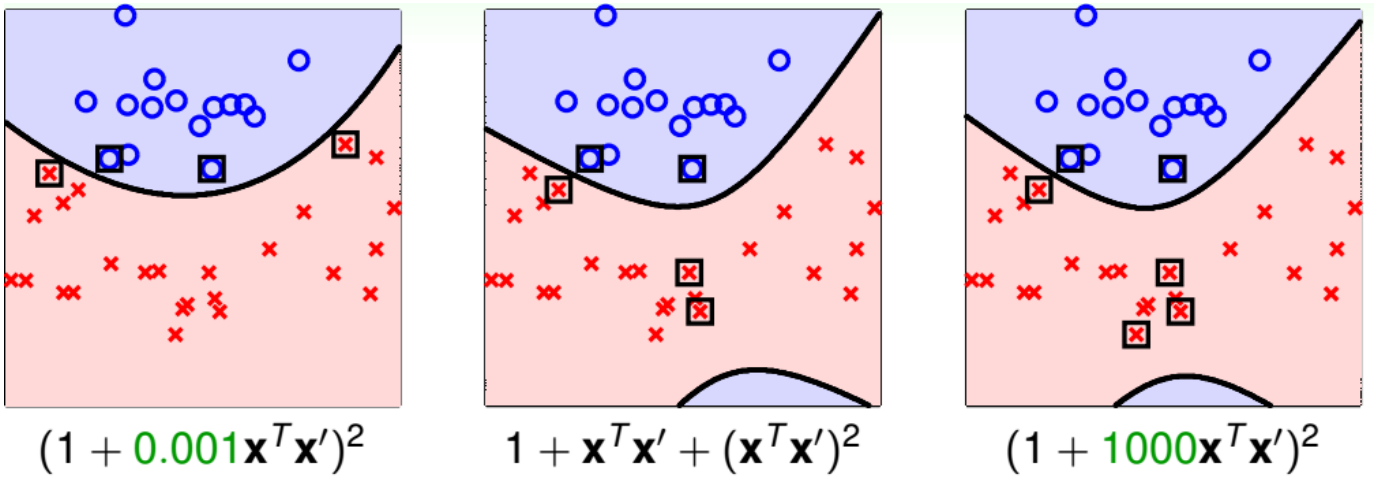
多项式核

除去上一节给出的变换，还有其它一些比较常见的二次多项式变换方法，例如

$$\begin{aligned}\boldsymbol{\Phi}_2(\mathbf{x}) = (1, \sqrt{2}x_1, \dots, \sqrt{2}x_d, x_1^2, \dots, x_d^2) &\Leftrightarrow K_{2'}(\mathbf{x}, \mathbf{x}') = 1 + 2\mathbf{x}^\mathsf{T}\mathbf{x}' + (\mathbf{x}^\mathsf{T}\mathbf{x}')^2 \\ \boldsymbol{\Phi}_2(\mathbf{x}) = (1, \sqrt{2\gamma}x_1, \dots, \sqrt{2\gamma}x_d, \gamma x_1^2, \dots, \gamma x_d^2) &\Leftrightarrow K_2(\mathbf{x}, \mathbf{x}') = 1 + 2\gamma\mathbf{x}^\mathsf{T}\mathbf{x}' + \gamma^2(\mathbf{x}^\mathsf{T}\mathbf{x}')^2 \\ &= (1 + \gamma\mathbf{x}^\mathsf{T}\mathbf{x}')^2 \quad (\gamma > 0)\end{aligned}$$

可以看到K_2的计算比$K_{\boldsymbol{\Phi}_2}$更简单一些，而能力相同（两者映射到的空间相同）。但是要注意的是，转换不同，内积就不同。内积不同，距离就不同。而SVM是由距离决定的，因此会得到不同的边界。实际上K_2更常用

下图中给出了使用不同的核和同一个核使用不同γ的效果。其中带方框的点是支持向量



使用不同的核和同一个核使用不同gamma的SVM。其中带方框的点是支持向量

由上图可知，不同的g_{SVM}会使用不同的支持向量，也就会产生的不同的边界。而使用什么核和γ也是很难在学习之前就能判断出来的。究竟哪个核，这个核的哪个γ最合适，也得通过交叉验证的方法来判断

进一步地，可以加入两个新的变量ζ和多项式次数Q，得到一个更广义的多项式核，即（均设定$\gamma > 0, \zeta \geq 0$）

$$\begin{aligned}K_2(\mathbf{x}, \mathbf{x}') &= (\zeta + \gamma\mathbf{x}^\mathsf{T}\mathbf{x}')^2 \\ K_3(\mathbf{x}, \mathbf{x}') &= (\zeta + \gamma\mathbf{x}^\mathsf{T}\mathbf{x}')^3 \\ &\vdots \\ K_Q(\mathbf{x}, \mathbf{x}') &= (\zeta + \gamma\mathbf{x}^\mathsf{T}\mathbf{x}')^Q\end{aligned}$$

这说明，原来的Q次多项式变换$\boldsymbol{\Phi}_Q$在引入参数为(ζ, γ)的核以后，可以得到相同的效果。而核函数的计算方法又使得不再需要将经过$\boldsymbol{\Phi}_Q$变换得到的$\mathbf{z}$展开，极大加快了计算效率。这种将SVM和多项式核结合起来的方法称为**多项式SVM**

多项式核里比较特殊的一种是$\zeta = 0, \gamma = 1, Q = 1$。这种核称为线性核。尽管高次多项式核非常强大，但是真正实验模型时还是应该先从线性核这种简单模型试起

高斯核

由上一节可知，如果设计一个很好的核函数，则可以使用$O(d)$的时间复杂度计算原始数据被映射到$\tilde{d}$维空间以后的内积。那么有没有可能有这么一种核函数，使得可以很方便地计算*无限维*空间的内积呢？的确存在这么一种核函数。接下来的推导中，为了方便起见，假设$\mathbf{x}$只有一个维度，记为x，则

$$
\begin{aligned}
K(x, x') &= \exp(-(x-x')^2) \\
&= \exp(-(x)^2)\exp(-(x')^2)\exp(2xx') \\
&= \exp(-(x)^2)\exp(-(x')^2)\left(\sum_{i=0}^{\infty}\frac{(2xx')^i}{i!}\right) \qquad \text{Taylor expansion} \\
&= \sum_{i=0}^{\infty}\left(\exp(-(x)^2)\exp(-(x')^2)\sqrt{\frac{2^i}{i!}}\sqrt{\frac{2^i}{i!}}(x)^i(x')^i\right) \\
&= \mathbf{\Phi}(x)^{\mathsf{T}}\mathbf{\Phi}(x')
\end{aligned}
$$

即多项式变换

$$
\mathbf{\Phi}(x) = \exp(-x^2)\cdot\left(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \dots\right)
$$

可以把x映射到一个无限维空间，而在该空间两个元素的内积可以在原始空间很方便地使用高斯函数求解。更普遍性地，称形如

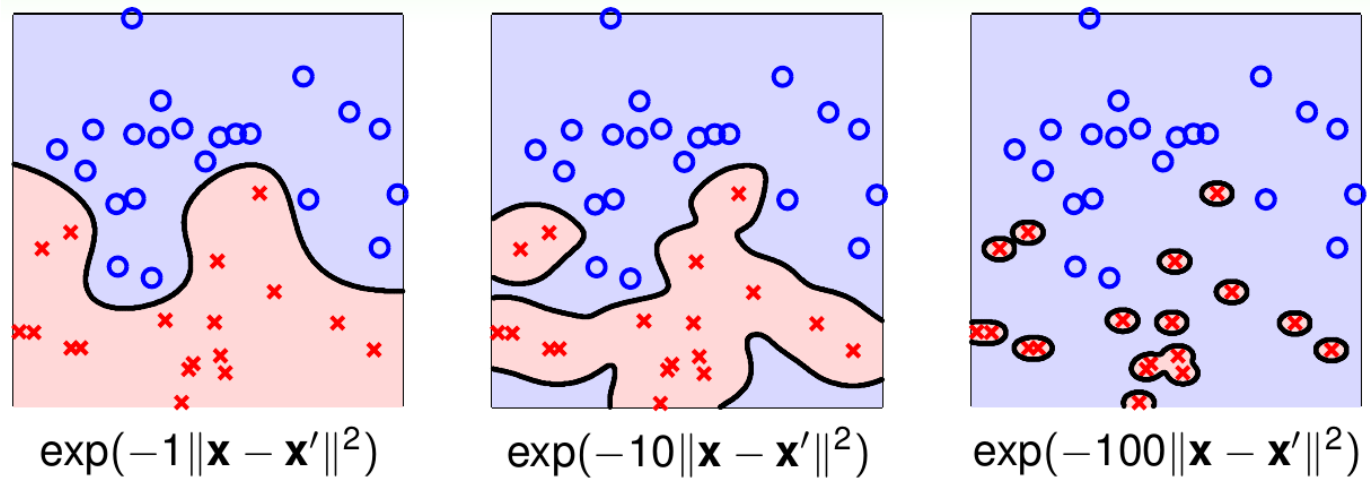
$$
K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}'\|^2\right), \qquad \gamma > 0
$$

这样的核函数为**高斯核**

将高斯核的表示代入到核SVM中，可以得到使用高斯核的SVM的形式

$$
\begin{aligned}
g_{\text{SVM}}(\mathbf{x}) &= \text{sign}\left(\sum_{\text{SV}}\alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b\right) \\
&= \text{sign}\left(\sum_{\text{SV}}\alpha_n y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right) + b\right)
\end{aligned}
$$

此时SVM实际上是以各支持向量$\mathbf{x}_n$为中心的高斯函数的线性组合。考虑到“线性组合”的意义，高斯核通常也称为径向基函数核（Radial Basis Function kernel, RBF kernel）。高斯核函数的使用进一步展示了核SVM的强大：不需要太多的数据点（仅使用支持向量），也不需要太多的计算（仅在原始d维空间做向量运算），就能得到特别复杂的边界（因为高斯核实际上隐含地将原始数据映射到了无限维空间），而且很鲁棒（SVM本身最大边界的性质）。核函数的简洁性使得我们也不需要去显式地把转换$\mathbf{\Phi}$写出来，对偶SVM的性质使得我们不需要去存储最优的$\mathbf{w}$，只需要存好哪些是支持向量以及这些支持向量对应的α_n就可以



使用不同gamma的高斯核的比较

上图给出了使用不同γ的高斯核SVM的示意图。可以看出，γ越大，对应的高斯函数越尖锐，模型越容易过拟合。这里也说明SVM是不容易过拟合，但是不会永远不过拟合。使用高斯核时，γ的选取仍然是需要仔细考虑的。当$\gamma \to \infty$时，如果K是高斯核函数，则$K(\mathbf{x}, \mathbf{x}') = [\![\mathbf{x} = \mathbf{x}']\!]$

核的比较

本讲一共介绍了三种常见的核，下面对它们做一比较

- 最简单的线性核$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^{\mathsf{T}}\mathbf{x}'$实际上没有对原始数据做任何变化。这种核最简单最安全，而且最快（直接对原始问题用QP求解器求解）。最后得到的模型解释性也比较强，$\mathbf{w}$会给出每个特征的权重，支持向量会给出每个数据点的重要性。其坏处是可用的场景比较有限，数据并不总是线性可分的。然而无论如何，线性核都是最简单直接的工具，应该在最开始的时候就试用一下

- 多项式核$K(\mathbf{x},\mathbf{x}')=(\zeta+\gamma\mathbf{x}^{\mathsf{T}}\mathbf{x}')^{Q}$的好处是可用场景比线性核要广泛，而且工程师可以通过限制Q的大小来对模型的复杂度加以控制。其坏处是当Q比较大时，如果$|\zeta+\gamma\mathbf{x}^{\mathsf{T}}\mathbf{x}'|<1$则核函数值会逼近0，如果$|\zeta+\gamma\mathbf{x}^{\mathsf{T}}\mathbf{x}'|>1$则核函数值会非常大。总体来讲，数值计算会不太稳定。而且，这个模型里有三个参数(γ,ζ,Q)，调参也比较困难。因此，多项式核可能仅适用于Q比较小的场景，而且在这种情况下有时反而把$\mathbf{\Phi}(\mathbf{x})$展开然后代入到线性核求解效率更高
- 高斯核$K(\mathbf{x},\mathbf{x}')=\exp\left(-\gamma\|\mathbf{x}-\mathbf{x}'\|^2\right)$可以算出最复杂的边界，而且由于其值域为$(0,1]$，计算稳定性也比较高，不会出现浮点数计算的上溢和下溢。这个核函数只有一个参数γ可调，也比较容易调参。但是由于它把原始数据映射到无限维空间，因此没有一个直观的$\mathbf{w}$可以用来解释模型。它的计算也比线性核慢，也更容易过拟合。总体来讲，高斯核是一种最常用的核函数，但是用的时候一定要小心

当然，还可以使用其它核函数。从本质上看，核函数实际上是相似性的某种度量方式，其度量的是$\mathbf{x}$和$\mathbf{x}'$经过变换以后在空间$\mathcal{Z}$中的相似性。那么所有相似性都能用一个合法的核函数来表示呢？并非总是如此。给定函数K，令$k_{ij}=K(\mathbf{x}_i,\mathbf{x}_j)$，则矩阵$\mathbf{K}$必须同时满足以下两个条件（充要条件），$K$才是合法的核函数。这两个条件包括

- $\mathbf{K}$是对称的
- K必须是半正定的

这个条件称为**Mercer条件**

自定义核是可能的，不过需要注意的是，也是很难的

额外的话

我大概算了算，这应该是我第五遍听关于SVM的课了（吴恩达Coursera课、吴恩达CS229、白思理哥伦比亚大学EdX课、林轩田课x2）。如果算上看pluskid大神、JerryLead大神两神的文章，SVM的理论我不知道过了多少遍。但是遗憾的是，一是由于我天性驽钝，二是因为我有学而不思的毛病，关于“核方法”这个概念，我每次都是很快拾起，然后更快忘记。这次重听林轩田老师的课，我决定努力从各个方面试着战胜自己一次。其中之一就是试着写一点自己关于核方法的理解，让自己能多思考一点，进而把核方法这个概念和思想留在脑子里的时间延长一些，+ns

既然是自己理解的，肯定会有这样那样的问题。欢迎在知乎专栏里指正

首先一点，核方法是一个独立的方法。大部分课程在讲授核方法的时候，都会在讲SVM时引入它。相比较而言，白思理的课程更微妙一些，是先讲核再讲SVM。所以我觉得核方法和SVM的关系应该是这样：核方法提供了一种在低维空间对向量做计算的方式，通过这种方式计算出来的结果，与通过某种映射将原始向量映射到高维空间以后在高维空间做内积的结果是一致的。SVM的对偶问题里计算$\mathbf{Q}$时、根据支持向量计算b时以及对新的$\mathbf{x}$计算估计值$\hat{y}$时，都有高维空间计算内积的需求，而这个计算恰好能用核函数在原始空间计算的结果替代，如此而已。更简单地说，是SVM可以用到核方法，而不是SVM产生了核方法

其次一点，在很多讲核SVM的讲义里，都可以看到类似这样的写法（事实上，在林的课件里，就有类似的写法）：

$$K\langle\mathbf{x},\mathbf{x}'\rangle=\mathbf{\Phi}(\mathbf{x})^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x}')$$

初学者最开始看到这个式子的时候，不知道会不会像驽钝的我一样，开始好奇一些其实不用好奇的问题，例如：

- 尖角符号代表了什么意义啊？
- 这个对应的$\mathbf{\Phi}$应该是怎样的啊？
- 映射了以后的那个高维空间$\mathcal{Z}$应该是什么样的啊？

在这里我试着回答以前的我一下

- 尖角符号没毛意义，写成$K(\mathbf{x},\mathbf{x}')$这种普通函数的形式一点问题都没有。核方法的核心是核函数，核函数，归根到底，还是函数，就是给你两个向量，定义一种计算方法，没了。值得玩味的是，这种计算方法里一般会有原始向量的内积运算。多项式核里这个内积是写在明面上的，高斯核稍微不那么明显，但是考虑两点：首先，$\|\mathbf{v}\|^2=\mathbf{v}^{\mathsf{T}}\mathbf{v}$，其次，看上面展开计算的式子，其实也有内积的影子
- K对应的$\mathbf{\Phi}$和$\mathcal{Z}$是什么样的，这个在理论证明提出的核函数有效时可能应该需要分析，但是如果只站在应用层面，我觉得是不用太关心的。事实上，提出核函数的目的就是为了避免直接像高维空间的变换和在高维空间的计算，如果问题使得我们必须要提出一个自定义的核函数，证明时只需要证明 1. 这个新的核函数确实好用 2. 这个核函数满足Mercer条件，可能就可以了。另外，讲义中给出了K对应的矩阵$\mathbf{K}$，我甚至怀疑在证明$\mathbf{K}$是半正定的时候，也不需要把所有$\mathbf{\Phi}(\mathbf{x}_i)^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x}_j)$都列出来放在一个大的矩阵里，肯定有其它更抽象的证明方法

所以最后再唠叨一句，核方法的意义就在于，我们只需要关心核函数K是什么样的就行，它后头的$\mathbf{\Phi}$, $\mathcal{Z}$都是透明的了

另外，关于核方法，我的好友咲神写了两篇更干，更数学的博客，也可以作为参考

NTUML 20. 软间隔支持向量机

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/09/23/NTUML-20-Soft-Margin-SVM/

动机与原始问题

上一讲最后提到，SVM也会出现过拟合的情况。那么在什么时候SVM会过拟合呢？其中一种情况是使用太强力的$\mathbf{\Phi}$，而另一种情况是坚持认为数据是完全可分的。回顾之前的理论，如果总是坚持要完全可分输入数据，就是要找到一个g使得输入数据可以被g打散，因此也就增大了过拟合的可能性

回想之前介绍的口袋算法，这个算法对那些线性不可分的数据集做了一些退让：不再要求得到的超平面可以完全分开已知数据，但求最优超平面犯的错误越少越好，即口袋算法得到的b, $\mathbf{w}$满足

$$\min_{b,\mathbf{w}} \sum_{n=1}^{N} [\![y_n \neq \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b)]\!]$$

相比而言，前面提到的硬间隔SVM的条件实在是太苛刻了：不仅要求每个样本都分对，还要求每个样本离分离超平面有一定的距离。因此，可以适当放松要求，让它对错分的样本不再施加到超平面的距离限制，只控制错分样本的数目让其越少越好（即不让模型无止境地犯错）。这样，得到的优化问题变成了

$$\begin{aligned} \min_{b,\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C \cdot \sum_{n=1}^{N} [\![y_n \neq \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b)]\!] \\ \text{s.t.} \quad & y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b) \geq 1 \text{ for correct } n \\ & y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b) \geq -\infty \text{ for incorrect } n \end{aligned}$$

这里C控制了在训练集上准确性和间隔大小这两者之间的平衡。C越大，越倾向于不要再训练集上犯错误；C越小，越倾向于在正确分类的数据上分类器间隔尽可能大

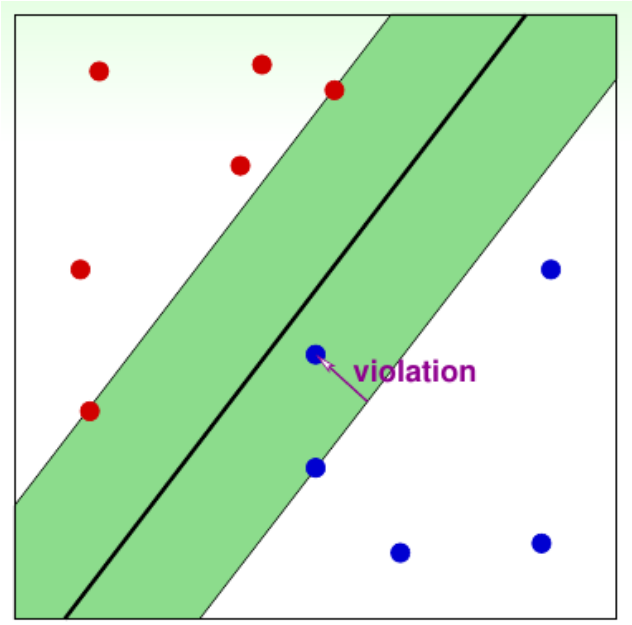
上述问题可以对表达形式做一步化简，得到一个等价的问题

$$\begin{aligned} \min_{b,\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C \cdot \sum_{n=1}^{N} [\![y_n \neq \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b)]\!] \\ \text{s.t.} \quad & y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b) \geq 1 - \infty \cdot [\![y_n \neq \text{sign}(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b)]\!] \end{aligned}$$

注意这个问题的限制条件里有N个布尔表达式$[\![\cdot]\!]$，因此不是线性约束，整个问题不能再用二次规划求解器来求解，之前的对偶方法、核方法也就没法做了。而且“预测错”也有两种错误的方法，一种是离边界很近的错误，一种是离边界很远的错误，这个问题也没法区分这两种错误，因此需要对这个问题进行修改。一种修改方法是对每个数据点引入ξ_n这个变量，表示分类器在这个数据点上犯的错误的严重性。最优化问题里要优化的问题也不再简单记录“犯了多少错误”，而是记录所有数据点总共犯错误的情况。即要求解的问题和约束为如下形式

$$\begin{aligned} \min_{b,\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n \\ \text{s.t.} \quad & y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n + b) \geq 1 - \xi_n,\ \xi_n \geq 0 \forall n \end{aligned}$$

这样一来，所有约束条件都是线性约束，要求解的目标问题是二次函数，又可以用QP求解器求解了。这种问题称为**软间隔支持向量机**，其中ξ_n记录的是数据点的“间隔破坏程度”，如下图所示



软间隔SVM的示意图。图中在间隔内的蓝点是破坏点，紫色的距离长度是它对间隔的破坏程度，也就是问题描述中的xi_n

上面描述的问题中，C越大，表示对错误的容忍程度越小，对所有点尽可能不错分；C越小，表示对错误的容忍程度越大，要求的间隔越宽

新的问题有$\tilde{d}+1+N$个变量，有$2N$个约束条件。接下来，就要用对偶问题来移除问题对$\tilde{d}$的依赖

对偶问题

对原始问题

$$\begin{aligned}\min_{b,\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}+C\cdot\sum_{n=1}^{N}\xi_n \\ \text{s.t.} \quad & y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n+b)\geq 1-\xi_n,\ \xi_n\geq 0 \forall n\end{aligned}$$

可以看到对每个点都有两个约束条件。引入拉格朗日乘子α_n和β_n，可以得到拉格朗日函数

$$\begin{aligned}\mathcal{L}(b,\mathbf{w},\boldsymbol{\xi},\boldsymbol{\alpha},\boldsymbol{\beta}) =& \frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}+C\cdot\sum_{n=1}^{N}\xi_n \\ &+\sum_{n=1}^{N}\alpha_n\cdot(1-\xi_n-y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n+b))+\sum_{n=1}^{N}\beta_n\cdot(-\xi_n)\end{aligned}$$

对应的对偶问题可以写为

$$\max_{\alpha_n\geq 0,\beta_n\geq 0}\left(\min_{b,\mathbf{w},\boldsymbol{\xi}}\frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}+C\cdot\sum_{n=1}^{N}\xi_n+\sum_{n=1}^{N}\alpha_n\cdot(1-\xi_n-y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n+b))+\sum_{n=1}^{N}\beta_n\cdot(-\xi_n)\right)$$

解这个问题可以用第18讲中相同的技巧：最优的$\boldsymbol{\xi}$必然满足$\frac{\partial\mathcal{L}}{\partial\xi_n}=0$，因此可以先解这个关系式

$$\frac{\partial\mathcal{L}}{\partial\xi_n}=0\Rightarrow C-\alpha_n-\beta_n=0$$

因此可以把β_n换掉，即$\beta_n=C-\alpha_n$，并施加一个写明的约束条件$0\leq\alpha_n\leq C$。将β_n的这个表达式代入到上面的对偶问题，可以消掉β_n和ξ_n，问题转化为

$$\max_{0\leq\alpha_n\leq C,\beta_n=C-\alpha_n}\left(\min_{b,\mathbf{w}}\frac{1}{2}\mathbf{w}^{\mathsf{T}}\mathbf{w}+\sum_{n=1}^{N}\alpha_n\cdot(1-y_n(\mathbf{w}^{\mathsf{T}}\mathbf{z}_n+b))\right)$$

这个问题和第18讲中推出的对偶问题非常相近，只是最外部α_n的条件有细微差别。利用同样的方法可以推出

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$
$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n$$

因此，软间隔SVM对偶问题的标准形式为

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \alpha_n \alpha_m y_n y_m \mathbf{z}_n^{\mathsf{T}} \mathbf{z}_m - \sum_{n=1}^{N} \alpha_n \\ \text{subject to} \quad & \sum_{n=1}^{N} y_n \alpha_n = 0; \\ & 0 \leq \alpha_n \leq C, \text{for } n = 1, 2, \ldots, N \end{aligned}$$

与前面硬间隔SVM对偶问题的标准形式的唯一区别就是这里α_n有了上限C。这也是一个标准的二次规划问题，有N个变量，$2N+1$个限制条件

软间隔SVM传递的信息

得到软间隔SVM对偶问题的标准形式以后，求解这个问题的步骤与第18讲硬间隔SVM对偶问题的求解步骤基本类似（如果引入核方法，那么就把在高维空间算内积的步骤用核函数代替）。由于C和ξ_n的存在，软间隔SVM不再像硬间隔SVM那样数据必须线性可分时才能有解，而是总有解的。因此软间隔SVM是一种更灵活的模型，在实践中是最常用的SVM模型

接下来的问题是软间隔SVM如何求解b，因为这两个毕竟是不同的QP问题，KKT条件不完全相同。对于软间隔SVM，其互补松弛条件（complementary slackness）为

$$\begin{aligned} \alpha_n(1 - \xi_n - y_n(\mathbf{w}^{\mathsf{T}} \mathbf{z}_n + b)) &= 0 \\ \beta_n(-\xi_n) &= 0 \end{aligned}$$

将前面推出的β_n的关系式代入，有

$$\begin{aligned} \alpha_n(1 - \xi_n - y_n(\mathbf{w}^{\mathsf{T}} \mathbf{z}_n + b)) &= 0 \\ (C - \alpha_n)\xi_n &= 0 \end{aligned}$$

假设$\alpha_s > 0$，即s是一个支持向量，由第一个式子，可以求出

$$b = y_s - y_s \xi_s - \mathbf{w}^{\mathsf{T}} \mathbf{z}_s$$

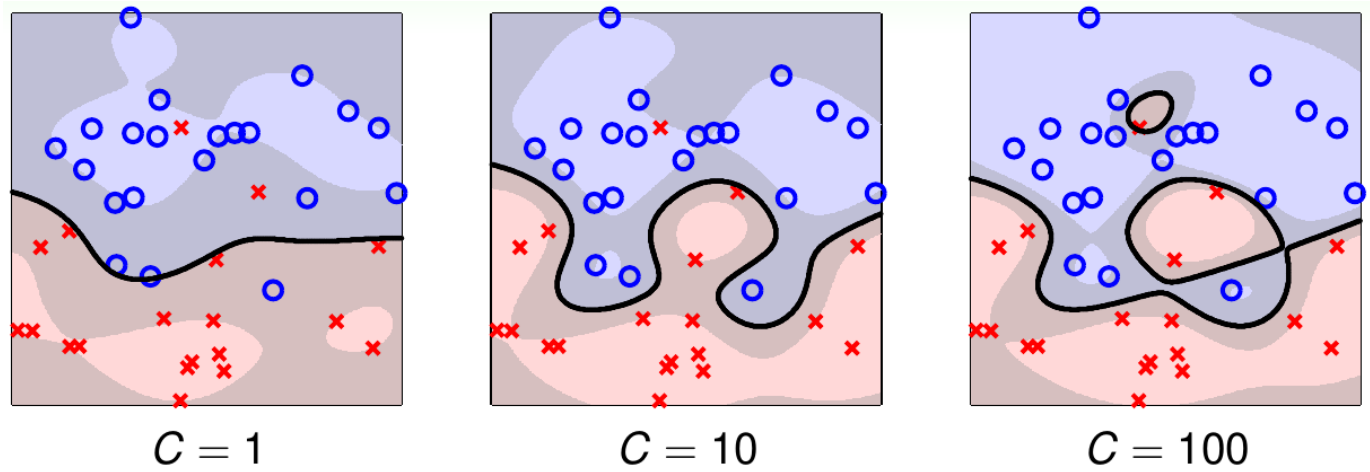
（注意这里用到了y_s的一个特点：因为$y_s \in \{-1, +1\}$，因此任意数除以y_s与乘以y_s的效果是一样的）

这里带了一个惩罚项，如果要算出ξ_s，就得先算出b，造成了一个鸡生蛋蛋生鸡的问题。因此要看第二个式子，如果$\alpha_s < C$，那么就会有$\xi_s = 0$。这种$0 < \alpha_s < C$的支持向量称为自由的支持向量。综上所述，使用任意*自由的*支持向量$(\mathbf{x}_s, y_s)$，都可以求出唯一的b为

$$b = y_s - \sum_{\text{SV indices } n} \alpha_n y_n K(\mathbf{x}_n, \mathbf{x}_s)$$

当然，极少数的情况下，没有自由的支持向量，那么b就只能通过一系列不等式得出，此时可能有不止一个b存在

软间隔SVM（使用高斯核）的例子如下图所示。可以看到，小的C可能会有欠拟合，而大的C仍然会导致过拟合的现象。因此，对软间隔SVM，仍然要小心地选择(γ, C)

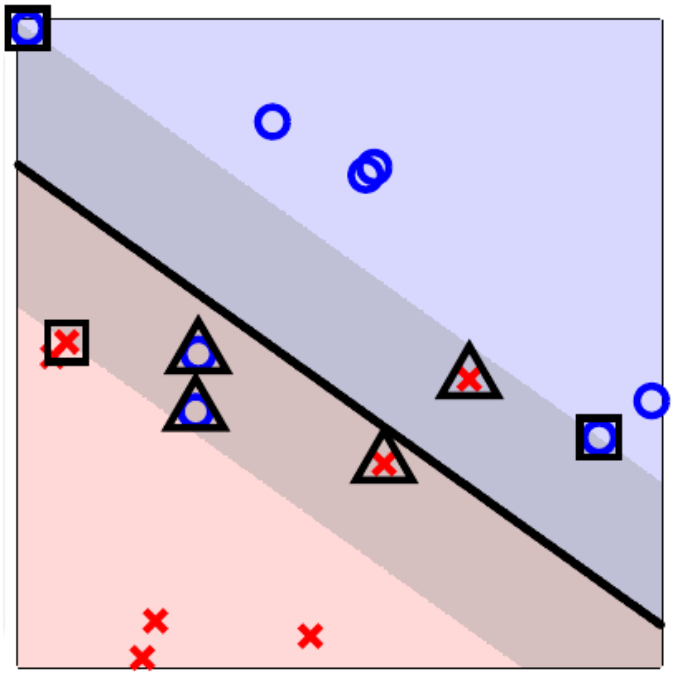


使用高斯核的软间隔SVM在设置不同的C时的效果

由前面的互补松弛条件，可以看到不同的数据点对应的α_n可以分为三种

- 非支持向量，$\alpha_n = 0, \xi_n = 0$。这种点被正确分类，且远离分离超平面的间隔边界
- 自由支持向量，$0 < \alpha_n < C, \xi_n = 0$。这种点被正确分类，骑在分离超平面的间隔边界上，决定b值
- 受限的支持向量，$\alpha_n = C$。这种点的$\xi_n = 1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b)$，肯定越过了间隔边界。这种店如果要分还能分成两种：一种是跨过了间隔边界，但是没跨过分离超平面；还一种是已经跨过了分离超平面

这三种点可见下图示意。其中自由支持向量以方框圈出，受限的支持向量以三角圈出



三种不同类型的支持向量

模型选择

前面说过，对高斯SVM（之后提到SVM都默认是软间隔SVM+核方法），不同的(C, γ)都会产生不同的分隔边界。到底怎样的组合才是最好的？还是要用交叉验证的方法来选择

这里着重观察交叉验证中最特殊的一种交叉验证：留一交叉验证（LOOCV）。由前面的讲解，可知在训练样本数为N的情况下，LOOCV可以看做是N折交叉验证。这里要证明的一点是：

> 对于SVM，有
>
> $$E_{\text{LOOCV}} \leq \frac{\#\text{SV}}{N}$$

证明如下：

假设对$(\mathbf{x}_N, y_N)$，解出来最优的$\alpha_N = 0$，则该点不是支持向量。因此，如果把这个点从原始数据集中除去，剩下的$(\alpha_1, \alpha_2, \ldots, \alpha_{N-1})$仍然是剩下的数据集按照算法求出来的最优的$\boldsymbol{\alpha}$。也就是说，在LOOCV时如果去掉的是非支持向量点，那么得到的g^-和不去掉这个点得到的g是一样的。考虑到非支持向量点肯定是被正确分类，且远离边界的点，因此有

$$\begin{aligned} e_{\text{non-SV}} &= \text{err}(g^-, \text{non-SV}) \\ &= \text{err}(g, \text{non-SV}) = 0 \end{aligned}$$

对支持向量，有$e_{\text{SV}} \leq 1$

将这些式子都加起来求平均，可知

$$E_{\text{LOOCV}} \leq \frac{\#\text{SV}}{N}$$

■

这意味着似乎可以通过支持向量的数量来判断SVM算法是否过拟合。如果一个算法的支持向量数比较少，那么它过拟合的风险可能就比较小。但是实际应用时，有这么几点需要注意

- $\#\text{SV}(C, \gamma)$是(C, γ)一个**不平滑**的函数，因此很难优化
- $\#\text{SV}$只是E_{LOOCV}的上界，并非确界

因此，如果在调参时太耗时间，可以先干掉一些支持向量数太多的模型，然后再去仔细验证剩下的模型

NTUML 21. 核Logistic回归

软间隔SVM与正则化模型

前面讲到，软间隔SVM对每个数据点记录了其“间隔破坏量”，记为ξ_n，并对该量施加一些惩罚，得到软间隔SVM的优化问题

$$\begin{aligned} \min_{b,\mathbf{w}} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n \\ \text{s.t.} \quad & y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b) \geq 1 - \xi_n,\ \xi_n \geq 0 \forall n \end{aligned}$$

这个问题可以使用另一种形式来表示，看看它到底要解决什么事情：给定b和$\mathbf{w}$，则ξ_n是怎么算出来的呢？ξ_n的本质是看这个点对间隔边界的破坏程度，因此对任意一个点，ξ_n有两种情况

- 这个点的确破坏了间隔边界，因此ξ_n记录它离想要的数字1有多远，此时$\xi_n = 1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b)$。由于这个点已经破坏了边界，因此$\xi_n \geq 0$
- 这个点没有破坏间隔边界，此时$\xi_n = 0$

综上所述，ξ_n可以用一个式子来表示，即$\xi_n = \max(1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b), 0)$。这也意味着软间隔SVM问题可以写成一个没有限制条件的优化问题，为

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C \cdot \sum_{n=1}^{N} \max(1 - y_n(\mathbf{w}^\mathsf{T}\mathbf{z}_n + b), 0)$$

此时ξ_n不再是一个变量，而是一个由b和$\mathbf{w}$算出来的结果

这种表示形式和之前讲过的正则化方法非常像：优化问题本身可以分解为两个部分，第一项是$\mathbf{w}$的长度，代表了模型的复杂度；而第二项可以看做是样本中所有误差的和，这不过这里似乎使用了一种新的误差函数，可以记为$\widehat{\text{err}}$。但是为什么不能直接解这个问题呢？首先，它不再是一个二次规划问题，因此对偶和核技巧可能不太好用上；其次，$\max(\cdot, 0)$并不是处处可微，因此这个优化问题更难解

下表给出了硬/软间隔SVM和正则化模型之间的一些比较

模型	最小化的目标函数	限制条件
带限制条件的正则化	E_{in}	$\mathbf{w}^\mathsf{T}\mathbf{w} \leq C$

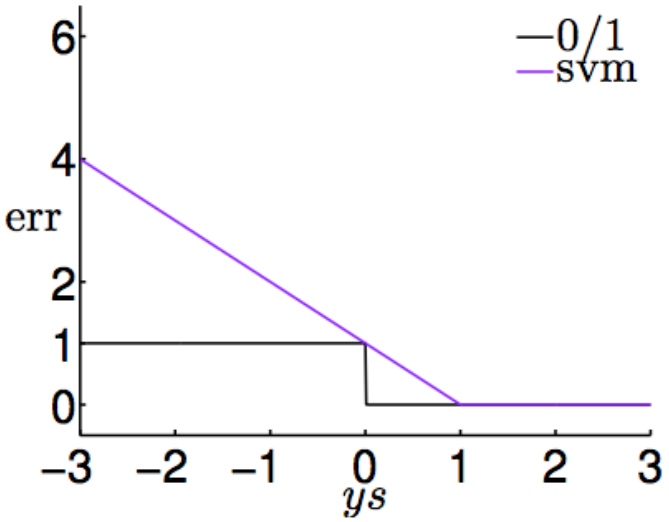
模型	最小化的目标函数	限制条件
硬间隔SVM	$\mathbf{w}^\mathsf{T}\mathbf{w}$	$E_{\text{in}}=0$（及其他）
L2正则化	$\frac{\lambda}{N}\mathbf{w}^\mathsf{T}\mathbf{w}+E_{\text{in}}$	
软间隔SVM	$\frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w}+CN\widehat{E_{\text{in}}}$	

即软间隔SVM和L2正则化这两个问题其实非常相似。从上表可知，大间隔其实就是正则化的一种实现，代表可选择的超平面要更少，实际上就是对$\mathbf{w}$的一种L2正则化；而软间隔则意味着要计算一种特殊的误差函数$\widehat{\text{err}}$。对于C来讲，无论是带限制条件的正则化问题还是软间隔SVM问题，C越大，对应着越小的λ，即越小的正则化

把SVM看作是一种正则化模型的好处是，可以将其理论延伸到其他模型，并与这些模型建立联系

SVM vs. Logistic回归

类似于第11讲中的做法，也可以把上面推出的SVM的损失函数$\widehat{\text{err}}_{\text{SVM}}$做出图像，与原始0/1误差函数$\text{err}_{0/1}(s,y)=[\![ys\neq 1]\!]$作比较。下图给出了这两个函数的图像



SVM的损失函数与0/1损失函数比较

由上图可知，$\widehat{\text{err}}_{\text{SVM}}$是$\text{err}_{0/1}$的上界，因此可以用这个上界推导出一些算法，这些算法间接地降低0/1误差，来把0/1分类问题做好。在文献中，SVM使用的这个误差函数通常被称作hinge loss（一些人翻译成“铰链损失函数”）。注意hinge loss不仅是0/1误差函数的上界，还是一个凸的上界，因此在优化时有些好的性质可以利用

接下来再把hinge loss和Logistic回归用的损失函数（经过缩放以后，记为SCE）$\log_2(1+\exp(-ys))$做一个比较。当ys趋近于$+\infty$时，$\widehat{\text{err}}_{\text{SVM}}(s,y)=0$；当$ys$趋近于$-\infty$时，$\widehat{\text{err}}_{\text{SVM}}=1-ys\approx ys$。而SCE也有类似的结果（只不过是$ys\to+\infty$时，$\text{SCE}\approx 0$。因此可以说，SVM可以看做是带有L2正则化的Logistic回归

下表对目前讲解过的三种用于0/1分类的模型做了比较

模型名称	基本思想	优点	缺点
PLA	直接最小化$\text{err}_{0/1}$	数据集线性可分时算法效率很高	只能用于数据集线性可分的情况，否则就要用口袋算法
带有正则化的Logistic回归	使用GD/SGD最小化正则化的err_{SCE}	易于优化，正则化使得模型不易过拟合	ys很小时（是很大的负数时）误差函数是0/1误差函数太松的上界
软间隔SVM	使用QP最小化正则化的$\widehat{\text{err}}_{\text{SVM}}$	易于优化，有理论保证	ys很小时（是很大的负数时）误差函数是0/1误差函数太松的上界

从这个角度来看，带有正则项的Logistic回归其实几乎就是在做SVM。那么当解SVM时，可否将这个解用在Logistic回归中（即估计每个点是正例的概率）？

用作软二元分类的SVM

那么怎么将SVM用在软二元分类问题中呢？一种比较直观的方法是，先使用软间隔SVM算法得到$(b_{\text{SVM}}, \mathbf{w}_{\text{SVM}})$，然后使用Logistic函数计算$\theta(\mathbf{w}_{\text{SVM}}^{\mathsf{T}}\mathbf{x} + b_{\text{SVM}})$，将该结果作为$g(\mathbf{x})$的结果返回。这种做法在实际应用中效果还不错，但是失去了之前推导Logistic回归时候用到的最大似然等等思想。另一种方法是，先用SVM算法得到$(b_{\text{SVM}}, \mathbf{w}_{\text{SVM}})$，将这个结果作为Logistic回归的初始权重，然后使用Logistic回归算法得到最后的$g(\mathbf{x})$。这种方法最后得到的结果跟直接用Logistic回归算法跑出的模型应该差不多，丢失了SVM自己的一些特性，例如核方法等等。因此，这两种简单的做法都不尽如人意，需要想一种办法融合这两个算法的优势

SVM得到的结果如果用$\mathbf{w}_{\text{SVM}}^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x}) + b_{\text{SVM}}$表示，其本质实际上就是一个分数。接着，对这个分数做缩放和平移，即返回$g(\mathbf{x}) = \theta(A \cdot (\mathbf{w}_{\text{SVM}}^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x}) + b_{\text{SVM}}) + B)$，通过用Logistic回归训练$A$和$B$这两个参数来达到使用MLE的目的和效果。由于这个算法的核心还是SVM，因此对偶和核技巧都可以使用。从几何的角度来看，SVM的用处是找到法向量，Logistic回归是使用MLE进行微调。如果$\mathbf{w}_{\text{SVM}}$比较好，那么$A > 0, B \approx 0$

因此，新的Logistic回归问题为

$$\min_{A,B} \quad \frac{1}{N}\sum_{n=1}^{N}\log\left(1 + \exp\left(-y_n(A \cdot (\underbrace{\mathbf{w}_{\text{SVM}}^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x}_n) + b_{\text{SVM}}}_{\Phi_{\text{SVM}}(\mathbf{x}_n)}) + B)\right)\right)$$

该问题可以看做是一个两阶段学习问题：先把数据用SVM做一个变换，再用Logistic回归学习两个参数。这个方法称为**Platt模型**。这种模型，这里称为软二元分类器，的边界和第一步得到的SVM分类器的边界是不太一样的，因为B的存在会使边界稍微有所移动

这里的做法是先把数据用核方法变换到空间$\mathcal{Z}$里，然后用Logistic回归得到一个模型。这种方法并没有直接在$\mathcal{Z}$空间里求解Logistic回归，因此并不是该空间里最好的Logistic回归的解。有没有办法真的在这个空间里找到Logistic回归的最优解呢？

核Logistic回归

要使用核方法做Logistic回归，其根本问题是Logistic回归根本就不是一个二次规划问题！但是之前的核方法为什么能管用呢？其基本思想就是把$\mathcal{Z}$空间做的内积转化为在$\mathcal{X}$空间可以轻易计算的函数。之前提到过，在求解核SVM问题时，不仅在求解$\mathbf{w}_*$时要用到$\mathcal{Z}$空间的内积，而在得到最优的g以后真正预测时，也需要用到$\mathcal{Z}$空间的内积。如果最优的$\mathbf{w}_*$可以表达为一堆已经看过的$\mathbf{z}$的线性组合，那么才能把最后计算$\mathbf{w}_*^{\mathsf{T}}\mathbf{z}$的过程表达为核函数计算的形式。而核SVM能做到这一点，因此核SVM最后有$\mathbf{w}_*^{\mathsf{T}}\mathbf{z} = \sum_{n=1}^{N}\beta_n\mathbf{z}_n^{\mathsf{T}}\mathbf{z} = \sum_{n=1}^{N}\beta_n K(\mathbf{x}_n, \mathbf{x})$。因此，**最优解$\mathbf{w}_*$是一些$\mathbf{z}_n$的线性组合，是能用核方法的关键条件**

之前讲过的SVM，其最优权重$\mathbf{w}_{\text{SVM}}$可以写为$\mathbf{w}_* = \sum_{n=1}^{N}(\alpha_n y_n)\mathbf{z}_n$这种形式，此时$\alpha_n$来自于对偶问题的解。PLA的最优权重实际上也可以写为这种形式，此时α_n来自于对错分样本的修正。Logistic回归的最优权重也有这样的性质，此时系数由做SGD过程中总共的移动步数决定。换句话说，这三种算法中$\mathbf{w}_*$都可以被$\mathbf{z}_n$**表示**

推而广之，可以先讨论一个更高级的问题：什么时候$\mathbf{w}_*$可以被$\mathbf{z}_n$表示？这里可以证明：

> 对任何带有L2正则化的线性模型
>
> $$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^{\mathsf{T}}\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N}\text{err}(y_n, \mathbf{w}^{\mathsf{T}}\mathbf{z}_n)$$
>
> 其最优的$\mathbf{w}_*$都可以被表示为$\mathbf{z}_n$的线性组合，即$\mathbf{w}_* = \sum_{n=1}^{N}\beta_n\mathbf{z}_n$。

这个定理称为**表示定理**。证明如下

将最优的$\mathbf{w}_*$分解为两个向量$\mathbf{w}_{||}$和$\mathbf{w}_{\perp}$，记由$\mathbf{z}_n$张成的空间为$\text{span}(\mathbf{z}_n)$，$\mathbf{w}_{||} \in \text{span}(\mathbf{z}_n), \mathbf{w}_{\perp} \perp \text{span}(\mathbf{z}_n)$，即$\mathbf{w}_* = \mathbf{w}_{||} + \mathbf{w}_{\perp}$。则问题转化为证明在这种分解下有$\mathbf{w}_{\perp} = \mathbf{0}$。假设$\mathbf{w}_{\perp} \neq \mathbf{0}$，则由于正交向量的内积为0，有$\text{err}(y_n, \mathbf{w}_*^{\mathsf{T}}\mathbf{z}_n) = \text{err}(y_n, (\mathbf{w}_{||} + \mathbf{w}_{\perp})^{\mathsf{T}}\mathbf{z}_n) = \text{err}(y_n, \mathbf{w}_{||}^{\mathsf{T}}\mathbf{z}_n)$，因此$\mathbf{w}_*$和$\mathbf{w}_{||}$有相同的err。又$\mathbf{w}_*^{\mathsf{T}}\mathbf{w}_* = \mathbf{w}_{||}^{\mathsf{T}}\mathbf{w}_{||} + 2\mathbf{w}_{||}^{\mathsf{T}}\mathbf{w}_{\perp} + \mathbf{w}_{\perp}^{\mathsf{T}}\mathbf{w}_{\perp} = \mathbf{w}_{||}^{\mathsf{T}}\mathbf{w}_{||} + \mathbf{w}_{\perp}^{\mathsf{T}}\mathbf{w}_{\perp} > \mathbf{w}_{||}^{\mathsf{T}}\mathbf{w}_{||}$。这意味着$\mathbf{w}_{||}$比最优解$\mathbf{w}_*$还要使目标函数更小，矛盾！因此肯定有$\mathbf{w}_{\perp} = \mathbf{0}$，最优解可以用$\mathbf{z}_n$的线性组合表示，得证 ■

表示定理表明，任何L2正则化的线性模型都可以核化。那么接下来的问题是怎么核化带有L2正则项的Logistic回归。考虑原先问题的定义

$$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^{\mathsf{T}}\mathbf{w}+\frac{1}{N}\sum_{n=1}^{N}\log(1+\exp(-y_n\mathbf{w}^{\mathsf{T}}\mathbf{z}_n))$$

由表示定理，最优的$\mathbf{w}_*=\sum_{n=1}^{N}\beta_n\mathbf{z}_n$。将该式代入原始问题，可以将原来关于$\mathbf{w}$的问题转化成关于$\boldsymbol{\beta}$的问题，即

$$\min_{\boldsymbol{\beta}}\frac{\lambda}{N}\sum_{n=1}^{N}\sum_{m=1}^{N}\beta_n\beta_m K(\mathbf{x}_n,\mathbf{x}_m)+\frac{1}{N}\sum_{n=1}^{N}\log\left(1+\exp\left(-y_n\sum_{m=1}^{N}\beta_m K(\mathbf{x}_m,\mathbf{x}_n)\right)\right)$$

这是一个没有条件的最优化问题，因此GD或者SGD方法仍然可以使用

记上面这个问题为KLR问题（Kernel Logistic Regression），可以从另一个角度理解：由前面的讲解，可知核函数从某种意义上也是相似度的一种体现，那么最后一项$\sum_{m=1}^{N}\beta_m K(\mathbf{x}_m,\mathbf{x}_n)$可以看做是先把所有数据点$\mathbf{x}_1,\dots,\mathbf{x}_N$与$\mathbf{x}_n$的相似度求出来，将这个变换后的数据$K(\mathbf{x}_1,\mathbf{x}_n),K(\mathbf{x}_2,\mathbf{x}_n),\dots,K(\mathbf{x}_N,\mathbf{x}_n)$组成向量与变量$\boldsymbol{\beta}$求内积，这其实也是一个小的线性模型。而前面的项$\sum_{n=1}^{N}\sum_{m=1}^{N}\beta_n\beta_m K(\mathbf{x}_n,\mathbf{x}_m)$如果写成矩阵形式就是$\boldsymbol{\beta}^{\mathsf{T}}\mathrm{K}\boldsymbol{\beta}$，其本质实际上是一个正则化。即KLR可以看做是一个关于$\boldsymbol{\beta}$的线性模型，其对应的$\mathbf{x}$是用核变换后得到的结果，而正则化也是使用核来做

需要注意的是，KLR与SVM最大的不同为，对KLR，其系数β_n通常都不为0

NTUML 22. 支持向量回归（SVR）

核岭回归

由上一讲提到的表示定理可知，任何带有L2正则项的线性模型都能被核化。那么如何把回归模型核化呢？而且回想之前讲岭回归（带有L2正则项的线性回归）时，曾经说过该模型可以得到一个闭合的解析解，那么使用了核方法以后的模型是否能同样有解析解？

首先，将原始岭回归的问题写出

$$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^{\mathsf{T}}\mathbf{w}+\frac{1}{N}\sum_{n=1}^{N}(y_n-\mathbf{w}^{\mathsf{T}}\mathbf{z}_n)^2$$

要使该问题的最优解$\mathbf{w}_*$可以写作若干个$\mathbf{z}_n$的线性组合，即$\mathbf{w}_*=\sum_{n=1}^{N}\beta_n\mathbf{z}_n$。类似于之前KLR的推导，可以把$\mathbf{w}_*$的表达式代回到原始问题，进而求解最优的$\boldsymbol{\beta}$。因此原问题可以化为

$$\min_{\boldsymbol{\beta}} \quad \frac{\lambda}{N}\sum_{n=1}^{N}\sum_{m=1}^{N}\beta_n\beta_m K(\mathbf{x}_n,\mathbf{x}_m)+\frac{1}{N}\sum_{n=1}^{N}\left(y_n-\sum_{m=1}^{N}\beta_m K(\mathbf{x}_n,\mathbf{x}_m)\right)^2$$

同样的，第一项可以看做是使用核矩阵K对$\boldsymbol{\beta}$做正则化，第二项是对N维经核变化过的样本做线性回归，权重为$\boldsymbol{\beta}$。经过矩阵/向量化，该问题可以进一步化简为

$$\min_{\boldsymbol{\beta}} \quad \frac{\lambda}{N}\boldsymbol{\beta}^{\mathsf{T}}\mathrm{K}\boldsymbol{\beta}+\frac{1}{N}(\boldsymbol{\beta}^{\mathsf{T}}\mathrm{K}^{\mathsf{T}}\mathrm{K}\boldsymbol{\beta}-2\boldsymbol{\beta}^{\mathsf{T}}\mathrm{K}^{\mathsf{T}}\mathbf{y}+\mathbf{y}^{\mathsf{T}}\mathbf{y})$$

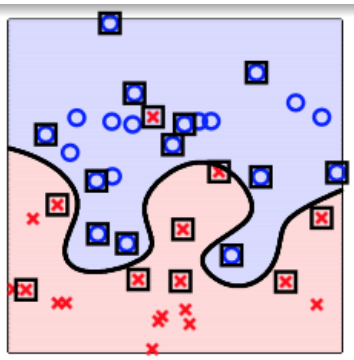
将上式对$\boldsymbol{\beta}$取梯度，同时注意K是对称矩阵（Mercer定理），可知$\mathrm{K}=\mathrm{K}^{\mathsf{T}}=\mathrm{K}^{\mathsf{T}}\mathrm{I}$。因此

$$\nabla E_{\text{aug}}(\boldsymbol{\beta})=\frac{2}{N}(\lambda\mathrm{K}^{\mathsf{T}}\mathrm{I}\boldsymbol{\beta}+\mathrm{K}^{\mathsf{T}}\mathrm{K}\boldsymbol{\beta}-\mathrm{K}^{\mathsf{T}}\mathbf{y})=\frac{2}{N}\mathrm{K}^{\mathsf{T}}((\lambda\mathrm{I}+\mathrm{K})\boldsymbol{\beta}-\mathbf{y})$$

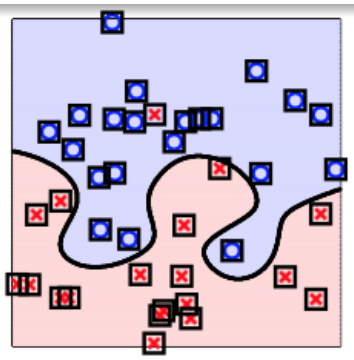
可知最后也可以得到$\boldsymbol{\beta}$的解析解为$\boldsymbol{\beta}=(\lambda\mathrm{I}+\mathrm{K})^{-1}\mathbf{y}$，返回的$g(\mathbf{x})=\sum_{n=1}^{N}\beta_n K(\mathbf{x}_n,\mathbf{x})$。由于$\mathrm{K}$是半正定矩阵且$\lambda>0$，因此$\lambda\mathrm{I}+\mathrm{K}$总是可逆的。求解该问题的时间复杂度为$O(N^3)$。相比而言，线性岭回归得到的结果$\mathbf{w}=(\lambda\mathrm{I}+\mathrm{X}^{\mathsf{T}}\mathrm{X})^{-1}\mathrm{X}^{\mathsf{T}}\mathbf{y}$更受限（如果不做多项式变换，只能得到线性模型），但是训练时间复杂度$O(d^3+d^2N)$，预测时间复杂度$O(d)$，在$N>>d$时效率更高。核岭回归尽管得到的模型更灵活，但是训练时间复杂度$O(N^3)$，预测时间复杂度$O(N)$，不适合处理大的数据集。这再次说明我们通常要在模型的灵活性和计算效率之间进行权衡

支持向量回归的原始问题

前面说到，线性回归（及其带正则化的变种）也可以用来解决分类问题。使用了核方法的核岭回归也可以解决同样的问题，此时这个模型被称为最小二乘支持向量机（Least-Squares SVM, LSSVM）。高斯LSSVM和软间隔高斯核SVM的对比可见下图



soft-margin Gaussian SVM

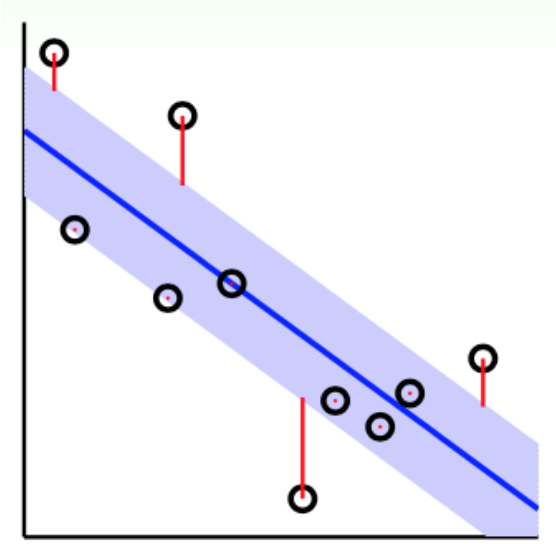


Gaussian LSSVM

软间隔高斯核SVM和高斯LSSVM的对比

可见两者的边界差别不是很大，但是对LSSVM，几乎所有点都是支持向量，而对软间隔SVM，只有少数点是支持向量。支持向量越多，预测的时间就越长，效率就越低。那么有没有办法得到这样一种模型：它既可以解决回归问题，又可以像标准SVM那样有稀疏的β呢?

首先考虑一个“管道回归”问题（tube regression）：之前讲线性回归的时候，对模型误差的判定非常严苛：对某个点$(\mathbf{x}_n, y_n)$，只有当它完全落在学习出的超平面上时，它的误差才是0。只要出现一点偏差，就要负责任。但是在管道回归问题里，错误评判的标准宽松了一些：在标线的基础上，拓展出来一部分“中立区”。即便是某些点没有落在超平面上，只要落在中立区里，误差就不计算了。对于落在中立区外的点，其误差计算也不是看它到超平面的距离，而是看它离中立区的距离。下图给出了一个示例



管道回归（tube regression）

上图中，蓝色线是学习出的超平面，蓝色带是划分出的“中立区”，红线表明了各点误差的大小。假设中立区的（单侧）高度为ϵ，管道回归的误差函数为

$$\mathrm{err}(y, s) = \max(0, |s - y| - \epsilon)$$

可以看出，这个误差函数（通常称为ϵ-不敏感误差）与前面讲到的SVM的hinge loss形式很像。而如果将此误差函数的图像做出来，可以发现当$|y - s|$比较小时，管道回归的误差与平方误差大致相等。$|y - s|$比较大时，管道回归的误差又远小于平方误差，这说明前者受离群点的影响更小

接下来要做的就是求解这个带有L2正则项的管道回归问题来得到一个稀疏的β。首先将问题描述出来

$$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^{\mathsf{T}}\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N}\max\left(0, |\mathbf{w}^{\mathsf{T}}\mathbf{z}_n - y| - \epsilon\right)$$

尽管这是一个无限制条件的优化问题，但是$\max$函数不是处处可微的，因此使用微分的方法会遇到问题。标准SVM也有类似的问题，但是重写为一个QP问题以后就好解了。此外，为了使用核方法，该问题可以用表示定理来转换为一个可核化的问题，但是可能不会有稀疏性的保证；而标准SVM由于有对偶问题和KKT条件，有稀疏性的保证。因此在求解此问题时，也会试着像QP、对偶和KKT条件靠拢。所以接下来仿照标准SVM问题对原问题进行重写

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{n=1}^{N}\max(0, |\mathbf{w}^\mathsf{T}\mathbf{z}_n + b - y_n| - \epsilon)$$

原始软间隔SVM问题对max函数的处理是引入了边界破坏量ξ_n。这里使用了类似的思想，但是由于绝对值符号的存在，只引入一个ξ_n是不够的，约束条件还不是一个线性问题。应该将绝对值符号展开，并引入两个量：向下惩罚项$\xi_n^\vee$和向上惩罚项$\xi_n^\wedge$。这样一来，就得到了标准支持向量回归（SVR）的原始问题，形式为

$$\begin{aligned}\min_{b,\mathbf{w},\boldsymbol{\xi}^\vee,\boldsymbol{\xi}^\wedge} \quad & \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{n=1}^{N}(\xi_n^\vee + \xi_n^\wedge) \\ \text{s. t.} \quad & -\epsilon - \xi_n^\vee \le y_n - \mathbf{w}^\mathsf{T}\mathbf{z}_n - b \le \epsilon + \xi_n^\wedge \\ & \xi_n^\vee \ge 0, \xi_n^\wedge \ge 0\end{aligned}$$

这里各参数的意义为

- C在正则的程度和对管道的破坏程度中做均衡
- ϵ控制管道在竖直方向上的宽度（比原来SVM多出来一个超参数）

这个QP问题有$\tilde{d} + 1 + 2N$个变量，$2N + 2N$个限制条件

支持向量回归的对偶问题

要写出上述原始问题的对偶问题，第一步是引入拉格朗日乘子。记约束条件$y_n - \mathbf{w}^\mathsf{T}\mathbf{z}_n - b \le \epsilon + \xi_n^\wedge$对应的拉格朗日乘子为$\alpha_n^\wedge$，约束条件$-\epsilon - \xi_n^\vee \le y_n - \mathbf{w}^\mathsf{T}\mathbf{z}_n - b$对应的拉格朗日乘子为$\alpha_n^\vee$。经过类似第18讲和20讲的一些计算，可以得到以下一些KKT条件

$$\frac{\partial \mathcal{L}}{\partial w_i} = 0 \Rightarrow \mathbf{w} = \sum_{n=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee)\mathbf{z}_n$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee) = 0$$

以及其中的互补松弛条件为

$$\begin{cases}\alpha_n^\wedge(\epsilon + \xi_n^\wedge - y_n + \mathbf{w}^\mathsf{T}\mathbf{z}_n + b) = 0 \\ \alpha_n^\vee(\epsilon + \xi_n^\vee + y_n - \mathbf{w}^\mathsf{T}\mathbf{z}_n - b) = 0\end{cases}$$

最终可以得到对偶问题为

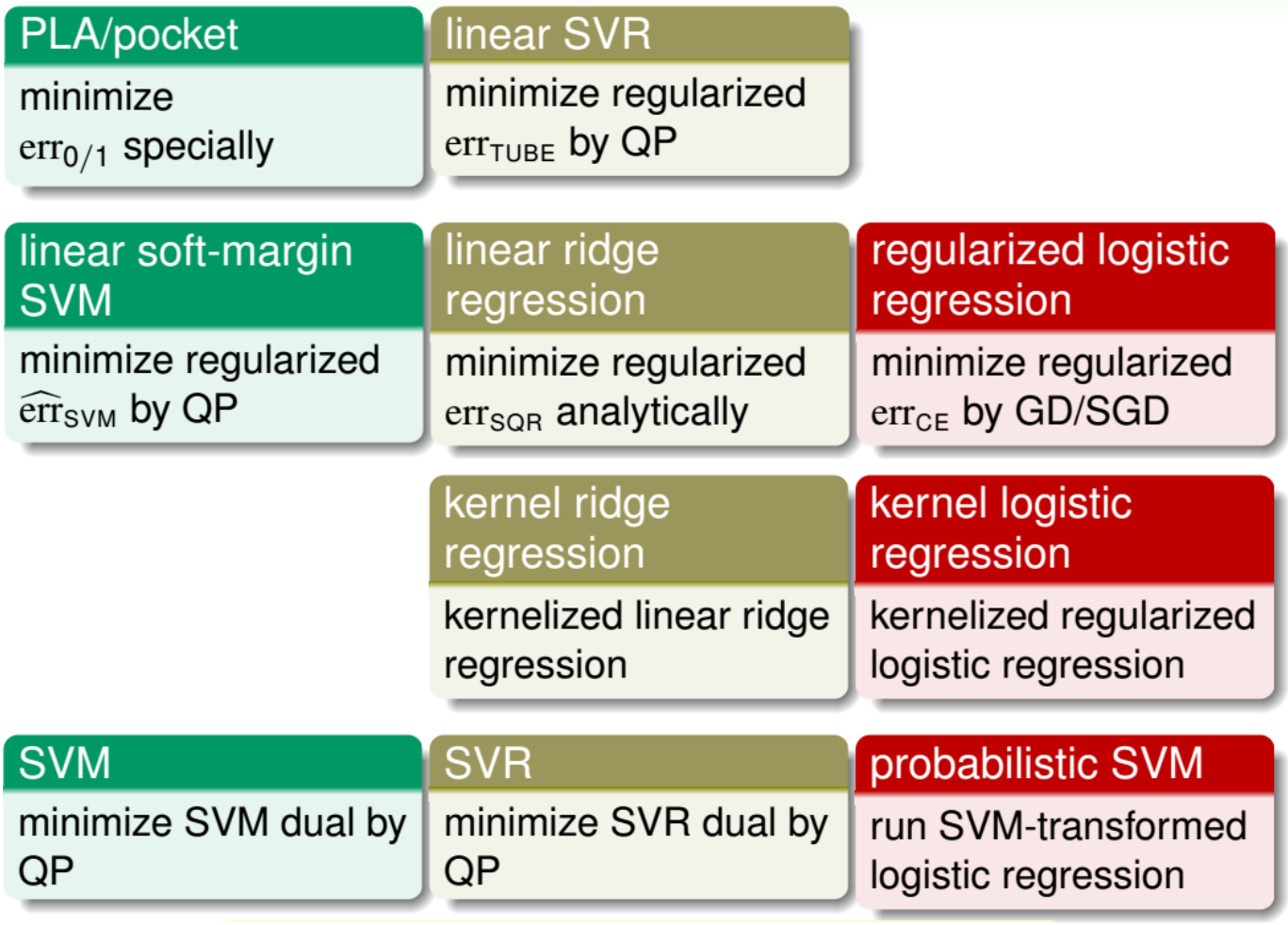
$$\begin{aligned}\min \quad & \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee)(\alpha_m^\wedge - \alpha_m^\vee)K(\mathbf{z}_n, \mathbf{z}_m) + \sum_{n=1}^{N}((\epsilon - y_n)\cdot\alpha_n^\wedge + (\epsilon + y_n)\cdot\alpha_n^\vee) \\ \text{s. t.} \quad & \sum_{n=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee) = 0 \\ & 0 \le \alpha_n^\wedge \le C, 0 \le \alpha_n^\vee \le C\end{aligned}$$

这个问题也可以用二次规划求解器求解

最后，由前面的KKT条件可知$\mathbf{w} = \sum_{n=1}^{N}(\alpha_n^\wedge - \alpha_n^\vee)\mathbf{z}_n$。记$\alpha_n^\wedge - \alpha_n^\vee$为$\beta_n$，要证明的就是$\beta_n$组成的系数向量$\boldsymbol{\beta}$是稀疏的。由前面的互补松弛条件，对那些在管道中的点，其对应的$\xi_n^\wedge$和$\xi_n^\vee$都为0。由于此时有$|\mathbf{w}^\mathsf{T}\mathbf{z} + b - y_n| < \epsilon$，因此互补松弛条件中$\epsilon + \xi_n^\wedge - y_n + \mathbf{w}^\mathsf{T}\mathbf{z}_n + b$和$\epsilon + \xi_n^\vee + y_n - \mathbf{w}^\mathsf{T}\mathbf{z}_n - b$都不为0，也就是它们对应的$\alpha_n^\wedge$和$\alpha_n^\vee$都为0，$\beta_n$也就为0。所以可以得出结论：落在管道边界上或管道外的数据点才对$\mathbf{w}$有贡献，它们才是支持向量，$\boldsymbol{\beta}$是稀疏的

核模型总结

最后对前面讲过的所有线性模型和使用了核技巧的模型（简称核模型）做一总结，参见下图



线性模型与核模型总结

上图中，

- 前两行均为线性模型，其中
  - 第一行较少使用，因为性能/效果比较差
  - 第二行是经典机器学习包liblinear的主力
- 后两行均为核模型，其中
  - 第三行较少使用，因为支持向量是稠密的
  - 第四行是经典机器学习包libsvm的主力

核模型中可用的核包括多项式核、高斯核，或者是任何满足Mercer条件的自定义核。它们是线性模型的有力扩展。但是“能力越大责任越大”，使用这些强力模型时，要更小心地调参以及做验证，避免过拟合现象的出现

NTUML 23. 模型混合与装袋（bagging）

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/10/05/NTUML-23-Blending-Bagging/

模型聚合的动机

如果已经得到了一些特征或假设，这些假设可以做预测，那么如何将这些有预测性的特征或假设合起来，让它们变得更好？这种模型就称为模型聚合

为什么要做模型聚合呢？举个例子，假设今天有15个朋友，每个人都会对股市的涨跌做一个预测，那么在得到这些预测以后应该怎么办？其实可以做的事情非常多，例如

- 根据过往的情况，选择最可靠的朋友，听ta的意见（类似于机器学习中的验证方法）
- 让朋友们一人一票，看投票结果
- 让朋友们投票，其中比较可靠的给的票数多（非平等投票）
- 根据每个人的特长，在不同的情况下听从不同人群的意见

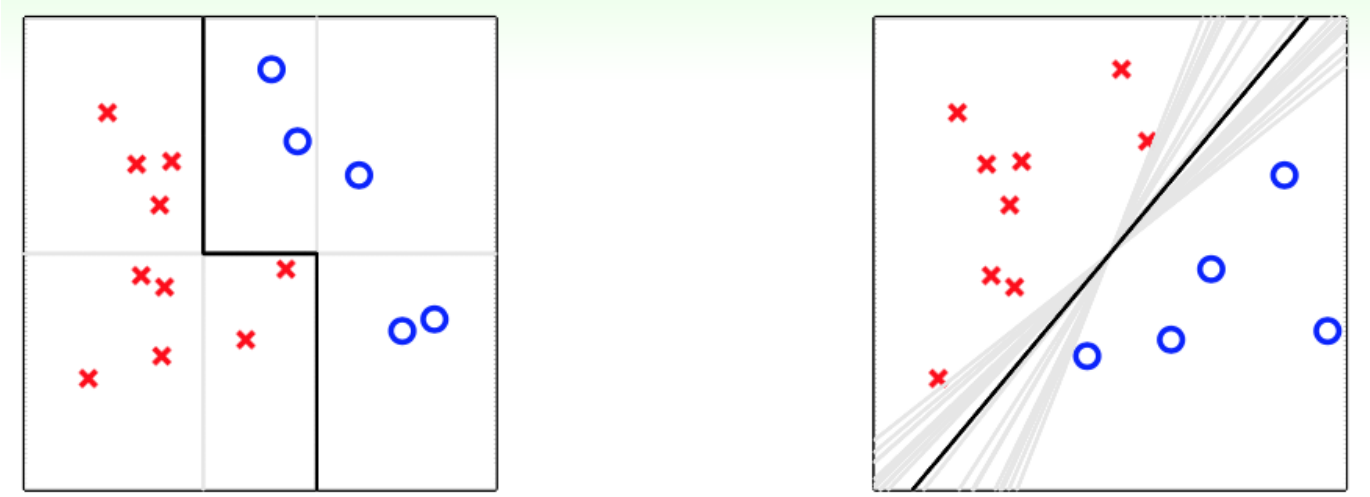
以上做法都可以对应于机器学习中的某种算法。总的来说，将这些人意见融合起来的过程，就可以类比为模型聚合的算法过程。可以使用更加形式化的方法来描述上述做法。假设有T个朋友对应于T个假设函数$g_1,\dots,g_T$，每个朋友t做出的预测为$g_t(\mathbf{x})$，则（假设每个人只预测涨跌，涨记为$+1$，跌记为-1）

- 验证法的形式化表示为$G(\mathbf{x})=g_{t*}(\mathbf{x})\text{ with }t_*=\arg\min_{t\in\{1,2,\dots,T\}}E_{\text{val}}(g_t^-)$
- 平等投票法的形式化表示为$G(\mathbf{x})=\text{sign}\left(\sum_{t=1}^T 1\cdot g_t(\mathbf{x})\right)$，这里每个$g_t(\mathbf{x})$前面乘的1就是每个人的权重
- 非平等投票法的形式化表示为$G(\mathbf{x})=\text{sign}\left(\sum_{t=1}^T \alpha_t\cdot g_t(\mathbf{x})\right),\alpha_t\geq 0$。注意这种方式包含了前面提到的验证法和平等投票法。当$\alpha_t=1$时，该方法退化为平等投票法；当$\alpha_t=[\![E_{\text{val}}(g_t^-)\text{ smallest}]\!]$时，该方法退化为验证法
- 根据特长选择法的形式化表示为$G(\mathbf{x})=\text{sign}\left(\sum_{t=1}^T q_t(\mathbf{x})\cdot g_t(\mathbf{x})\right),q_t(\mathbf{x})\geq 0$。这里$q_t(\mathbf{x})$实际上是根据输入对$t$做判断给ta几票的函数。当$q_t(\mathbf{x})=\alpha_t$时，该方法退化为非平等投票法

可见，聚合模型是一个很大的模型族

在正式介绍聚合模型法之前，有必要先从验证法入手，分析一下验证法与（其它）聚合模型法之间的不同。回忆之前讲过的验证法的过程，不难发现验证法最后返回的g_t^-是**一个**很强的模型，因为使用验证的目的就是让被选出的模型有足够小的E_{out}，对该模型的评估也是要求其有足够小的E_{val}。再次强调，验证法最后只返回一个模型。而聚合模型的出发点是，如果我们手里有**多个**假设模型，是否可以做得更好（尽管这些模型可能每个单拿出来效果都不是太好）？

为什么模型聚合以后的效果会比较好？可以看下面两个例子



模型聚合的例子

上图左描述了这样一个问题：对于图示中的数据集，只允许用水平和竖直的线做分类器，将数据集完全分开（但是可以对分类器产生的结果进行组合）。可以看到，图中选择的两条竖直线和一条水平线各自都没有将数据集完全分开的能力，但是它们组合（投票）的结果却可以完成目标。这表明若干弱分类器聚合的结果可能会突破个体的能力限制，完成某种类似于“特征变换”的功能

上图右描述了另一个问题：对于图示中的数据集，PLA算法可能会产生无限多条线将样本完全分开（因为PLA算法有随机性）。这些线有的可能会偏向红方，有的可能会偏向蓝方，但是它们组合（投票）的结果最后可能得到的是图中的黑色实线，这条实线比较中庸平均，离两边都不近——而这条线很有点像使用SVM算法得到的分类器。这说明从另一方面，模型经过聚合以后可能可以达到正则化的目的

在之前的课程里，以开车为例，特征变换使模型变得强大，有点像踩油门；而正则化使模型不要过分复杂，有点像踩刹车。从上面的例子中，可以看到这两个效果都可以通过模型聚合达到，因此可以初步判断说，恰当的聚合可以使模型的效果更好

均匀混合

均匀混合法，对应于之前提到的平等投票法。其原理是对于若干个已经知道了形式的假设函数g_t（这里首先考虑二元分类问题，因此假设每个函数返回都是+1或者-1），给它们分配相等的权重，对分类的结果进行统计，即

$$G(\mathbf{x})=\text{sign}\left(\sum_{t=1}^T 1\cdot g_t(\mathbf{x})\right)$$

如果所有g_t返回的结果都一样，那么跟使用其中任何一个g_t也没什么差别。但是如果每个g_t返回的结果很不一样，那么大多数模型的结果可以对小部分模型的异端结果做出纠正（少数服从多数）——当然，如果要解决的是多元分类问题，可以采用类似的思想。不过此时具体$G(\mathbf{x})$的形式稍有不同，为

$$G(\mathbf{x}) = \arg\max_{1\le k\le K} \sum_{t=1}^{T} [\![g_t(\mathbf{x}) = k]\!]$$

对于回归问题，均匀混合法应如何做？直观（且的确是可行）的方法就是把所有分类器预测的结果做平均，即

$$G(\mathbf{x}) = \frac{1}{T}\sum_{t=1}^{T} g_t(\mathbf{x})$$

假设g_t返回的结果很不一样，那么很可能出现的结果是一部分$g_t(\mathbf{x}) > f(\mathbf{x})$，另一部分$g_t(\mathbf{x}) < f(\mathbf{x})$。求均值的结果是将这些盈余和不足都抵消掉，最后的结果会比每个独立g_t预测的结果都更靠近f

综上所述，如果各个g_t的结果都很不一样，即使最后聚合模型的时候采用的是非常简单的均匀混合法，得到的结果也比任意一个单个假设返回的结果要好一些

对于回归问题，有一个更理论的证明，可以表明$G(\mathbf{x})$的结果肯定比单一模型的结果更好。从最简单的情况入手，先考虑对一个数据点$\mathbf{x}$的情况。为了记法上的方便，可以把$\mathbf{x}$都省略掉，即记为$G = \frac{1}{T}\sum_{t=1}^{T} g_t = \text{avg}(g_t)$。因此

$$\begin{aligned}
\text{avg}\left((g_t(\mathbf{x}) - f(\mathbf{x}))^2\right) &= \text{avg}(g_t^2 - 2g_t f + f^2) \\
&= \text{avg}(g_t^2) - 2Gf + f^2 \\
&= \text{avg}(g_t^2) - G^2 + (G-f)^2 \\
&= \text{avg}(g_t^2) - 2G^2 + G^2 + (G-f)^2 \\
&= \text{avg}(g_t^2 - 2g_t G + G^2) + (G-f)^2 \\
&= \text{avg}((g_t - G)^2) + (G-f)^2
\end{aligned}$$

上面讨论的是对一个数据点的情况。将其扩展到整个数据集上，有如下关系式

$$\begin{aligned}
\text{avg}(E_{\text{out}}(g_t)) &= \text{avg}\left(\mathcal{E}(g_t - G)^2\right) &&+E_{\text{out}}(G) \\
&\ge &&+E_{\text{out}}(G)
\end{aligned}$$

■

那么如何运用这个模型聚合的方法呢？可以想象一个虚拟的机器学习过程：对每个$t = 1, 2 \ldots, T$，假设有N条来自于某个分布P^N的数据，组成数据集$\mathcal{D}_t$（满足独立同分布性）。假设每个g_t都是用不同的N条数据根据算法$\mathcal{A}(\mathcal{D}_t)$学习得出，那么所有这些$g$的均值，记为$\bar{g}$，有

$$\bar{g} = \lim_{T\to\infty} G = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} g_t = \mathcal{E}_{\mathcal{D}}\mathcal{A}(\mathcal{D})$$

套用上面的结论，有

$$\text{avg}(E_{\text{out}}(g_t)) = \text{avg}\left(\mathcal{E}(g_t - \bar{g})^2\right) + E_{\text{out}}(\bar{g})$$

这里各项可以解释如下：

- $\text{avg}(E_{\text{out}}(g_t))$：算法$\mathcal{A}$效果的期望
- $E_{\text{out}}(\bar{g})$：是各g_t“共识”的效果，称作**偏差（bias）**
- $\text{avg}\left(\mathcal{E}(g_t - \bar{g})^2\right)$：各个$g_t$与它们“共识”之间偏离量的期望，称作**方差（variance）**。方差衡量了g_t内部的“混乱程度”

而均匀混合的过程实际是减少方差的过程，因此模型的表现更加稳定

线性混合与任意混合法

线性混合法比均匀混合法更复杂一些，每个g_t的权重不再相等，而是各自有一个α_t作为权重与之对应，形式为

$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot g_t(\mathbf{x})\right), \alpha_t \ge 0$$

可以看到，其实就是对$g_t(\mathbf{x})$做了一个线性组合，这也是线性混合法这个名称的来历。这里最核心的问题就是怎样的α_t最好。一个简单且直接的想法就是，最好的α_t肯定使E_{in}最小，即满足$\min_{\alpha_t\ge 0} E_{\text{in}}(\boldsymbol{\alpha})$。对应的优化问题为（以回归问题举例）

$$\min_{\alpha_t\ge 0} \frac{1}{N}\sum_{n=1}^{N}\left(y_n - \sum_{t=1}^{T}\alpha_t g_t(\mathbf{x}_n)\right)^2$$

这个优化问题的形式很像线性回归+多项式变换的问题，求解则像概率SVM那样，使用两阶段学习的方法求解：第一步先学习出g_t，第二步再做线性回归学习出α_t。因此线性混合可以理解为下面的过程：总体是一个线性模型，每个g_t看作是多项式变换，然后再加上条件说$\alpha_t \geq 0$。但是有了限制条件以后怎么求解？对那些小于0的g_t，只需要反过来用就行，即$\alpha_t g_t(\mathbf{x}) = |\alpha_t|(-g_t(\mathbf{x}))$（尤其是在二元分类中，99%的错误率实际就是99%的正确率）。因此在真正使用时，通常不考虑限制条件

在现实世界中，这些g_t通常来自于不同的假设集合，即$g_1 \in \mathcal{H}_1, g_2 \in \mathcal{H}_2, \dots, g_T \in \mathcal{H}_T$。一个直观的做法是我们使用不同的假设集合，在每个集合上都找到E_{in}最小的g_t，将它们混合起来。但是回忆之前的课程曾经提过，在一堆最好的假设函数里再选择一个更好的假设函数（称为best of best），模型的复杂度其实是$d_{\text{VC}}\left(\bigcup_{t=1}^{T}\mathcal{H}_t\right)$，需要对最后的模型做小心的验证。而线性混合法中的一个特例是验证法，这就表示用E_{in}来做线性混合的标准，复杂度要$\geq d_{\text{VC}}\left(\bigcup_{t=1}^{T}\mathcal{H}_t\right)$（因为best of best只是特例而已）。因此在实践中不建议用E_{in}来选择，而是选择让E_{val}最小的$\boldsymbol{\alpha}$，而且各个g_t要用从训练集得到的g_t^-

所以，这里推荐的做法为

- 使用训练数据$\mathcal{D}_{\text{train}}$得到$g_1^-, g_2^-, \dots, g_T^-$
- 对验证集$\mathcal{D}_{\text{val}}$中的每条数据做变换得到$(\mathbf{z}_n = \boldsymbol{\Phi}^-(\mathbf{x}_n), y_n)$，其中$\boldsymbol{\Phi}^-(\mathbf{x}) = (g_1^-(\mathbf{x}), \dots, g_T^-(\mathbf{x}))$

在经过上述训练和转换后，线性混合法对$\{(\mathbf{z}_n, y_n)\}$做线性模型得到最优的$\boldsymbol{\alpha}$，然后返回的$G_{\text{LINB}}(\mathbf{x})$是$\boldsymbol{\alpha}$与$\boldsymbol{\Phi}(\mathbf{x})$内积的线性模型，这里$\boldsymbol{\Phi}(\mathbf{x}) = (g_1(\mathbf{x}), \dots, g_T(\mathbf{x}))$

当然，这里可以不必要拘泥于线性模型，甚至可以使用非线性模型。此时整个过程称为任意混合法，或者称为**堆叠法（stacking）**。即首先计算$\tilde{g} = \text{Any}(\{(\mathbf{z}_n, y_n)\})$，最后返回$G_{\text{ANYS}}(\mathbf{x}) = \tilde{g}(\boldsymbol{\Phi}(\mathbf{x}))$。堆叠法能力更强，实际上实现了前面提到的条件混合法（根据特长混合法），但是也更容易过拟合

经典案例：KDD Cup 2011冠军队的解决方案

Bagging

前面讲的混合模型的方法基本都是在得到g_t以后的进一步处理。那么有没有可能一边学g_t一边将它们聚合起来？首先，由前面均匀混合模型一节的讲解，混合模型的关键是所有g_t必须非常不一样。这种多样性从何而来？可以有这么几个方面：

- g_t来自于不同的模型集合，即$g_1 \in \mathcal{H}_1, g_2 \in \mathcal{H}_2, \dots, g_T \in \mathcal{H}_T$
- g_t来自于同一个模型集合的不同超参数，例如GD时设置学习率$\eta = 0.001, 0.01, \dots, 10$
- 算法本身有随机性，例如使用不同的随机种子来得到不同的PLA
- 数据本身具有随机性，例如交叉验证时得到不同的g_v^-

那么有没有可能进一步，我们使用同一份数据，对这份数据施加一些随机性，又不使用g^-，而是制造很多不同的g？回顾前面讲到的偏差-方差之间的关系，可知大家的“共识”比直接使用$\mathcal{A}(\mathcal{D})$要稳定，但是代价是每次都要新鲜的数据集$\mathcal{D}_t$。此外，$\bar{g}$是从无限多个g产生，这个无限也不可能达到。因此，需要做两个妥协：使用有限多但是足够大的T，以及只使用一份$\mathcal{D}$来模拟产生T份不同的数据集。对于后者，要用到统计学上一个重要的工具——bootstrapping（直译为“拔靴法”，或者意译为“自助法”），其核心做法是通过对原始数据集$\mathcal{D}$进行有放回抽样，每取N'条数据形成一个$\mathcal{D}_t$（N'不一定非要等于N）。由于有放回的动作，因此每个$\mathcal{D}_t$中都会有某几条数据重复了若干次，同时又有若干条数据没有出现

这样，就可以得到一个bootstrapping模型聚合的过程：对$t = 1, 2, \dots, T$，通过bootstrapping得到一个大小为N'的数据集$\tilde{\mathcal{D}}_t$，在这个数据集上运行算法$\mathcal{A}$得到g_t。最后，对得到的T个g_t，通过均匀混合的方法得到G。这种**元算法**（建立在基算法$\mathcal{A}$之上的算法）称为Bootstrap AGgregration，简写为**BAGging**（中文译为装袋法或者引导聚集算法）

如果基算法对随机性比较敏感，那么bagging的结果会比较好

NTUML 24. 自适应提升算法（Adaptive Boosting）

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/10/05/NTUML-24-Adaptive-Boosting/

本文中“提升算法”一词皆使用英文原文boosting代替

Boosting算法的动机

假设幼儿园老师要教小孩子辨认是什么苹果，那么每次他会给出若干张“是苹果”的照片，另若干张”不是苹果“的照片，让学生给出如何分辨哪些是苹果。有的学生可能会说”圆形的是苹果“，那么老师会高亮犯错的照片，让别的学生继续提出分辨方法，再高亮新分辨方法

犯错误的样本，以此类推

Boosting算法的思想和动机就类似上述过程：用一些很弱的模型g_t（称作基分类器）聚合起来，形成一个复杂的G。这个聚合的算法主要让基分类器专注在重要的（犯过错的）数据上

权重重设造成的多样性

首先回顾之前讲过的bootstrapping方法。假设原始数据集一共有4条数据，$\mathcal{D}=\{(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),(\mathbf{x}_3,y_3),(\mathbf{x}_4,y_4)\}$，在第$t$步使用bootstrapping方法得出来的数据集为$\tilde{\mathcal{D}}_t=\{(\mathbf{x}_1,y_1),(\mathbf{x}_1,y_1),(\mathbf{x}_2,y_2),(\mathbf{x}_4,y_4)\}$，则在计算$\tilde{\mathcal{D}}_t$上的$E_{\text{in}}^{0/1}=\frac{1}{4}\sum_{(\mathbf{x},y)\in\tilde{\mathcal{D}}_t}[\![y\neq h(\mathbf{x})]\!]$时，$(\mathbf{x}_1,y_1)$会被计算两次，$(\mathbf{x}_3,y_3)$不会被计算。从另一个角度看待这个问题，可以说此时自助法实际对原始数据中每个数据点$(\mathbf{x}_n,y_n)$隐含地给出了一个权重$u_n^{(t)}$。对于本例，有$u_1=2,u_2=1,u_3=0,u_4=1$。此时E_{in}的计算方法等价于计算$E_{\text{in}}^{\mathbf{u}}(h)=\frac{1}{4}\sum_{n=1}^4u_n^{(t)}\cdot[\![y_n\neq h(\mathbf{x}_n)]\!]$。也就是说，bagging算法也可以理解为每轮产生一个各个样本的权重分配，该轮的g_t通过最小化加权（该轮权重）的bootstrapping误差来产生不同的分类结果

推而广之，可以通过对各个基算法赋予权重，得到一个基算法的加权

$$E_{\text{in}}^{\mathbf{u}}(h)=\frac{1}{N}\sum_{n=1}^Nu_n\cdot\text{err}(y_n,h(\mathbf{x}_n))$$

当基算法是SVM时，由对偶二次规划问题可知$E_{\text{in}}^{\mathbf{u}}\propto C\sum_{n=1}^Nu_n\widehat{\text{err}}_{\text{SVM}}$，因此最后$\alpha_n$的上限被调整为$Cu_n$。当基算法是Logistic回归时，$E_{\text{in}}^{\mathbf{u}}\propto\sum_{n=1}^Nu_n\text{err}_{\text{CE}}$，而算法一般是使用随机梯度下降SGD，为了体现权重的影响，某个点权重越高，其在SGD时被抽中的机会就应该越大。权重高了几倍，SGD时该点被抽中的机会就应该高几倍。这种调整方法可以看作是本系列课程第八讲中”类别权重“的扩展

既然u的不同可以导致基算法回传的g不同，接下来的问题自然就是如何合理地设置u使得g越不一样越好。根据刚才的分析，有

$$g_t\leftarrow\mathop{\arg\min}_{h\in\mathcal{H}}\left(\sum_{n=1}^Nu_n^{(t)}[\![y_n\neq h(\mathbf{x}_n)]\!]\right)$$
$$g_{t+1}\leftarrow\mathop{\arg\min}_{h\in\mathcal{H}}\left(\sum_{n=1}^Nu_n^{(t+1)}[\![y_n\neq h(\mathbf{x}_n)]\!]\right)$$

如果g_t在使用$\mathbf{u}^{(t+1)}$时非常不好，在g_{t+1}的最小化过程中，就不应该选到g_t，也不应该选到和g_t很像的假设函数，此时g_{t+1}和g_t就非常不像了。为了做到这一点，可以考虑构造一个最优的$\mathbf{u}^{(t+1)}$，使得此时g_t的表现就像在随机瞎猜，即

$$\frac{\sum_{n=1}^Nu_n^{(t+1)}[\![y_n\neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^Nu_n^{(t+1)}}=\frac{1}{2}$$

将分母展开，有

$$\frac{\sum_{n=1}^Nu_n^{(t+1)}[\![y_n\neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^Nu_n^{(t+1)}[\![y_n\neq g_t(\mathbf{x}_n)]\!]+\sum_{n=1}^Nu_n^{(t+1)}[\![y_n=g_t(\mathbf{x}_n)]\!]}=\frac{1}{2}$$

也就是说，让g_t犯过错误的样本的总权重，等于其正确分类的样本的总权重就可以了。假设前者的总权重是a，后者的总权重是b，那么让前者乘以b，后者乘以a就可以达到想要的效果。或者说，假设g_t在t时犯错的比例为ϵ_t，则令每个犯错的点权重都$\propto(1-\epsilon_t)$，每个正确的点权重都$\propto\epsilon_t$即可得到一个最优的权重调整方法

自适应Boosting算法

根据上一节最后推出来的结论，可以定义一个缩放因子$\blacklozenge_t$：

$$\blacklozenge_t=\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

其中

$$\epsilon_t=\frac{\sum_{n=1}^Nu_n^{(t)}[\![y_n\neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^Nu_n^{(t)}}$$

每个错分的数据点的权重乘以$\blacklozenge_t$，正确分类点的权重除以$\blacklozenge_t$即可。这种调整方法跟刚才提到的最优权重调整法是一样的，而且透露了更多的物理意义：当且仅当$\epsilon_t \leq \frac{1}{2}$时，才有$\blacklozenge_t \geq 1$。这意味着错分样本会被放大，正确样本会被缩小。这也是自适应提升算法（Adaboost）得到各种不一样的g_t的方法

这样可以得到一个初始的算法的核心部分：每一轮t通过让算法$\mathcal{A}$最小化$\mathbf{u}^{(t)}$加权的0-1误差，可以得到该轮情况下最优的g_t。然后，使用上述缩放动作，将$\mathbf{u}^{(t)}$更新为$\mathbf{u}^{(t+1)}$，为下一轮训练做准备。这里还有两个问题没有解决：

- 初始的$\mathbf{u}^{(1)}$怎么获得
- 最后如何聚合这些g_t得到$G(\mathbf{x})$

第一个问题比较直接，要是最开始的E_{in}最差，只需要让每个样本的权重都一样就可以，即$u_n^{(1)} = 1/N$。第二个问题稍微要绕一下：直观的方法是跟每个g_t相同的票数，但是仔细想一想，g_1的E_{in}可能会比较好，g_2的E_{in}可能会比较差，让这两个效果不同的分类器票数相同，不太公平。因此最适合的方法是使用线性或者非线性的方法将这些g_t混合起来。

下面讲的方法就是一种在训练出g_t时就能顺便求出α_t的算法。确定各个g_t时核心思想是对好的g_t权重α_t更大，给坏的g_t权重α_t更小。什么样的g_t是好的g_t呢？很自然，E_{in}越小的g_t应该越好，因此ϵ_t应该越小越好，即$\blacklozenge_t$应该越大越好。这也就意味着，α_t应该是$\blacklozenge_t$的单调（增）函数。这里使用了$\alpha_t = \ln(\blacklozenge_t)$来确定，而且通过两个特例可以看出选择这个函数的合理性：

- 如果$\epsilon_t = \frac{1}{2}$，说明此时这个g_t其实就是在瞎猜。其对应的$\blacklozenge_t = 1, \alpha_t = 0$，因此这个分类器没有话语权
- 如果$\epsilon = 0$，说明一个g_t就能搞定这个数据集，其对应的$\blacklozenge_t = \infty, \alpha_t = \infty$，它的意见压倒一切

因此，最后得到的这个算法，称为自适应boosting（AdaBoost）算法，其实包含了三个重要的组成成分

- 很弱的基分类算法$\mathcal{A}$
- 最优的重赋权因子$\blacklozenge_t$
- 神奇的线性组合系数α_t

AdaBoost算法的流程可整理如下

> 初始化$\mathbf{u}^{(1)} = [\frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N}]$
>
> 对$t = 1, 2, \ldots, T$
>
> 1. 通过$\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$得到$g_t$。其中$\mathcal{A}$是要减小$\mathbf{u}^{(t)}$加权的0/1误差
> 2. 使用如下方法将$\mathbf{u}^{(t)}$更新为$\mathbf{u}^{(t+1)}$
>
> $$\begin{aligned} [\![y_n \neq g_t(\mathbf{x}_n)]\!] : u_n^{(t+1)} &\leftarrow u_n^{(t)} \cdot \blacklozenge_t \quad \text{(incorrect examples)} \\ [\![y_n = g_t(\mathbf{x}_n)]\!] : u_n^{(t+1)} &\leftarrow u_n^{(t)} / \blacklozenge_t \quad \text{(correct examples)} \\ \blacklozenge_t &= \sqrt{\frac{1-\epsilon_t}{\epsilon_t}} \\ \epsilon_t &= \frac{\sum_{n=1}^N u_n^{(t)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^N u_n^{(t)}} \end{aligned}$$
>
> 3. 计算$\alpha_t = \ln(\blacklozenge_t)$
>
> 返回$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \alpha_t g_t(\mathbf{x})\right)$

AdaBoost比较引人关注的地方是，它存在一个理论保证，即对最后返回的聚合模型G，有

$$E_{\text{out}}(G) \leq E_{\text{in}}(G) + O\left(\sqrt{O(d_{\text{VC}}(\mathcal{H}) \cdot T \log T) \cdot \frac{\log N}{N}}\right)$$

原文证明，如果能保证总有$\epsilon_t \leq \epsilon < \frac{1}{2}$，则经过$T = O(\log N)$就能有$E_{\text{in}}(G) = 0$，也就是说上式中上界的第一项可以非常小。在上界的第二项中，$O(d_{\text{VC}}(\mathcal{H}) \cdot T \log T)$实际是所有可能$G$的VC维，随着$T$的增长其增长比较慢，因此第二项也会很小。这意味着，即便是对很弱的$\mathcal{A}$，只要其效果比瞎猜的好，哪怕只是好一点，在AdaBoost的作用下最后的G就可以特别强（$E_{\text{in}} = 0$且E_{out}很小）

自适应Boosting实战

（本节只是算法演示，唯一的一点就是重新提了作业2中的决策树桩decision stump。这个模型的核心思想就是在空间里做水平/竖直切面，而且其很容易优化：如果数据集的大小为N，维度为d，则优化这个模型的时间复杂度仅为$O(d \cdot N \log N)$。最后，AdaBoost还

有特征选择的功能，例如人脸识别的那个例子）

NTUML 25. 决策树

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/10/07/NTUML-25-Decision-Tree/
- **版权声明：** 本博客所有文章除特别声明外，均采用 CC BY-NC-SA 3.0 许可协议。转载请注明出处！

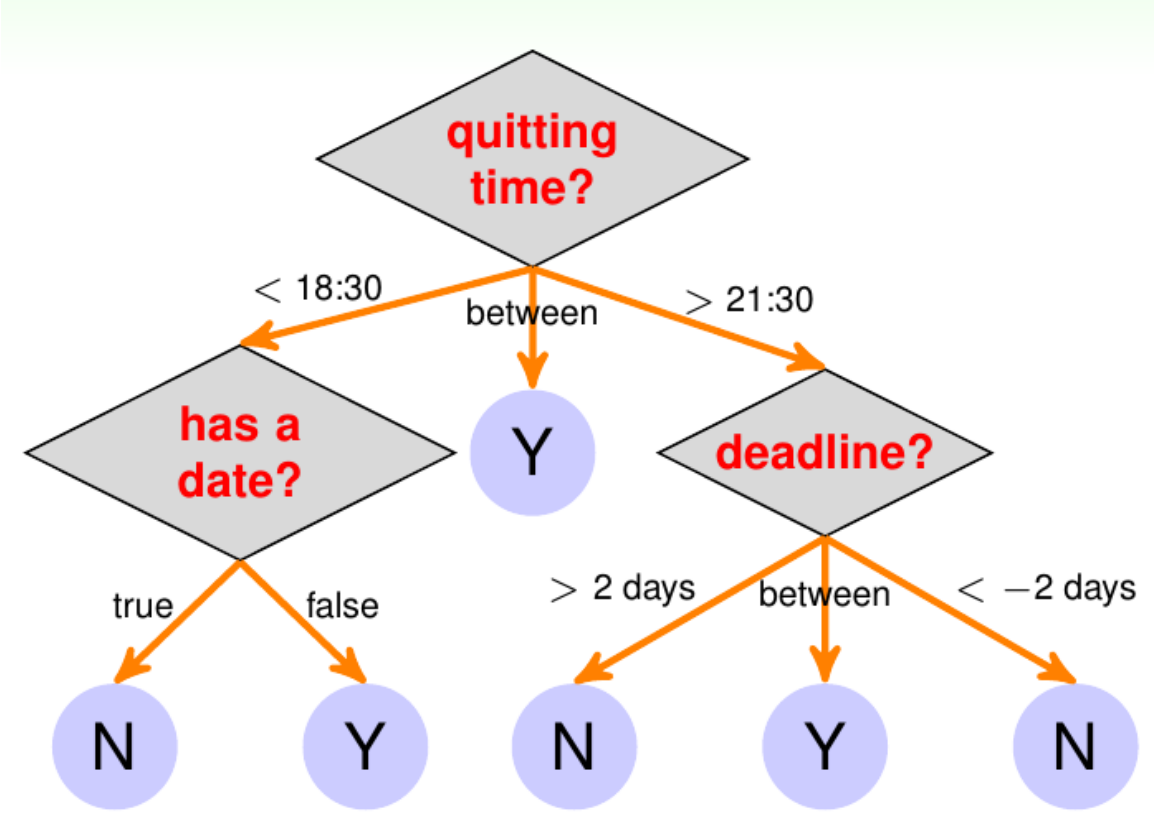
决策树假设

模型聚合总体来讲有两种面向。一种是模型混合法，即在已经知道有哪些基分类器g_t以后做聚合，返回聚合得到的模型G；另一种是模型学习法，即事前不知道有哪些基分类器，一边求g_t一边做聚合。聚合这些模型时，总体来讲又有三种方式：将每个g_t按照等权重聚合、按照重要性分配不同权重聚合，以及按照条件做聚合。决策树就是一种传统的实现条件聚合的学习模型，模仿了人类做决策的过程

之前讲的所有方法可以按照如上的划分方式做出下表

聚合形式	混合	学习
均匀	投票法/求均值法	Bagging
非均匀	线性组合	AdaBoost
条件	堆叠法	决策树

下图给出了一个使用决策树决定是否学习MOOC课程的例子



使用决策树决定是否学习MOOC课程

从图中可知，从决策树的树根出发，其到任意一个叶子节点的路径都是一个完整的决策过程

接下来，考虑如何表示决策树的假设模型。如前所述，决策树是一个条件模型，因此可以套用第23讲的内容，将其最基本的形式写出，为

$$G(\mathbf{x}) = \sum_{t=1}^{T} q_t(\mathbf{x}) \cdot g_t(\mathbf{x})$$

这里的基假设g_t其实就是路径t中最后一个节点（叶子节点）的内容，通常是一个常数。条件$q_t(\mathbf{x})$是上图中橘色的箭头，可以写为$[\![\text{is } \mathbf{x} \text{ on path } t]\!]$，即判断输入是否在某条路径$t$上。通常决策的过程（即路径$t$上）会有很多简单的决定（在上图中以菱形的内部节点表示）。可以看出，做决定的瞬间并不复杂（即内部节点和叶子节点的操作不复杂），决策树的复杂性是埋藏在条件的抉择中

前面对决策树做出的解释是站在了“决策路径”的角度。从这个角度观察，决策树的假设函数可以写为

$$G(\mathbf{x}) = \sum_{t=1}^{T} [\![\mathbf{x} \text{ on path } t]\!] \cdot \text{leaf}_t(\mathbf{x})$$

由于树结构天然的递归性，也可以站在递归的角度对描述决策树。此时，决策树由根节点、分枝条件和子决策树构成。此时决策树的假设函数为

$$G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \cdot G_c(\mathbf{x})$$

这里，C是由该根节点分出的子树个数，也是该节点的分枝条件数。$G(\mathbf{x})$是整棵树对应的假设函数，$b(\mathbf{x})$是分枝条件，$G_c(\mathbf{x})$是第c个分枝对应的子树

决策树直观描述了一个做决策的过程，而且跟人的思维方式类似。由于模型的这种可解释性，它被广泛应用在商业和医疗领域的数据分析中。决策树本身算法的实现也不是很难。效率方面，决策树在训练和测试时都非常有效率，因此总体来看决策树是一个很重要的模型。但是决策树没有太多理论解释，其中的原理基本都是来自直觉。决策树的核心算法是一些启发式算法，可能会让初学者感到迷惑，而且没有一个非常有代表性的算法

决策树算法

从上面递归角度定义的决策树的假设函数，可以写出一个基本的决策树算法。记输入数据为$\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$，函数$\text{DecisionTree}(\mathcal{D})$的基本思想为：如果其发现满足终止条件，则返回基假设函数$g_t(\mathbf{x})$，否则递归做如下四步

1. 学习分枝条件$b(\mathbf{x})$
2. 将数据集$\mathcal{D}$分为C个部分，$\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$
3. 对每个分枝条件构建子树$G_c \leftarrow \text{DecisionTree}(\mathcal{D}_c)$
4. 返回$G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \cdot G_c(\mathbf{x})$

有上述伪代码可知，具体涉及决策树算法时，需要注意考虑的有四个关键点，包括

- 每一步的分枝数
- 分枝条件
- 终止条件
- 基假设函数

这里要讲的决策树算法是C&RT算法（Classification & Regression Tree，从后文看，其更广为人知的名字应该写作CART算法）。在该算法中，有两点的实现非常简单

- 决策树在每一步都分裂成两棵子树，即$C = 2$（决策树是一棵二叉树）
- 基假设函数$g_t(\mathbf{x})$是一个使E_{in}最优的常数
  - 对分类问题（使用0/1误差函数），返回$\{y_n\}$的多数值
  - 对回归问题（使用平方误差函数），返回$\{y_n\}$的平均值

C&RT在分裂内部节点时使用的是决策树桩算法：选择一个特征，在该特征上选择一个最优分割点切下去，一半判定为正例，另一半判定为负例。那么如何决定怎么切最好呢？算法使用了一种叫做“纯度”的指标来判别。由于C&RT最后每个叶子节点（也就是每个g_t）返回的都是一个常数，显然当这个节点包含的所有数据其y_n基本相同时（也就是纯度越高时），常数假设函数的效果最好。因此分枝条件的形式为

$$b(\mathbf{x}) = \underset{\text{decision stumps } h(\mathbf{x})}{\arg\min} \sum_{c=1}^{2} |\mathcal{D}_c \text{ with } h| \cdot \text{impurity}(\mathcal{D}_c \text{ with } h)$$

上式的意义可以解释为，对所有的决策树桩（每个决策树桩实际对应了一个特征），判断使用该决策树桩将数据集分成两块以后，各个子数据集不纯度的加权总和。不纯度最低的（也就是纯度最高的）决策树桩用来真正做分裂。这里函数impurity函数用来判定某个数据集的不纯度，后面会定义。不纯度的权重为该数据集的大小

接下来来看算法核心的部分，即数据集的纯度如何定义。由于纯度的作用是为了返回最后最优的那个常数，因此最后返回的常数好还是不好从某种角度就决定了纯度高还是不高。同样的道理，不纯度的也可以由这个常数的E_{in}来衡量。对于回归问题，不纯度的定义类似于平方误差

$$\text{inpurity}(\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}(y_n - \bar{y})^2$$

其中$\bar{y}$是$\{y_n\}$的平均值

对于分类问题，做法类似，也可以定义为

$$\text{inpurity}(\mathcal{D}) = \frac{1}{N}\sum_{n=1}^{N}[\![y_n \neq y^*]\!]$$

其中y^*是$\{y_n\}$的多数值。不过对于分类问题，可以进一步分析：如果使用分类误差，要做的其实是在所有$[\![y_n = k]\!]$的选择里面，找到最多人资瓷的那个K。也就是说，分类误差可以写为

$$1 - \max_{1\leq k\leq K}\frac{\sum_{n=1}^{N}[\![y_n = k]\!]}{N}$$

这个值只有在$k = y^*$时才能取得最优。进一步地，如果考虑进所有k，可以得到一个新的衡量不纯度的函数，为

$$1 - \sum_{k=1}^{K}\left(\frac{\sum_{n=1}^{N}[\![y_n = k]\!]}{N}\right)^2$$

该值又称为**Gini指数**。实际上，在做决策树算法时，如果要解决分类问题，通常用Gini判定不纯度；如果要解决回归问题，通常用回归误差来判定不纯度

最后，考察算法何时终止。事实上，算法在以下两种条件下都会停止

- 所有y_n都一样，则这个子数据集已经达到了至纯的境地，$g_t(\mathbf{x}) = y_n$
- 所有$\mathbf{x}_n$都一样，则不会学出决策树桩

使用这种终止条件得到的C&RT决策树，称为一棵*完全长成*的C&RT决策树

C&RT启发式算法

将上述对四个关键点的设计代回到决策树的基础算法框架中，就能得到一个可用的完整算法。但是这个算法还有一个问题：如果所有$\mathbf{x}_n$两两不同，这个算法的E_{in}必然是0。这意味着这个算法的VC维非常大，很容易有过拟合的危险。尤其是随着分枝的深入，越接近叶子节点时，决策树子树处理的数据实际上越少，这就更增大了过拟合发生的可能性，因此需要向这个算法加入一些正则项

决策树算法中一个很自然的正则项是对树的叶子节点数做一个限制，即$\Omega(G) = \text{NumberOfLeaves}(G)$。这样，带有正则项的决策树要优化的目标函数变为

$$\underset{\text{all possible } G}{\arg\min}\ E_{\text{in}}(G) + \lambda\Omega(G)$$

这样得到的树称作*修剪过的*决策树

但是注意目标函数里是考虑*所有可能的*G，这个搜索空间在实际实现时有一些困难，因此需要做一些修改，即实际的搜索空间是

- 完全生长的树$G^{(0)}$
- $G^{(i)} = \arg\min_G E_{\text{in}}(G)$，其中$G$是从$G^{(i-1)}$中拿走一片叶子得到的树（实际上，是将其与其兄弟节点合并得到的树）

此时实际的搜索空间大小就是$G^{(0)}$中叶子节点的数目。正则项系数λ仍然是由验证得到

决策树还有一些其他特点。之前考虑的特征基本都是数值特征，分枝时候使用的是决策树桩

$$b(\mathbf{x}) = [\![x_i \leq \theta]\!] + 1,\ \ \theta \in \mathbb{R}$$

但是真正在工作中，可能会遇到大量离散的类别特征，例如性别={男, 女}。对于这种特征，线性模型通常不能直接处理，而决策树不会受到影响，只需要修改一下分枝时用到的模型即可。一般来讲，对类别特征使用的切割方式是使用**决策子集**

$$b(\mathbf{x}) = [\![x_i \in S]\!] + 1,\ \ S \subset \{1, 2, \ldots, K\}$$

（补充一点课堂里没有提到的。对于类别特征，在使用线性模型做处理时，通常的方法是对特征进行**独热编码**，即：第一步将该项类别特征用数字代替，每一位数字对应一个类别特征的具体值，例如上面性别这一项，将男对应为1，将女对应为0。第二步，假设该项类别特征有K种不一样的取值，那么就设置一个长度为K的向量$\mathbf{k}$。对每个具体的值，假设其在第一步被编码为$i, i \in \{0, 1, \dots, K-1\}$，将这个$\mathbf{k}$的其它项设为0，但是把第$i$项设为1。这样，如果样本中"性别"这一列的值是"男"，送进模型之前性别这部分的向量应该是[0, 1]；如果是"女"，则应该是[1, 0]）

C&RT算法在具体实现时还会对缺失值进行处理。假设有一个分枝条件是$b(\mathbf{x}) = [\![\text{weight} \leq 50\text{kg}]\!]$，但是某些样本没有获取到体重的具体值，应该怎么办？在训练时，C&RT算法会使用一种叫做"替代分枝"（surrogate branch）的方法，即它会寻找一些类似的条件来做近似的分枝判断，例如它可能会使用高度是否小于等于155cm来代替判断体重是否小于等于50kg，而替代特征的选择也会在训练时学习到。其原理是，替代分枝$b_1(\mathbf{x}), b_2(\mathbf{x})$等等的分枝效果应该跟最优的分枝效果近似

决策树实战

概括了C&RT算法的好处：解释性强，能比较容易地处理多类别问题、离散特征、缺失特征，而且可以非常有效地训练出一个非线性模型。机器学习里几乎没有其它的模型可以同时具有以上优点

除了C&RT算法，其它决策树算法还包括C4.5和ID3算法，它们使用了一些不同的启发式算法

NTUML 26. 随机森林

随机森林算法

之前的课程里曾经讲述过两种算法：bagging算法和决策树算法。这两种算法的特点都很明显：

- bagging算法需要基分类器g_t的方差尽可能大，然后它会通过投票或求均值的操作来减少最后算法G的方差
- 决策树会根据数据情况选择最佳切割点。数据有一点变化，切割点可能就会不同。因此决策树对数据很敏感，方差比较大（当决策树是完全生长的时候，尤其如此）

既然两个算法一个是能降低方差，另一个方差特别大，那么能不能把这两种算法结合在一起，取长补短？答案是可以的。这种将聚合算法做聚合的算法，称为随机森林，即随机森林是把完全生长的C&RT决策树做bagging，其中"随机"点出了bagging算法所依赖的随机性，"森林"点出了基算法是决策树的内涵。其具体过程为

> 对$t = 1, 2, \dots, T$
>
> 1. 通过对原始数据集$\mathcal{D}$做bootstrapping，得到大小为N'的数据集$\tilde{\mathcal{D}}_t$
> 2. 在$\tilde{\mathcal{D}}_t$上运行决策树算法Dtree，得到基分类器g_t
>
> 对得到的所有g_t做均匀混合，返回$G = \mathrm{Uniform}(\{g_t\})$

随机森林的优点在于，其底层bagging过程可以拆开到不同机器上，而各个基分类器g_t也是独立学出来的，互相没有什么依赖关系，因此整个算法可以并行运行。加上决策树的训练效率很高，因此随机森林非常高效。该算法在继承了决策树算法优点的同时，还因为使用bagging而降低了方差，弥补了完全生长的决策树的缺点。综上，随机森林是一个非常有用的模型

除去bootstrapping中对数据做采样的随机性，随机森林还采用了另一种技术来保证底层基分类器尽量不同（回忆之前所讲的课程，基分类器越千差万别，bagging算法的效果越好）：它还会数据集的特征做随机采样。假设原始数据集$\mathcal{D}$有N条数据，d个特征，那么随机森林用来训练基分类器的数据集$\tilde{\mathcal{D}}_t$大小为N'，特征数为d'。这个对特征做采样的过程可以看作是对原始样本空间$\mathcal{X} \in \mathbb{R}^d$做一个特征投影，投影到一个随机子空间$\mathcal{Z} \in \mathbb{R}^{d'}$的过程。由于通常情况下有$d' << d$，因此随机森林算法的效率可以得到进一步的提升（这种做法也可以用在使用其它模型做基模型的bagging算法里）。随机森林的提出者建议，对g_t的每次节点分裂$b(\mathbf{x})$时都重采样一遍，使用不同的特征子空间，效果会更好

在此基础上，随机森林还可以再做进一步扩展！上面随机采样出d'个特征的过程，其实就是对原始数据$\mathbf{x}$乘以一个投影矩阵P，做特征变换$\mathbf{\Phi}$的过程，即$\mathbf{\Phi}(\mathbf{x}) = \mathrm{P} \cdot \mathbf{x}$，而$\mathrm{P}$是通过对标准正交基（natural basis）做随机采样得到的。新的扩展就在于，投影矩阵P不再限定于是由标准正交基得到的矩阵，而是原始数据投影到不同的方向上，即新的特征是某些原始特征的线性组合。随机森林里考虑的投影通常是低维投影，即只选取d''个非零项做组合。注意当$d'' = 1$，P的每一行$\mathbf{p}_i$都是标准正交基时，这种方法退化到了原来随机子空间的情况。随机森林的提出者建议，对g_t的每次节点分裂$b(\mathbf{x})$时都做一次随机低维投影，效果会更好

袋外估计

本节对之前所讲授的bagging过程做一个更深入的探索：有前面bagging算法可知，在训练每个基分类器g_t时，都会使用一个bootstrapping得到的数据集$\tilde{\mathcal{D}}_t$。由于bootstrapping是有放回抽样的随机均匀采样过程，因此每个$\tilde{\mathcal{D}}_t$有$1-N!/N^N$的概率不会包含所有数据。所以，可以画一个表，来看对每个基分类器g_t，在训练它时用到了哪些数据，没有用哪些数据。下表就给出了这么一个例子

数据	g_1	g_2	g_3	$\cdots$	g_T
$(\mathbf{x}_1,y_1)$	$\tilde{\mathcal{D}}_1$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_2,y_2)$	$\star$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_3,y_3)$	$\star$	$\tilde{\mathcal{D}}_2$	$\star$		$\tilde{\mathcal{D}}_T$
$\cdots$					
$(\mathbf{x}_N,y_N)$	$\tilde{\mathcal{D}}_1$	$\tilde{\mathcal{D}}_2$	$\star$		$\star$

上表中，第t列第n行的星号$\star$代表训练基分类器g_t时没有用到数据$(\mathbf{x}_n,y_n)$，这条数据就称为g_t的**袋外样本**（OOB样本）。首先看抽样N次以后有多少样本是OOB的：每次抽取时，任意一条数据被抽中的概率为$1/N$，所以其不被抽中的概率为$1-1/N$。因此，任意一条样本连续N次都没被抽中的概率为$(1-1/N)^N$。当N很大时，有

$$\lim_{N\to+\infty}\left(1-\frac{1}{N}\right)^N=\lim_{N\to+\infty}\frac{1}{\left(\frac{N}{N-1}\right)^N}=\lim_{N\to+\infty}\frac{1}{\left(1+\frac{1}{N-1}\right)^N}=\frac{1}{e}$$

因此对大小为N的数据集做bootstrapping以后，如果得到的新数据集大小还是N，则OOB样本的数量应该约为N/e（三分之一强）

再看前面给出来的表，可以发现OOB数据有点像以前将验证时候提到的验证集：这些数据也是随机从原始数据中抽取，没有在训练中被使用，而且样本数量足够多。所以要用这些数据来验证g_t吗？理论上可以，但是实际中不需要。因为bagging的方法是模型聚合的方法，它的目的是要让最后聚合到的模型G好，因此需要找到一种方法，能够使用OOB数据来验证G的效果。由前面的知识可知，如果要用OOB数据验证G，就要保证这部分数据没有受到污染。以上表为例，$(\mathbf{x}_N,y_N)$是g_3和g_T的OOB数据，那么它可以用来验证$G_N^-(\mathbf{x})=\text{average}(g_3,g_T)$。以此类推，对每一条数据$(\mathbf{x}_n,y_n)$，都可以构造出来这么一个子模型$G_n^-$，因此整个模型$G$的oob误差就可以通过下式计算

$$E_{\text{oob}}(G)=\frac{1}{N}\sum_{n=1}^N\text{err}(y_n,G_n^-(\mathbf{x}_n))$$

即bagging算法的特性是，得到G后就能马上知道它的效果如何，因为它可以通过E_{oob}来做一个自我验证，对超参数进行调整。与之前的验证方法相比，有了OOB数据和E_{oob}以后，既不用将数据集分为两份（验证集和训练集），也不用再在确定好超参数以后重新训练模型了

特征选择

OOB数据除了可以对G做验证，还有别的用途。对于一份原始的d维数据$\mathbf{x}=(x_1,x_2,\ldots,x_d)$，这里可能会有两种特征：

- 冗余特征，例如数据里可能同时有“年龄”和“生日”两项
- 不相关特征，例如做癌症推断的数据里可能有“医保类型”一项，而这一项对判断是否为癌症起不到作用

为了降低模型的复杂度，提高训练效率，需要将这些特征移除掉。尽管这些事情理论上可以手工完成，但是我们仍然希望能够通过学习的方法达到目的，即希望能学到一个特征子集的转换$\mathbf{\Phi}(\mathbf{x})=(x_{i1},x_{i2},\ldots,x_{i'd}),d'<d$，然后用转换后的特征来学习$g_t$

特征选择的好处，首先它的效率得到了提升，而且排除掉了$d-d'$项噪声，降低了过拟合的可能性。最后，对d'项特征作分析可以集中在有用的特征上，提高模型的可解释性。但是从另一个角度看，特征选择也会带来风险。首先它是一个组合问题，选择最优组合的过程可能会占用额外时间；其次，如果特征选择没有做好，可能反而就会导致过拟合，得到错误的解释（或者分析时只能看到相关性不能看到因果性）。因此，这里责任最重的地方就是如何在d项特征中找到d'维特征子集。幸运的是，决策树本身自带一些特征选择的方法（因为它在训练时要决定在哪个特征上做分割是最有效的）

前面提过，特征选择本质是一个特征组合问题。有组合，就会有组合爆炸的可能性。那么如何降低这种可能性的发生，降低计算量？一种比较自然的启发式算法是，可以尝试计算每个特征的重要程度$\text{important}(i)\text{ for }i=1,2,\ldots,d$。如果这个重要程度能够计算出来，那么只需要选$d'$个重要程度最大的特征$i_1,i_2,\ldots,i_{d'}$就可以。这种简单的方式在线性模型中很容易实现，因为重要程度可以根据模型权重向量$\mathbf{w}$的分量$|w_i|$来判断。$|w_i|$越大，特征的重要程度就越大。而$\mathbf{w}$的学习很直接，也很容易。然而，非线性模型中由于增添

了特征之间的交互，特征重要性的判断通常会难很多。但是随机森林本身的一些机制，使得其成为了一种很容易做特征选择的非线性模型

使用随机森林做特征选择的基本原理是，如果特征i比较重要，那么往这个特征上塞入一些噪声，模型的表现肯定会变差。那么问题是，如何给某个特征加入一些随机值？不能给入服从均匀分布或者高斯分布的噪声，因为原始数据可能不服从这样的分布，武断地加入某个确认分布的噪声会改变数据在第i个维度上的分布$P(x_i)$。这里利用了类似bootstrapping的方法：为了验证某个特征i的重要性，对$\{x_{n,i}\}_{n=1}^N$重新生成一个排列（permutation），这样，数据在该特征上的分布$P(x_i)$就不会有什么变化了。记重排列特征i上的值以后得到的数据集为$\mathcal{D}^{(p)}$，特征i的重要性可以根据如下方式（称为排列测试 permutation test）做一粗略估计：

$$\text{importance}(i) = \text{performance}(\mathcal{D}) - \text{performance}(\mathcal{D}^{(p)})$$

这里，求$\text{performance}(\mathcal{D}^{(p)})$时需要用重排列后的数据集重新训练和验证模型。不过对于随机森林，可以免去验证的步骤，使用OOB就可以。甚至，可以不重新训练G，而是把OOB中的第i个特征做一个重排列。即

$$\text{importance}(i) = E_{\text{oob}}(G) - E_{\text{oob}}^{(p)}(G)$$

其中，在计算$E_{\text{oob}}(G)$时，如果基分类器$g(t)$用到了第i维特征$x_{(n,i)}$，随机选择g_t的一个OOB数据点的第i维特征替代（这样该数据没有参与训练），就可以得到$E_{\text{oob}}^{(p)}(G)$

随机森林实战

本节通过一些实验进一步展示了随机森林的特点：在树的数量足够多的情况下，随机森林可以学习出光滑，而且类似最大间隔的效果。而且，随机森林要求训练时用的树越多越好。理论上，用无限多棵树可以学到一个稳定的模型G，有限多的树只是能不断逼近这个结果

NTUML 27. 梯度提升决策树（GBDT）

AdaBoost决策树

在介绍AdaBoost决策树之前，先回顾一下之前讲过的随机森林。随机森林的核心是，将一些决策树（内层）使用bagging的方法（外层）聚合起来。如果把决策树和AdaBoost算法结合起来，那么就得到了AdaBoost决策树。但是有一点需要注意：AdaBoost算法里要求基算法在训练时能够接受样本的权重，但是之前CART算法不能根据各个样本的权重做调整，因此现在最关键的问题是要对决策树算法作出修改，使其可以接受样本的权重，在每个t得到不一样的g_t ，即要实现一个加权的决策树算法$\text{DTree}(\mathcal{D}, \mathbf{u}^{(t)})$

加权算法的核心是，其需要根据权重最佳化根据权重加权过的E_{in}。因此相对应的有两种处理办法

- 将原有的基算法看作是白盒，查找其在何处修改E_{in}，然后在该处加上对权重的处理逻辑。例如SVM中，就是对应hinge loss的部分。但是决策树算法实现里有很多启发式算法，如果挨个分析哪部分跟E_{in}有关，就比较麻烦
- 将原有的基算法看作是黑盒，将送入算法的数据做变化。回想之前对bootstrapping的描述，其实质上也隐含了“权重”的概念：权重是3的数据，其实就是该数据复制3份；权重是7的数据，其实就是该数据复制7份。那么，推而广之，对送入算法的数据的变化方法就是根据数据的权重做采样。这样，就可以得到一组大小为N'的新数据集$\tilde{\mathcal{D}}_t$，其中每种数据的比例正比于$\mathbf{u}$，也就能体现权重的影响

因此，AdaBoost决策树的核心算法就是，将原始数据集$\mathcal{D}$按照$\mathbf{u}^{(t)}$做抽样，然后使用得到的新数据集$\tilde{\mathcal{D}}_t$按照原有的决策树算法训练。不过这里还有一个矛盾：AdaBoost通常需要一些比较弱的基分类器，而原有的决策树算法初衷是要树完全增长。回想原来AdaBoost里每个基分类器的权重（票数）α_t是由加权错误率ϵ_t决定，即$\alpha_t = \ln \blacklozenge_t = \ln \sqrt{(1-\epsilon_t)/\epsilon_t}$。如果决策树$g_t$在*所有*$\mathbf{x}_n$上自由生长，且训练数据两两不等，那么就会有$E_{\text{in}}(g_t) = 0$，进而$E_{\text{in}}^{\mathbf{u}}(g_t) = \epsilon_t = 0$，即$\alpha_t = \infty$，这棵树成了一个独裁者！

鉴于上面的情势，考虑AdaBoost的核心思想，为了避免上述状况的发生，还是需要基分类器都足够弱。让基分类器（这里是决策树）足够弱的方法有两种：

- 对决策树进行剪枝，参见前面讲过的剪枝方法。或者一种最简单有效的方法是限制树的高度
- 不送入全部数据，使用前面提到的采样方法。采样概率$\propto \mathbf{u}^{(t)}$

综合以上讨论，可以得到：AdaBoost决策树 = AdaBoost + 采样概率$\propto \mathbf{u}^{(t)}$ + 剪枝过的决策树$\text{DTree}(\tilde{\mathcal{D}}_t)$

对树做剪枝的极端情况是，树的高度≤ 1。考虑之前讲过的决策树（C&RT算法），树高为1的数只有一个分枝。回顾决策树的分枝条件定义，其要做的是让左右子树各自的不纯度两者加起来的值最小。如果不纯度的定义为二元分类误差，那么实际上此时决策树退化成

了一个决策树桩。即前面提到的AdaBoost决策树桩是AdaBoost决策树的一种特殊情况

最优化角度看AdaBoost

之前讲解AdaBoost时，曾经提出过样本权重的更新法则

$$u_n^{(t+1)} = \begin{cases} u_n^{(t)} \cdot \blacklozenge_t & \text{if incorrect} \\ u_n^{(t)} / \blacklozenge_t & \text{if correct} \end{cases}$$

这个式子可以做进一步化简：考虑之前提到的，如果$y_n \in \{-1, +1\}$，那么当分类结果正确时，有$y_n g_t(\mathbf{x}_n) = 1$，否则$y_n g_t(\mathbf{x}_n) = -1$。因此上式可以简写为递推式

$$u_n^{(t+1)} = u_n^{(t)} \cdot \blacklozenge_t^{-y_n g_t(\mathbf{x}_n)}$$

带入$\blacklozenge_t = e^{\alpha t}$，可以得到

$$u_n^{(t+1)} = u_n^{(t)} \cdot \exp(-y_n \alpha_t g_t(\mathbf{x}_n))$$

将这个递推式展开，有

$$u_n^{(T+1)} = u_n^{(1)} \cdot \prod_{t=1}^{T} \exp(-y_n \alpha_t g_t(\mathbf{x}_n)) = \frac{1}{N} \cdot \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

注意最后指数里有一部分可以对应于AdaBoost的总假设函数$G(\mathbf{x})$：$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})\right)$。可以将$\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})$记为$\{g_t\}$在$\mathbf{x}$上的投票得分（voting score）

前面还说过，AdaBoost实际上是一个线性混合的过程：将所有基假设函数看作是对原始数据的一个变换，然后在转换后的空间里学习出一个线性模型，这个线性模型就说明了每个g_t会占几票。如果将基假设函数的变换过程$g_t(\mathbf{x}_n)$写为$\phi_i(\mathbf{x}_n)$，将每个g_t的票数写为权重的样子w_i，那么投票得分就可以改写为$\sum_{i=1}^{T} w_i \phi_i(\mathbf{x}_n)$。回顾之前提到的SVM间隔的定义，$\text{margin} = \frac{y_n \cdot (\mathbf{w}^\mathsf{T} \phi(\mathbf{x}_n) + b)}{\|\mathbf{w}\|}$，如果忽略掉这个式子中的常数项$b$，正则项$1/\|\mathbf{w}\|$和用来判断点是落在分离超平面正确一侧还是错误一侧的y_n，其核心部分$\mathbf{w}^\mathsf{T} \phi(\mathbf{x}_n)$其实传递的是点到超平面距离的概念。因此，$y_n \cdot \text{voting score}$实际上就是带符号的且没有做过正则化的间隔，我们也希望这个间隔是一个比较大的正数。既然如此，那么$\exp(-y_n(\text{voting score}))$就应该越小越好，$u_n^{(T+1)}$也应该越小越好。**实际上，AdaBoost就是在减小**$\sum_{n=1}^{N} u_n^{(t)}$，其本质也是要让分离超平面离数据点的间隔变大。也就是说，它要减小的是

$$\sum_{n=1}^{N} u_n^{(T+1)} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

再次搬出之前提到过很多次的线性得分$s = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)$，以及之前提到的0/1误差函数$\text{err}_{0/1}(s, y) = [\![ys \le 0]\!]$。如果将上面的这个式子看作是AdaBoost某种意义上要减小的误差函数，记为$\widehat{\text{err}}_{\text{ADA}}$，那么有$\widehat{\text{err}}_{\text{ADA}} = \exp(-ys)$（称为指数损失函数）也是$\text{err}_{0/1}$的上界。因此可以说，AdaBoost也是在减小0/1误差函数的凸上限，进而达到解好分类问题的目的

接下来，要证明AdaBoost算法的确可以减小$\widehat{\text{err}}_{\text{ADA}}$。这个证明过程类似于之前提到的梯度下降法（GD），只不过GD是要找到一个向量$\mathbf{v}$作为下降方向，而AdaBoost是要找到一个基函数g_t作为下降方向。即在第t步迭代，为了寻找最优的g_t，就是要求解

$$\min_h \widehat{E}_{\text{ADA}} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n)\right)\right)$$

将上式展开，代入$u_n^{(t)} = \frac{1}{N} \cdot \exp(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n))$，可知

$$\begin{aligned} \min_h \widehat{E}_{\text{ADA}} &= \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n)\right)\right) \\ &= \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n)\right) \cdot \exp(-y_n \eta h(\mathbf{x}_n)) \\ &= \sum_{n=1}^{N} u_n^{(t)} \exp(-y_n \eta h(\mathbf{x}_n)) \end{aligned}$$

将上式在零点附近做泰勒展开，可得

$$\begin{aligned}\min_h \widehat{E}_{\text{ADA}} &= \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta h(\mathbf{x}_n)) \\ &\approx \sum_{n=1}^N u_n^{(t)} (1 - y_n \eta h(\mathbf{x}_n)) \\ &= \sum_{n=1}^N u_n^{(t)} - \eta \sum_{n=1}^N u_n^{(t)} y_n h(\mathbf{x}_n)\end{aligned}$$

这就很像之前GD $\min_{\|v\|=1} E_{\text{in}}(\mathbf{w}_t + \eta \mathbf{v}) \approx E_{\text{in}}(\mathbf{w}_t) + \eta \mathbf{v}^{\mathsf{T}} \nabla E_{\text{in}}(\mathbf{w}_t)$的形式了！同理，这里$\sum_{n=1}^N u_n^{(t)}$是常数项，因此最好的$h$会最小化$\sum_{n=1}^N u_n^{(t)}(-y_n h(\mathbf{x}_n))$。

由于这里考虑的是二元分类问题，因此$y_n \in \{-1, +1\}, h(\mathbf{x}_n) \in \{-1, +1\}$。利用这个性质，展开要最小化的式子，有

$$\begin{aligned}\sum_{n=1}^N u_n^{(t)}(-y_n h(\mathbf{x}_n)) &= \sum_{n=1}^N u_n^{(t)} \begin{cases} -1 & \text{if } y_n = h(\mathbf{x}_n) \\ +1 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^N u_n^{(t)} + \sum_{n=1}^N u_n^{(t)} \begin{cases} 0 & \text{if } y_n = h(\mathbf{x}_n) \\ 2 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^N u_n^{(t)} + 2 \sum_{n=1}^N u_n^{(t)} [\![y_n \neq h(\mathbf{x}_n)]\!] \\ &= -\sum_{n=1}^N u_n^{(t)} + 2 E_{\text{in}}^{\mathbf{u}^{(t)}}(h) \cdot N\end{aligned}$$

也就是说，要最小化$\sum_{n=1}^N u_n^{(t)}(-y_n h(\mathbf{x}_n))$，实质上就是要最小化$E_{\text{in}}^{\mathbf{u}^{(t)}}(h)$，而这正是AdaBoost的算法$\mathcal{A}$做的事！即$\mathcal{A}$找到了一个好的梯度下降的函数方向$g_t = h$

根据上面的推导，AdaBoost通过近似最小化$\widehat{E}_{\text{ADA}} = \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta h(\mathbf{x}_n))$来找到一个最优的函数$g_t = h$。那么在找到$g_t$以后，能不能不满足于像原始GD那样每次只移动固定的一小步η，而是通过最小化$\widehat{E}_{\text{ADA}} = \sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta g_t(\mathbf{x}_n))$，找一个最优的$\eta_t$，让这一步迈得大一点？观察求和项里的情况：如果分类正确，即$y_n = g_t(\mathbf{x}_n)$，则要求和的那一项变为$u_n^{(t)} \exp(-\eta)$；如果分类错误，就会有$y_n \neq g_t(\mathbf{x}_n)$，要求和的那一项变为$u_n^{(t)} \exp(+\eta)$。因此对求和项分拆并代入$\epsilon_t$的定义，有

$$\begin{aligned}\sum_{n=1}^N u_n^{(t)} \exp(-y_n \eta g_t(\mathbf{x}_n)) &= \sum_{n=1}^N \left(u_n^{(t)} [\![y_n = g_t(\mathbf{x}_n)]\!] \exp(-\eta) + u_n^{(t)} [\![y_n = g_t(\mathbf{x}_n)]\!] \exp(+\eta) \right) \\ &= \exp(-\eta) \cdot \sum_{n=1}^N u_n^{(t)} \cdot \frac{\sum_{n=1}^N u_n^{(t)} [\![y_n = g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^N u_n^{(t)}} + \exp(+\eta) \cdot \sum_{n=1}^N u_n^{(t)} \cdot \frac{\sum_{n=1}^N u_n^{(t)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^N u_n^{(t)}} \\ &= \exp(-\eta) \cdot \sum_{n=1}^N u_n^{(t)} \cdot (1 - \epsilon_t) + \exp(+\eta) \cdot \sum_{n=1}^N u_n^{(t)} \cdot \epsilon_t \\ &= \sum_{n=1}^N u_n^{(t)} \cdot ((1 - \epsilon_t) \exp(-\eta) + \epsilon_t \exp(+\eta))\end{aligned}$$

求解$\frac{\partial \widehat{E}_{\text{ADA}}}{\partial \eta} = 0$可以得到最优的$\eta_t$，而

$$\begin{aligned}&\frac{\partial \widehat{E}_{\text{ADA}}}{\partial \eta} = 0 \\ \Leftrightarrow &\sum_{n=1}^N u_n^{(t)} \cdot (-(1 - \epsilon_t) \exp(-\eta) + \epsilon_t \exp(+\eta)) = 0 \\ \Leftrightarrow &-(1 - \epsilon_t) + \epsilon_t \exp(+2\eta) = 0 \\ \Leftrightarrow &\exp(+2\eta) = \frac{1 - \epsilon_t}{\epsilon_t} \\ \Leftrightarrow &\eta = \ln \sqrt{\frac{1 - \epsilon_t}{\epsilon_t}} = \alpha_t\end{aligned}$$

因此，其实α_t是跨度最大的η_t

综上所述，AdaBoost是沿着近似函数梯度（方向是函数而不是向量），向最优方向做了最大程度的下降

梯度提升

对上一节所讲的内容做一个总结，就是AdaBoost每一步都是在最小化指数误差：每一步找到一个h作为g_t，然后决定要走多远，这个跨度就是α_t

既然AdaBoost实际上每一步都是在最小化某个误差函数，那么可否对其进行扩展，让其每一步优化的误差函数可以是之前讲过的那些误差函数？答案是肯定的，扩展以后得到的算法称为梯度提升算法（Gradient Boosting），形式为

$$\min_{\eta}\min_{h}\frac{1}{N}\sum_{n=1}^{N}\text{err}\left(\sum_{\tau=1}^{t-1}\alpha_\tau g_\tau(\mathbf{x}_n)+\eta h(\mathbf{x}_n),y_n\right)$$

在这个框架下，就可以使用不同的h来做Boosting，解决不同的问题

例如，如果要使用Boosting方法求解回归问题，将$\sum_{\tau=1}^{t-1}\alpha_\tau g_\tau(\mathbf{x}_n)$写成$s_n$，那么误差函数就应该是平方误差，$\text{err}(s,y)=(s-y)^2$。对上式内层求最小$h$的问题，在零点做泰勒展开，有

$$\begin{aligned}\min_{h}\frac{1}{N}\sum_{n=1}^{N}\text{err}(s_n+\eta h(\mathbf{x}_n),y_n)&\approx\min_{h}\frac{1}{N}\sum_{n=1}^{N}\text{err}(s_n,y_n)+\frac{1}{N}\sum_{n=1}^{N}\eta h(\mathbf{x}_n)\frac{\partial\text{err}(s,y_n)}{\partial s}\Big|_{s=s_n}\\&=\min_{h}\text{constants}+\frac{\eta}{N}\sum_{n=1}^{N}h(\mathbf{x}_n)\cdot 2(s_n-y_n)\end{aligned}$$

由于是一个最小化问题，因此要让h越小越好。如果s_n-y_n为正，那么h就应该为负；否则h就应该为正。但是这里没有限制，因此此时有$h(\mathbf{x}_n)=-\infty\cdot(s_n-y_n)$。由于要做的只是寻找一个方向，因此这里可以（也应该）对h的大小做一个限制，h的大小由η来解决。直观的做法是在求解时对h的大小做一个限制，例如$\|h\|=1$，但是有限制的最优化问题求解起来比较麻烦。另一种方法，是借鉴拉格朗日乘子和ℓ_2正则化的思想，将h的大小作为惩罚项放进要求解的问题中，有

$$\begin{aligned}\min_{h}\quad&\text{constants}+\frac{\eta}{N}\sum_{n=1}^{N}(2h(\mathbf{x}_n)(s_n-y_n)+(h(\mathbf{x}_n))^2)\\=\quad&\text{constants}+\frac{\eta}{N}\sum_{n=1}^{N}\left(\text{constant}+(h(\mathbf{x}_n)-(y_n-s_n))^2\right)\end{aligned}$$

如果将上式所有常数项都去掉（其存在与否不影响结果），那么其实就是要找$\min_h\sum_{n=1}^{N}(h(\mathbf{x}_n)-(y_n-s_n))^2$，而这个问题实际上还是一个回归问题，只不过此时目标值从y_n变成了残差y_n-s_n。因此用来做回归的梯度提升就是通过对残差做回归，来找到最优的$g_t=h$

在有了最优的h以后，接下来就是要求解最优的η。这个优化问题可以写为

$$\begin{aligned}&\min_{\eta}\frac{1}{N}\sum_{n=1}^{N}(s_n+\eta g_t(\mathbf{x}_n)-y_n)^2\\=&\min_{\eta}\frac{1}{N}\sum_{n=1}^{N}((y_n-s_n)-\eta g_t(\mathbf{x}_n))^2\end{aligned}$$

其实质是一个单变量的线性回归问题，输入是经过g_t变换过的原始输入，输出是残差项

将上述内容组合在一起，就可以得到梯度提升决策树算法（GBDT）

> $s_1=s_2=\ldots=s_N=0$
>
> 对$t=1,2,\ldots,T$
>
> 1. 使用$\mathcal{A}(\{(\mathbf{x}_n,y_n-s_n)\})$获得$g_t$，其中$\mathcal{A}$是一个最小化平方误差的回归算法，通常是被剪枝过的C&RT决策树
> 2. 计算$\alpha_t=\text{OneVarLinearReg}(\{(g_t(\mathbf{x}_n),y_n-s_n)\})$
> 3. 更新$s_n\leftarrow s_n+\alpha_t g_t(\mathbf{x}_n)$
>
> 返回$G(\mathbf{x})=\sum_{t=1}^{T}\alpha_t g_t(\mathbf{x})$

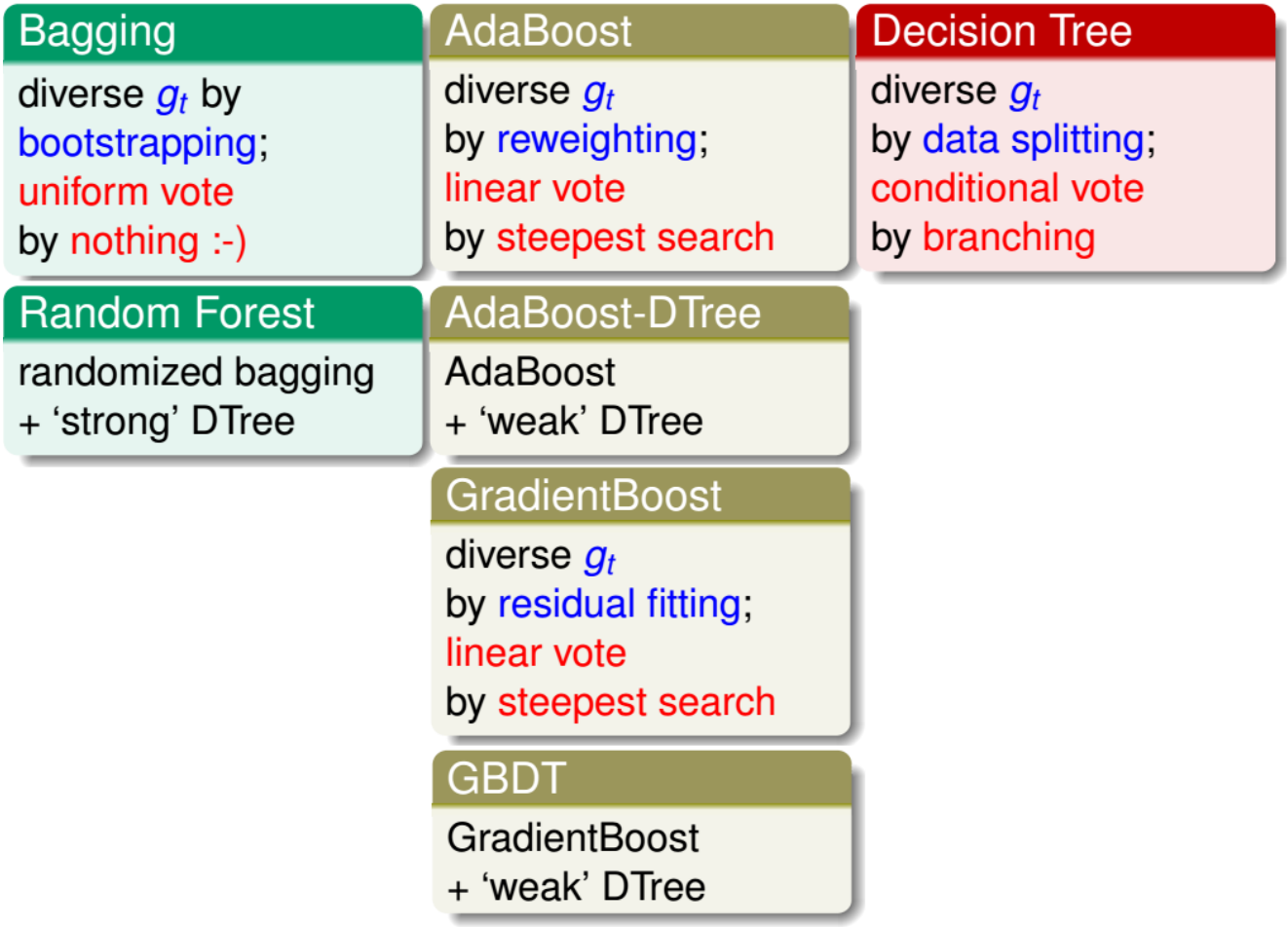
GBDT是AdaBoost-决策树用来求解回归问题的“表亲”，在实践中非常常用

（笔者注：GBDT的两个很有名的实现XGBoost和LightGBM也可以用来求解分类问题，sklearn也有GBDTClassifier）

聚合模型方法综述

至此，所有聚合模型已介绍完毕，这里对它们做一回顾

- 模型的混合法，是在得到各个不同的g_t以后做聚合，分为一下三种混合方式
  - 均匀混合：每个g_t票数相同，追求模型的复杂性
  - 非均匀混合：实际上就是对输入通过g_t做一个变换，然后训练一个线性模型。其追求的是模型的复杂性，但是需要小心使用，防止过拟合
  - 条件混合：与非均匀混合类似，只不过最后训练的是非线性模型。这种方法通常又称为模型的堆叠法（stacking）。其追求的也是模型的复杂性
- 模型的学习法，是一边学习g_t一边做聚合。关系可见下图



其中随机森林、AdaBoost-DTree和GBDT都是很常用的模型

模型聚合有时可以起到特征变换的作用，使得最后得到的$G(\mathbf{x})$很强，避免了欠拟合的问题；有时可以起到正则化的作用，使最后得到的$G(\mathbf{x})$不偏不倚，避免了过拟合的问题。总之，适当的模型聚合（有时又称组合ensemble），可以得到更好的效果

NTUML 28. 神经网络

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/10/19/NTUML-28-Neural-Network/
- **版权声明：** 本博客所有文章除特别声明外，均采用 CC BY-NC-SA 3.0 许可协议。转载请注明出处！

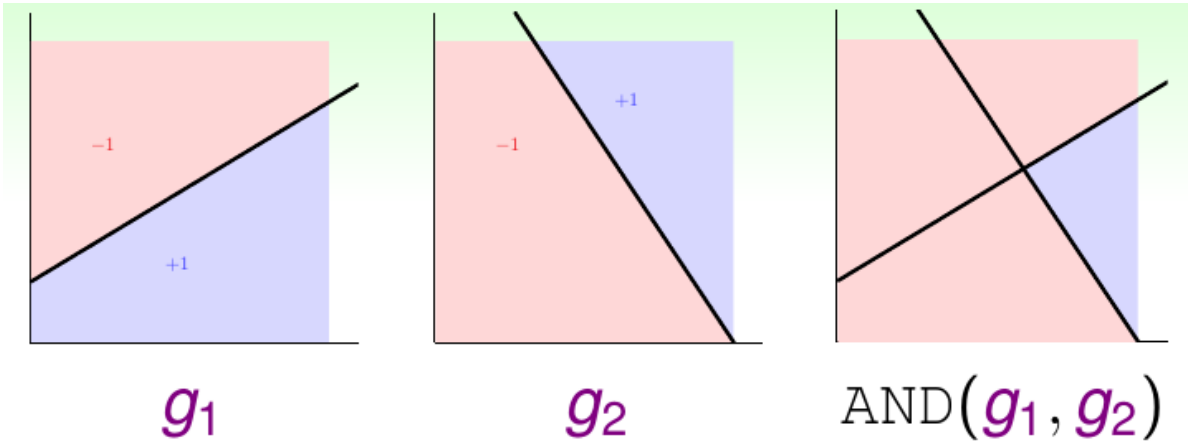
动机

本课在很早之前就讲过了感知机的概念，前面又讲过了模型聚合的概念，那么有没有一种方法可以将若干个感知机模型通过线性的方式聚合起来？显然是可以的。假设一共有t个感知机，每个感知机训练出来的系数为$\mathbf{w}_t$，返回的模型为g_t，其在整个模型里的权重为α_t，则最后得到的聚合模型G为

$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \underbrace{\text{sign}(\mathbf{w}_t^{\mathsf{T}}\mathbf{x})}_{g_t(\mathbf{x})}\right)$$

可以看到这个过程里有两组权重：第一组$\mathbf{w}_t$用来投票，第二组α_t用来让g_t投票。另外，这里有两组阶梯函数的输出，一组由每个g_t产生，一组由最后的G产生

将感知机聚合以后，可以实现某些布尔函数。下图给出了AND函数的例子



布尔函数AND的示意图

在这种情况下，只有被g_1和g_2全部判断为+1的例子，才会被G判断为+1，否则就会判断为-1。这种分类器不难实现，只需要使得

$$G(\mathbf{x}) = \mathrm{sign}(-1 + g_1(\mathbf{x}) + g_2(\mathbf{x}))$$

就可以学习出上图中由两条直线界定出的边界，达到AND函数的效果。同理，OR和NOT的效果也可以用类似的方法产生。实际上，任意多的感知机聚合得到的模型能力非常强大，它可以逼近凸包集合（注意凸包集合的VC维是∞，是一种能力很强的模型），也可以逼近一个平滑的边界，例如一个圆形的边界

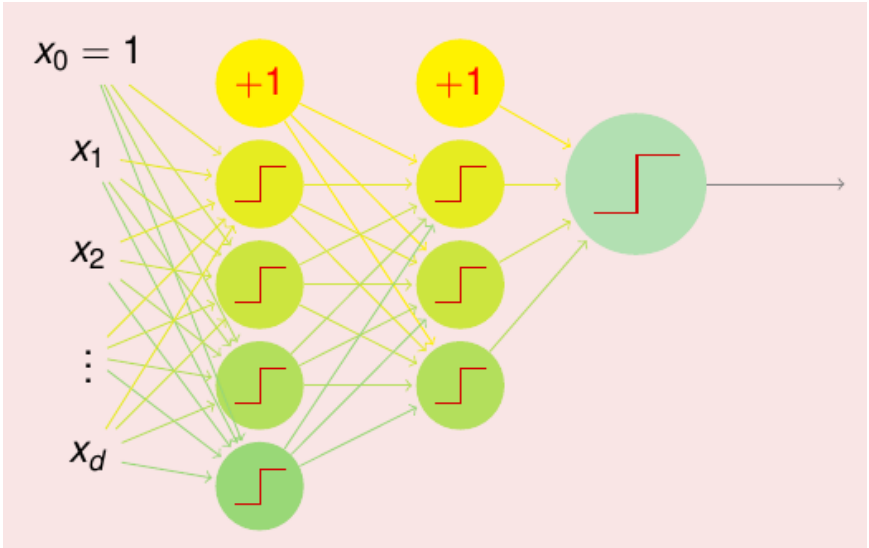
但是，感知机的线性组合不能实现XOR函数，这是一个最大的限制。其本质原因是，如果把g_1和g_2看作是特征转换，则在新的空间$\phi(g_1(\mathbf{x}), g_2(\mathbf{x}))$中，经过转换的数据集仍然不是线性可分的

那么怎么以感知机为基础去实现XOR函数呢？回顾之前所讲的，非线性可分的数据集可以通过某些变换使其线性可分，这个道理在这里同样适用。将XOR函数转写，有

$$\mathrm{XOR}(g_1, g_2) = \mathrm{OR}(\mathrm{AND}(-g_1, g_2), \mathrm{AND}(g_1, -g_2))$$

由于AND函数可以用感知机线性聚合来实现，OR函数也可以用感知机线性聚合来实现，因此通过上面的方法就可以以感知机为基础实现XOR了，只不过此时多了一**层**。这种更强大的模型称为**多层感知机**，它是后面提到的所有神经网络的基础

多层感知机实际上是对生物钟神经元工作原理的模拟。这里每个感知机就可以看作是一个神经元，神经元的系数可以看做是神经元之间的连接，系数绝对值越大，连接越强。神经元之间通过连接，就形成了一个网络结构，称为**神经网络** 。如下就给出了一个多层感知机结构的示意图



多层感知机结构示意

上图中每个圆圈就是一个感知机，圆圈上的的折线表示做一个非线性变换（这里是sign函数，折线也是sign函数的图像），圆圈之间的连线以及圆圈与x_i的连线表示了输入输出关系

神经网络假设集合

如果将每一层感知机做的事情看作是对原始数据的一个变换，记第一层感知机做的变换为$\boldsymbol{\phi}^{(1)}$，第二层感知机做的变换为$\boldsymbol{\phi}^{(2)}$，最后大绿色圆对应的感知机实际上就是输出数据结果，记其权重为$\mathbf{w}$，则整个模型可以表示为$s = \mathbf{w}^{\mathsf{T}}\boldsymbol{\phi}^{(2)}(\boldsymbol{\phi}^{(1)}(\mathbf{x}))$。可见，其外核实际上就是一个线性模型，这个线性模型可以根据要解决的问题选择不同的具体实现，例如线性分类模型（感知机）、线性回归或者Logistic回归。这里为了简单起见，将选取线性回归做线性模型而讲解（即最后一步绿圆圈使用线性回归做输出）

另外，中间的非线性变换是对分数s的变换。如果这个变换是阶梯状函数（例如sign），那么每个神经元就是一个感知机。可不可以将这个变换函数也做一些修改呢？变成线性函数是不太好的，因为这样会使整个网络退化为一个线性变换（线性变换的线性组合还是线性变换），增加了计算复杂度，但是降低了模型复杂度。实际上，由于阶梯状函数是离散函数，难以最优化，因此通常使用s形函数，尤其是双曲正切函数$\tanh(x)$，其定义为

$$\tanh(x) = \frac{\exp(x) - \exp(-x)}{\exp(x) + \exp(-x)}$$

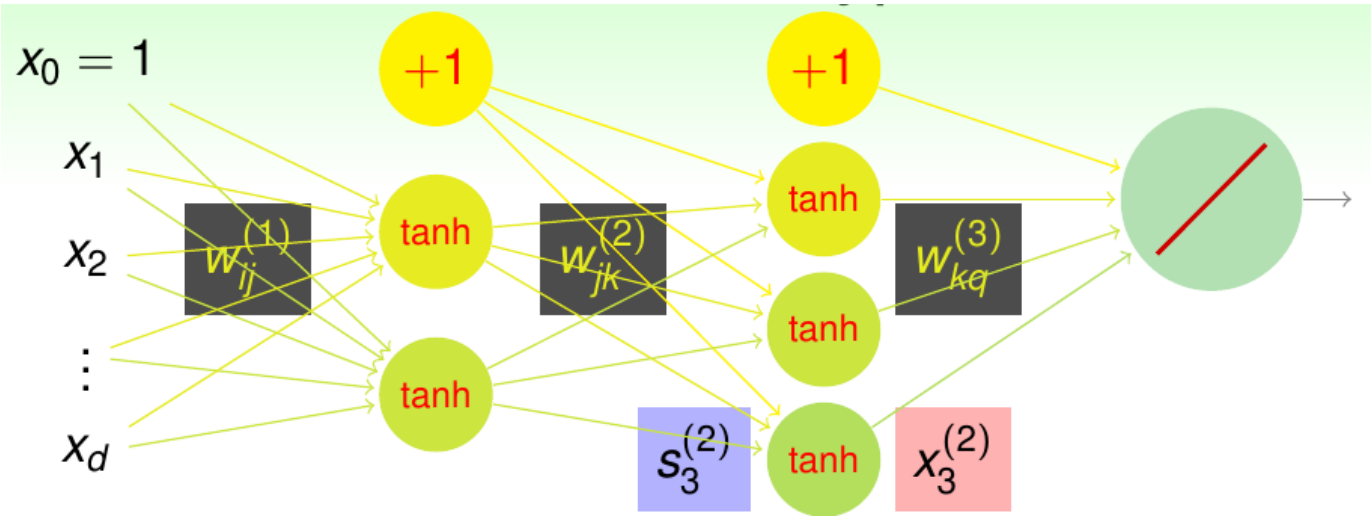
可以把它看作是sign函数的模拟版本，但是更容易最优化，而且更类似于生物中神经元的工作原理

$\tanh$函数与之前提到的sigmoid函数也有关系：

$$\tanh(x) = 2\theta(2x) - 1$$

之后的讲解中，中间的非线性变化会使用$\tanh$函数

接下来，为了更好地定义神经网络假设集合，需要对网络中出现的各个权重和得分进行定义。总体可见下图



神经网络示意图（带变量）

其中，L代表了神经网络总的层数。输入层记为第0层，第一步变换称为第1层，以此类推。$d^{(\ell)}$为网络中第ℓ层所含有的总节点数。神经元之间每个连接其本质是一个权重，可以记为$w_{ij}^{(\ell)}$，其中$1 \le \ell \le L$指明该权重连进第ℓ层；$0 \le i \le d^{(\ell-1)}$是第$\ell-1$层神经元的序号，从0开始标记（0对应的是常数项，也就是截距），可以看作是输入索引；$1 \le j \le d^{(\ell)}$是第ℓ层神经元的序号，从1开始标记，可以看作是输出索引。换句话说，$w_{ij}^{(\ell)}$表示第$\ell-1$层第i个单元指向第ℓ层第j个单元的权重

第ℓ层中，第j个节点送进其变换函数之前的值，是上一层所有节点的加权得分总和，记为$s_j^{(\ell)}$。由上一段的定义，有

$$s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} x_i^{(\ell-1)}$$

其中$x_j^{(\ell)}$是得分经过变换后得到的值（如果$\ell = 0$，则对应原始数据中的一个特征），有

$$x_j^{(\ell)} = \begin{cases} \tanh\left(s_j^{(\ell)}\right) & \text{if } \ell < L \\ s_j^{(\ell)} & \text{if } \ell = L \end{cases}$$

称中间使用非线性变换的层为**隐含层**（更确切地说，是输出结果被其它层当做输入的层），则神经网络假设集合做的事情就是将原始$\mathbf{x}$作为输入层$\mathbf{x}^{(0)}$，经过中间隐含层的一系列计算得到$\mathbf{x}^{(\ell)}$，最后由输出层做预测得到$x_1^{(L)}$的过程

从物理意义上看，中间各层实际就是对数据做各种变换$\boldsymbol{\phi}^{(\ell)}(\mathbf{x})$，

$$\boldsymbol{\phi}^{(\ell)}(\mathbf{x}) = \tanh\left(\begin{bmatrix} \sum_{i=0}^{d^{(\ell-1)}} w_{i1}^{(\ell)} x_i^{(\ell-1)} \\ \vdots \end{bmatrix}\right)$$

当x与w接近（平行）时，wx的值更大，tanh的结果越接近+1。换句话说，此时$\mathbf{x}$与权重向量$\mathbf{w}$在模式上更加匹配。即神经网络中的每一层都在做**模式提取**，这种模式是跟各层权重吻合的模式

具体的变换方式是从数据中学习得到的，下一节就会介绍具体的学习方法

神经网络学习

前面讲述了神经网络结构中每个权重$w_{ij}^{(\ell)}$是怎么产生，那么一个很自然的问题就是，如何学习出最优的$\{w_{ij}^{(\ell)}\}$，使$E_{\text{in}}\left(w_{ij}^{(\ell)}\right)$最小。如果只有一个隐藏层，整个神经网络就会退化为感知机的聚合模型，因此可以使用梯度提升来确定隐藏层的权重。但是如果有多个隐藏层，问题就没有这么简单了。但是换一个角度思考，对于每一条输入$\mathbf{x}_n$，当神经网络确定以后，它的输出就是确定的，记为$\text{NNet}(\mathbf{x}_n)$，则预测输出的误差也可以确定。由前面的说明，这里使用平方误差衡量，所以误差函数为$e_n = (y_n - \text{NNet}(\mathbf{x}_n))^2$，这个值也被称为残差。由于神经网络的输出跟每个权重之间都存在变化关系，因此可以计算$\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$，进而可以使用前面讲过的梯度下降法（GD）或者随机梯度下降法（SGD）做最优化。所以接下来要解决的问题，就是如何计算$\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$

首先看一个最简单的情况：对神经网络最后一层中的权重$w_{ij}^{(L)}$，其与残差之间的关系。注意最后一层只有一个节点，因此j肯定为1。而且最后一层的节点按照前面的约定仅做一个线性变换，因此其输出就是$s_1^{(L)}$。这样一来，可以将残差项展开，即

$$\begin{aligned} e_n &= (y_n - \text{NNet}(\mathbf{x}_n))^2 = \left(y_n - s_1^{(L)}\right)^2 \\ &= \left(y_n - \sum_{i=0}^{d^{(L-1)}} w_{i1}^{(L)} x_i^{(L-1)}\right)^2 \end{aligned}$$

因此，对于神经网络的最外层，要计算$\frac{\partial e_n}{\partial w_{i1}^{(L)}}\left(0 \le i \le d^{(L-1)}\right)$，可以借助中间变量$s_1^{(L)}$，使用链式法则

$$\frac{\partial e_n}{\partial w_{i1}^{(L)}} = \frac{\partial e_n}{\partial s_1^{(L)}} \cdot \frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}} = -2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right)$$

推而广之，对于前面的隐含层$\ell, 1 \le \ell < L$，$\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}\left(0 \le i \le d^{(\ell-1)}; 1 \le j \le d^{(\ell)}\right)$可以通过类似的方法计算：

$$\frac{\partial e_n}{\partial w_{ij}^{(\ell)}} = \frac{\partial e_n}{\partial s_j^{(\ell)}} \cdot \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}}$$

其中乘数可以很容易地得到为$x_i^{(\ell-1)}$，但是被乘数的计算并不直观。暂且将被乘数记为$\delta_j^{(\ell)}$，上面的式子可以写为

$$\frac{\partial e_n}{\partial w_{ij}^{(\ell)}} = \frac{\partial e_n}{\partial s_j^{(\ell)}} \cdot \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} = \delta_j^{(\ell)} \cdot \left(x_i^{(\ell-1)}\right)$$

当$j = 1, \ell = L$时，由上面的推导有$\delta_1^{(L)} = -2\left(y_n - s_1^{(L)}\right)$。接下来的任务就是推出其它的$\delta$

由于δ是最终残差对神经网络中隐藏层某个节点给出的得分的偏导数，因此可以思考这个得分是如何影响残差的。某个节点得到得分$s_j^{(\ell)}$以后，会对其做非线性变换tanh，得到节点的输出$x_j^{(\ell)}$。这个输出乘以下一层的各个权重$w_{jk}^{(\ell+1)}$，得到的是一个向量，向量中的每一维都对应了下一层每个节点的一个输出，即

$$s_j^{(\ell)} \xrightarrow{\tanh} x_j^{(\ell)} \xrightarrow{w_{jk}^{(\ell+1)}} \begin{bmatrix} s_1^{(\ell+1)} \\ \vdots \\ s_k^{(\ell+1)} \\ \vdots \end{bmatrix} \Longrightarrow \cdots \Longrightarrow e_n$$

因此求偏导时使用的链式法则里就需要包含$x_j^{(\ell)}$和$s_k^{(\ell+1)}$这两组中间变量（其中后一组一共有$d^{(\ell+1)}$个变量）。具体计算，有

$$\begin{aligned}\delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}} &= \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \cdot \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \cdot \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} \\ &= \sum_k \left(\delta_k^{(\ell+1)}\right)\left(w_{jk}^{(\ell+1)}\right)\left(\tanh'\left(s_j^{(\ell)}\right)\right)\end{aligned}$$

因此，第ℓ层的$\delta_j^{(\ell)}$可以从第$\ell+1$层的$\delta_k^{(\ell+1)}$推出。将以上推导汇总起来，就可以得到神经网络的学习算法，称为**反向传播算法**，其具体过程为

> 初始化所有权重$w_{ij}^{(\ell)}$
>
> 在时刻$t = 0, 1, \dots, T$
>
> 1. 随机选取某个$n \in \{1, 2, \dots, N\}$
> 2. 正向：计算所有$x_i^{(\ell)}$，其中$\mathbf{x}^{(0)} = \mathbf{x}_n$
> 3. 反向：计算所有$\delta_j^{(\ell)}$
> 4. 梯度下降：$w_{ij}^{(\ell)} = w_{ij}^{(\ell)} - \eta x_i^{(\ell-1)} \delta_j^{(\ell)}$
>
> 返回$g_{\text{NNET}}(\mathbf{x}) = \left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_i\right)\right)\right)$

注意有时第1步到第3步是并行地执行很多次，然后将得到的所有数值求出平均$\left(x_i^{(\ell-1)}\delta_j^{(\ell)}\right)$来在第4步更新权重。这种方法介于GD和SGD之间，称为mini-batch GD

优化与正则化

前面讲解神经网络反向传播算法时，使用的误差函数是平方误差函数。将这种情况推广，可以知道神经网络一般是要最小化如下的E_{in}

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \text{err}\left(\left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_i\right)\right)\right), y_n\right)$$

但是，当有多个隐藏层存在时，这通常不是一个凸优化问题，GD/SGD很难找到全局最优解，只能找到局部最优解。而且，对$w_{ij}^{(\ell)}$不同的初始化可能会达到不同的局部最优解，而且并不知道如何初始化能让最后得到的解更好。但是，根据经验，大的初始权重会使得$\tanh$函数的值变得很大，导致梯度（变化量）很小，出现饱和（saturate）现象。因此，一般都是用一些随机的，小的值做初始权重

神经网络是一种很强大的算法。假设V是神经元的数目，D是神经网络中权重的数目（也就是神经元之间连接的数目），那么整个神经网络的VC维约为$O(VD)$。当V很大时，神经网络几乎可以模拟任意函数，但是也就更容易过拟合

神经网络避免过拟合有一种很基本，也可以看作是“老朋友”的方法，就是加入正则化。直观的方法是加入L2正则化项$\Omega(\mathbf{w}) = \sum\left(w_{ij}^{(\ell)}\right)^2$，但是这种权重缩减方法的结果是，大的权重缩减得大，小的权重缩减得小，最后哪个权重都没有缩减到0。而这里做正则化的目的是希望得到稀疏的$\mathbf{w}$，也就是让尽量多的$w_{ij}^{(\ell)}$为0，这样才能有效地减小整个神经网络的VC维。提到使权重稀疏的正则化方法，另一个老朋友是L1正则化$\sum\left|w_{ij}^{(\ell)}\right|$，但是这个正则化项是不可微的，而反向传播是微分导向的算法，因此不能使用L1正则化

通常情况下，神经网络里使用了一种称为“权重淘汰”（weight-elimination）的方法，其实质是一种加权的L2正则化，它将大的权重和小的权重都缩减相同的大小，这样小的权重就会被缩减到0。权重淘汰正则项的形式为

$$\sum \frac{\left(w_{ij}^{(\ell)}\right)^2}{1 + \left(w_{ij}^{(\ell)}\right)^2}$$

神经网络里其实还有一种自己独有的正则化机制，称为“提前停止”法。由于神经网络算法的核心还是GD/SGD，而这种算法随着迭代次数的增多，其在各种不同的w里选择得越多。如果降低这种算法的迭代次数，它看过的w就会更少。因此，迭代次数多，有效的VC维就大；反之，迭代次数少，有效的VC维就小。这意味着**比较小的迭代次数可以有效降低VC维**，而合适的迭代次数可以使用前面讲过的验证法做判断。这种方法对所有基于GD/SGD的算法都适用

NTUML 29. 深度学习

- **本文作者：** Tingxun Shi
- **本文链接：** http://txshi-mt.com/2017/10/21/NTUML-29-Deep-Learning/
- **版权声明：** 本博客所有文章除特别声明外，均采用 CC BY-NC-SA 3.0 许可协议。转载请注明出处！

本讲原标题有点标题党，只讲了一点关于深度学习的皮毛，还有一部分在讲PCA，不要被误导!

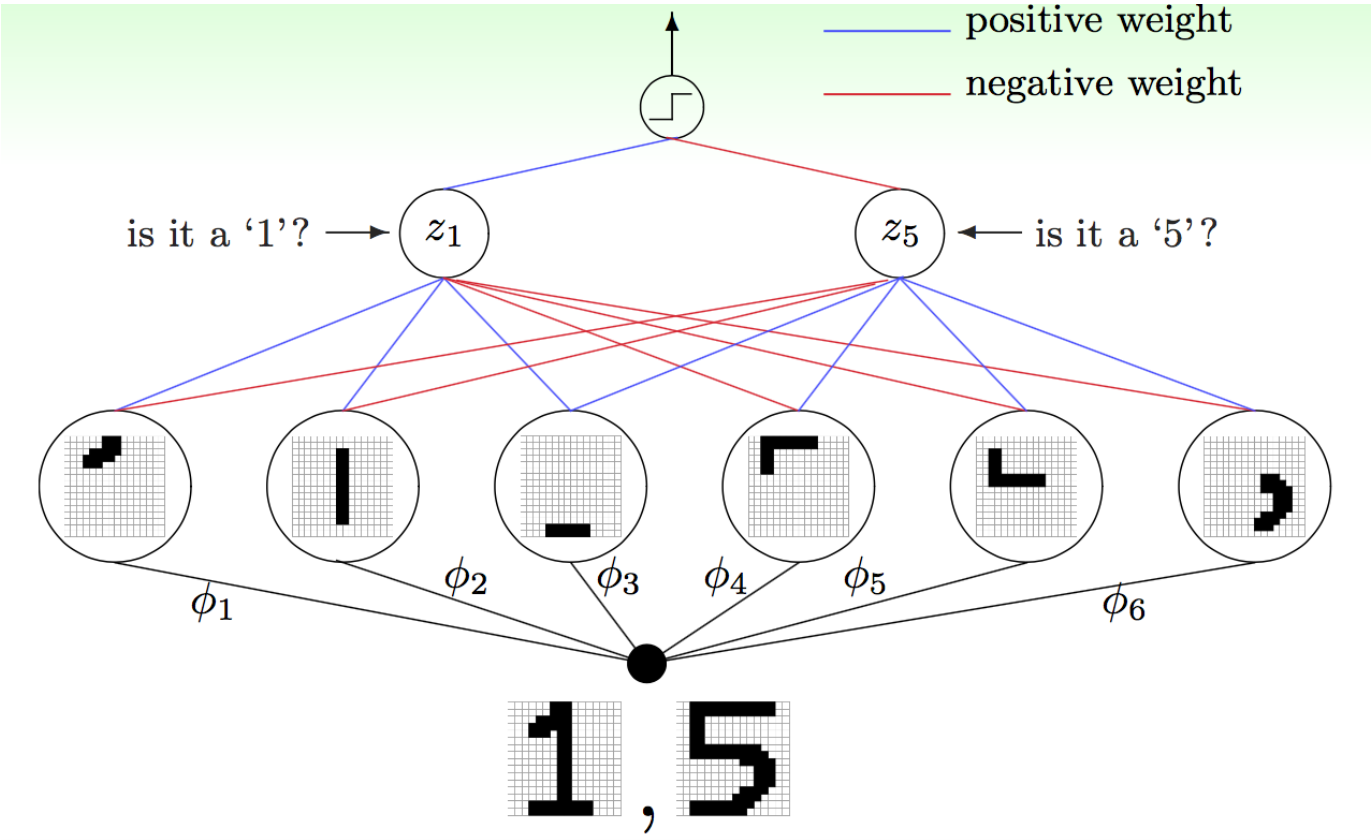
本讲原标题有点标题党，只讲了一点关于深度学习的皮毛，还有一部分在讲PCA，不要被误导！！

本讲原标题有点标题党，只讲了一点关于深度学习的皮毛，还有一部分在讲PCA，不要被误导！！！

深度神经网络

神经网络的核心是一层层的神经元和它们之间的连接关系。一个比较直接的问题，是应该如何设计神经网络？它应该有多少层，每一层各自应该有多少神经元？或者更泛泛地说，应该选择什么样的网络结构？这是神经网络应用时一个非常核心也非常困难的问题。神经网络结构可以粗浅地分成两种：浅层神经网络和深度神经网络，两者区别是前者只有少量的隐藏层，而后者有很多层。浅层神经网络训练起来更有效，关于结构的决定更简单，有足够有力的理论基础。相比而言，深度学习训练起来比较难，也难以决定网络的结构。但是当层数足够多时，网络的能力也会非常强，而且它可以提取出有物理意义的特征

例如，现在要分辨手写的数字是1还是5，那么可以先想办法在笔记里萃取出各个部位的笔画特征，例如中间是竖，底层是不是弯等等。这些特征（笔画）可能通过组合变得更复杂，到最后一步再用来做辨识动作，如下图所示



深度学习示例：手写数字识别

这里实际每一层都可以解读出物理意义。例如要辨识1，可能需要ϕ_1, ϕ_2, ϕ_3，不需要ϕ_4, ϕ_5, ϕ_6。这样看，第一层其实做的是从像素点提取笔画的功能，第二层则是对第一层做组合，设计不同权重，等等。层数越多，就能表示越多种不同的变化，每层要做的事情也相对来讲变得更加简单。在这种架构下，每一层只需要贡献出一份微薄的力量，做一点微小的工作，就可以使整个网络完成从简单特征到复杂特征的变换。因此，深度学习适用于那些难以从原始特征做学习的任务，例如语音处理

前面提到过深度学习会面临几个难题，目前（2015年）主要采取如下的方式来应对

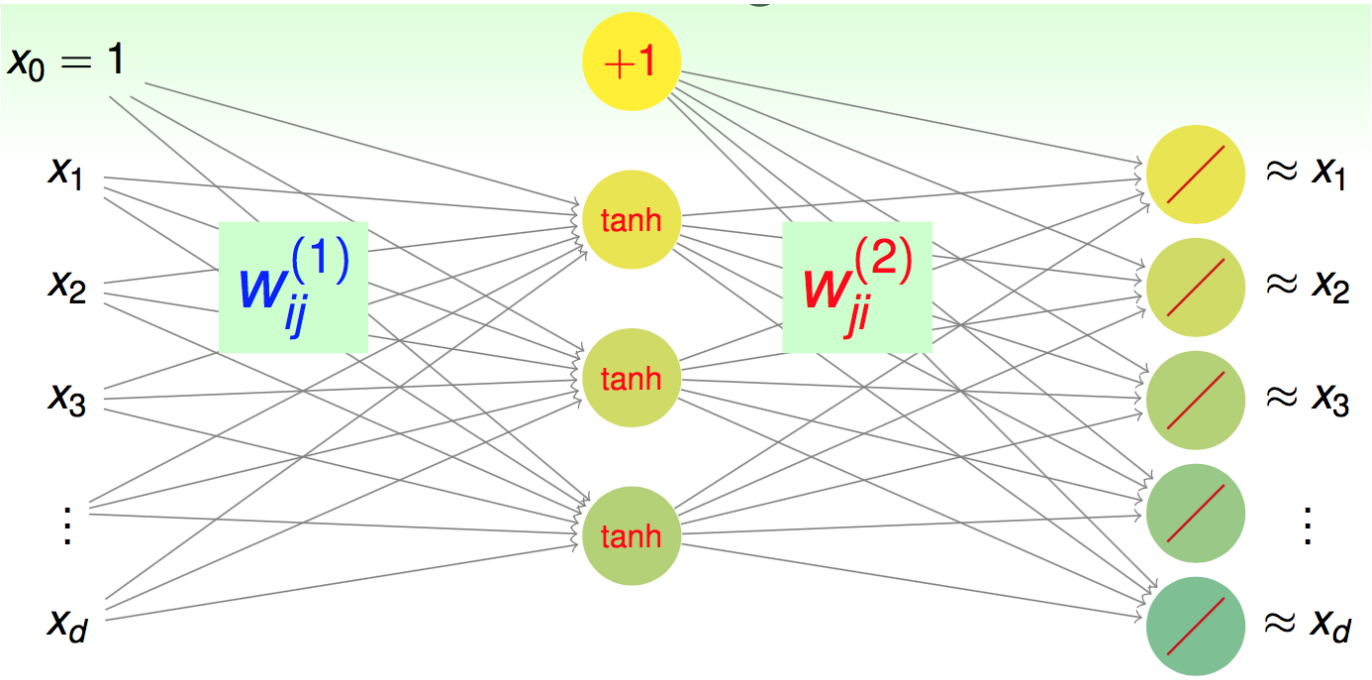
- 难以决定具体应使用何种网络结构：使用一些领域知识，加入对问题的了解。例如处理图像时使用卷积神经网络（Convolutional NN，CNN），这样让每层神经元都只处理一小块像素
- 模型复杂度很高：如果数据足够大，就不是问题。如果的确想控制模型复杂度，想容忍噪声，就使用一些特有的正则化方法，例如dropout（训练时随机丢弃一部分隐藏层神经元）或者降噪（denoising，后面介绍）
- 难以最优化：谨慎选择初始权重，以避免陷入差的局部最优解（称为预训练）
- 过高的计算复杂度（尤其是在大数据背景下此问题更严重）：使用新的硬件和体系结构，例如使用GPU做mini-batch

本讲无法涵盖所有深度学习的技巧，只能介绍一点入门的部分。这里再多说几句，介绍一种比较简单的预训练方法。不像之前提到的神经网络那样上来对所有权重做决定，这里使用了一种分层决定的方法：对第$\ell = 1, \ldots, L$层，假设$w_*^{(1)}, \ldots, w_*^{(\ell-1)}$都已经固定，一层一层去对$\{w_{ij}^{(\ell)}\}$做预训练。每层权重都预训练好了以后，再使用反向传播来调优所有的$\{w_{ij}^{(\ell)}\}$。接下来的问题就是怎么在每一层进行预训练，以及怎么加入正则化来控制模型复杂度

自动编码器

前面讲过，预训练的意义是寻找一个好的初始化权重，那么什么叫"好的"初始化权重呢？这里引发了我们对权重的物理意义的思考。由前面的讲解，可知权重本质上是告知模型如何做特征变换，或者说，怎么把原始数据换一种表示形式（称为经过了一次**编码**）。做深度学习时，并不知道初始化（预训练得到）的权重在后面会被如何调整。既然如此，不如希望预训练后得到的权重能够（精炼地）保持原始数据的特征：传给后面神经元的数据不是乱七八糟的，而是对相同信息的一种不同表示。例如，对前面的手写数字例子，第一层提取出的笔画特征经过组合，可以还原出原始的数据。那么什么是维持原来数据的特征变换呢？如果使用变换后的数据可以轻易重建原来的数据，就说明原始数据的信息没有什么损失。也就是说，我们希望深度学习预训练时得到的特征变换（初始权重）都是能保持原始信息的

那么如何得到这种能保持原始信息的特征变换呢？一种做法就是仍然使用一个神经网络来解决：把原始的数据通过变换送入到隐藏层神经元以后，这些神经元的输出经过一些变换还能还原成原始的输入（与原始的输入类似）。这个神经网络g有一个独特的名字，称为**自动编码器**，它是一个$d - \tilde{d} - d$三层的神经网络，从输入层到隐藏层是编码操作，从隐藏层到输出层是解码操作，保证最后输出$g_i(\mathbf{x}) \approx \mathbf{x}_i$（也就是说，此网络是学习逼近一个恒等函数）。其中$\tilde{d}$是编码的维度。整个自动编码器的结构如下图所示



自动编码器示意

其中$w_{ij}^{(1)}$称为编码权重，$w_{ji}^{(2)}$称为解码权重

那么为什么要设计这么一个复杂的结构，费劲去逼近一个恒等函数（相当于一个什么也不做的函数）？如果真的能得到$g(\mathbf{x}) \approx \mathbf{x}$，则它必然依赖了原始数据一些*隐藏的*特征/结构。如果能通过这个学习过程得到隐藏的特征/结构，那么就可以把这些特征/结构当做特征变换，用在预训练中。这些特征/结构告诉我们如何用一些有效的信息表示原始数据。对于非监督学习问题，自动编码器也有其实际意义。例如如果要做密度估计问题，则对于密度大的区域，编码器能学习得比较好，会更有$g(\mathbf{x}) \approx \mathbf{x}$。那么对应的，如果对于新的数据有$g(\mathbf{x}_n) \approx \mathbf{x}_n$，就说明它落在了密度大的区域。如果要做离群点检测，就可以通过$g(\mathbf{x}_n) \not\approx \mathbf{x}_n$来找出离群点。也就是说，学习数据的隐藏特征/结构，也就是在学习什么样的数据是"典型的"数据，自动编码器可以判断数据典型与否

综上所述，自动编码器的目标看似只是学好恒等函数，实际它只是表象。重要意义是看隐含层，看数据有效的表示方式

如前面所述，自动编码器就是一个结构为$d - \tilde{d} - d$，使用$\sum_{i=1}^{d}(g_i(\mathbf{x}) - x_i)^2$做误差函数的神经网络。它是一种浅层神经网络，因此容易训练。而且，通常$\tilde{d} < d$，以保证学到的是原始数据的*压缩*表示。构造的数据集是$\{(\mathbf{x}_1, \mathbf{y}_1 = \mathbf{x}_1), (\mathbf{x}_2, \mathbf{y}_2 = \mathbf{x}_2), \ldots, (\mathbf{x}_N, \mathbf{y}_N = \mathbf{x}_N)\}$（不看数据集中的标签，因此通常这个学习过程被看做是无监督学习的过程）。有时，还会加入限制条件$w_{ij}^{(1)} = w_{ji}^{(2)}$做正则化，不过这样会使最后计算梯度的过程更加复杂

所以，回到上节最后，讲述深度学习如何初始化权重的方法。这个方法里提到说，初始化方法最有效的一招是每一层做一个预训练，这个预训练的过程就是用前一层的输出（对于第一层，是用输入数据）$\{\mathbf{x}_n^{(\ell-1)}\}$训练一个自动编码器，该自动编码器的隐藏层神经元数$\tilde{d}$与深度神经网络中第$\ell$层的节点个数$d^{(\ell)}$相等

使用不同的体系结构加上不同的正则化，可以得到更精巧的自动编码器，进而得到更好的预训练结果。不过其本质都是相同的

降噪自动编码器

前面讲述了如何在深度学习里做权重的预训练，这一节主要看一看如何加入一些独有的正则化技巧。前面提到过，深度学习里用到的正则化方法包括对结构做出一些限制、加入与权重有关的正则项以及提早结束训练等。但是考虑过拟合的成因，如果数据中噪声太大，也会造成过拟合。因此还可以再独辟蹊径，找到一种方法去降低噪声，这样也可以降低过拟合的风险。最简单的方法是直接做数据清洗，不过一种更疯狂的方法是向数据里加入噪声

在上一节里，自动编码器逼近的是恒等函数，即使得$g(\mathbf{x}) \approx \mathbf{x}$。但是一个足够鲁棒的自动编码器不仅能做到这一点，对于与输入有些微不同的$\tilde{\mathbf{x}}$，还能做到$g(\tilde{\mathbf{x}}) \approx \mathbf{x}$。也就是说，送进干净的数据可以产生干净的数据，而送进脏数据也可以出来干净的数据。这意味着足够鲁棒的自动编码器还能起到降噪，清洗数据的功能。因此，训练编码器时，可以故意往训练数据里加入一些人工噪声。即此时输入数据为$\{(\tilde{\mathbf{x}}_1, \mathbf{y}_1 = \mathbf{x}_1), (\tilde{\mathbf{x}}_1, \mathbf{y}_2 = \mathbf{x}_2), \ldots, (\tilde{\mathbf{x}}_N, \mathbf{y}_N = \mathbf{x}_N)\}$ 其中$\tilde{\mathbf{x}}_n = \mathbf{x}_n + \text{人工噪声}$。通过这种方法训练出来的自动编码器一般称为**降噪自动编码器**，在深度学习中通常是使用它，而不是原始的自动编码器。加入人工噪声这种方法告诉了算法我们所需要的性质，得到的模型对噪声容忍度有了很大的增强，因此这种手段也可以看作是一种正则化机制，可以适用到其它模型训练的过程中

主成分分析（PCA）

前面提到的自动编码器实际上是一种非线性模型，但是前面的课程一直都在说模型应该从最简单的，线性的模型试起，为什么这次反其道而行之了？原因是线性编码器是在深度学习预训练权重的背景下提出，而深度学习本身就是一个非线性的问题，因此在这个背景下直接讨论非线性的解决方案更有意义

当然，自动编码器也存在其线性形式，而且线性的自动编码器也有很多适用场合，因此仍然有研究的必要。线性自动编码器类似于前面提到的非线性自动编码器，只不过在对输入$\mathbf{x}$求出得分以后不需要再做非线性变化$\tanh$。这样一来，对$\mathbf{x}$的第k个维度，其假设函数形式可以写为

$$h_k(\mathbf{x}) = \sum_{j=0}^{\tilde{d}} w_{kj} \left(\sum_{i=1}^{d} w_{ij} x_i \right)$$

注意内层求和项的指标是从1开始计数，因为对常数项逼近一个恒等函数没什么意义。此外，这里加入了前面所提到的正则化手段，即编码器和解码器的权重相等，$w_{ij}^{(1)} = w_{ji}^{(2)}$，这里统一写成了$w_{ij}$。以及，为了避免产生非平凡解，且考虑到实际要得到的是原始数据的*压缩*表示，还限制编码后数据的维度$\tilde{d}$要小于原始维度d。这样，记所有权重组成的矩阵为W，线性自动编码器假设函数的形式为

$$h(\mathbf{x}) = \mathrm{W}\mathrm{W}^{\mathsf{T}}\mathbf{x}$$

既然假设函数的形式已经被定义好，那么接下来的做法似乎水到渠成：只需要按部就班写出损失函数，就可以让损失函数对W求偏导，找出最优的W。损失函数的形式为（由于最后求出来的是一个向量，因此下面公式中用了黑体$\mathbf{h}$）

$$E_{\text{in}}(\mathbf{h}) = E_{\text{in}}(\mathrm{W}) = \frac{1}{N} \sum_{n=1}^{N} \left\| \mathbf{x}_n - \mathrm{W}\mathrm{W}^{\mathsf{T}}\mathbf{x}_n \right\|^2, \quad \mathrm{W} \in \mathbb{R}^{d \times \tilde{d}}$$

前面讲到的线性问题都可以得到一个解析解，但是这里乍看之下并不容易，因为求和项里实际上是w_{ij}的四次多项式，求解起来可能要费点周折，需要使用一些线性代数里的工具。由于$\mathrm{W}\mathrm{W}^{\mathsf{T}}$是半正定矩阵，因此可以对它做特征值分解，记为$\mathrm{W}\mathrm{W}^{\mathsf{T}} = \mathrm{V}\Gamma\mathrm{V}^{\mathsf{T}}$。这里得到的$\mathrm{V}$是正交矩阵，即$\mathrm{V}\mathrm{V}^{\mathsf{T}} = \mathrm{V}^{\mathsf{T}}\mathrm{V} = \mathrm{I}_d$，而$\Gamma$是一个对角矩阵，由特征值组成，对角线上非零元素的数量至多为$\tilde{d}$个。因此$\mathrm{W}\mathrm{W}^{\mathsf{T}}\mathbf{x}_n = \mathrm{V}\Gamma\mathrm{V}^{\mathsf{T}}\mathbf{x}_n$，接下来可通过优化$\mathrm{V}$和$\Gamma$来最小化$E_{\text{in}}$

在进入下一步之前，先看一下$\mathrm{W}\mathrm{W}^{\mathsf{T}}\mathbf{x}_n = \mathrm{V}\Gamma\mathrm{V}^{\mathsf{T}}\mathbf{x}_n$的几何意义。这里原数据左乘一个正交矩阵$\mathrm{V}^{\mathsf{T}}\mathbf{x}_n$实际上就是对原数据做一个坐标变换，类似于某种旋转或镜像。再左乘Γ，由于Γ是对角矩阵且只有至多$\tilde{d}$个非零元素，因此这一步是将原数据至少$d - \tilde{d}$个维度的分量置为0，然后对其他分量做缩放，得到一个新的坐标。最后再乘V，就是将新的坐标变换到原来的坐标系。因此，类似地，如果对原数据不做任何修改，那么有$\mathbf{x}_n = \mathrm{V}\mathrm{I}\mathrm{V}^{\mathsf{T}}\mathbf{x}_n$。原问题就变成了

$$\min_{\mathrm{V}} \min_{\Gamma} \frac{1}{N} \sum_{n=1}^{N} \left\| \mathrm{V}\mathrm{I}\mathrm{V}^{\mathsf{T}}\mathbf{x}_n - \mathrm{V}\Gamma\mathrm{V}^{\mathsf{T}}\mathbf{x}_n \right\|^2$$

首先来看最优的Γ。由于乘以V和V^{T}是旋转和镜像，不改变向量的长度，因此去掉它对最优化问题的解没有影响。即上述问题等价于

$$\min_{\Gamma} \sum \|(\mathrm{I}-\Gamma)(\text{some vector})\|^2$$

这里Γ是唯一的变量，因此为了让上式小，就要让$\mathrm{I}-\Gamma$中有尽量多的0。由于Γ是对角矩阵，I是单位矩阵，那么为了达到效果，就要让Γ的对角线上有尽量多的1。由前面的推导，Γ上最多有$\tilde{d}$个1。不失一般性地，最优的Γ可以为

$$\Gamma = \begin{bmatrix} \mathrm{I}_{\tilde{d}} & 0 \\ 0 & 0 \end{bmatrix}$$

这样，求解原问题等价于求解如下问题

$$\min_{\mathrm{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{I}_{d-\tilde{d}} \end{bmatrix} \mathrm{V}^{\mathsf{T}} \mathbf{x}_n \right\|^2$$

直观地看，这个问题是在问“对向量$\mathrm{V}^{\mathsf{T}}\mathbf{x}_n$，留下哪几个分量可以使剩下的向量范数最小”。由于原向量是固定的，因此实际上这个问题等价于“对向量$\mathrm{V}^{\mathsf{T}}\mathbf{x}_n$，拿走哪几个分量可以使拿走的向量范数最大”。也就是说，有

$$\min_{\mathrm{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{I}_{d-\tilde{d}} \end{bmatrix} \mathrm{V}^{\mathsf{T}} \mathbf{x}_n \right\|^2 \equiv \max_{\mathrm{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} \mathrm{I}_{\tilde{d}} & 0 \\ 0 & 0 \end{bmatrix} \mathrm{V}^{\mathsf{T}} \mathbf{x}_n \right\|^2$$

考虑极端的情况，如果$\tilde{d}=1$，此时只与V^{T}的第一个行向量$\mathbf{v}^{\mathsf{T}}$有关。考虑V是正交矩阵，有$\mathbf{v}^{\mathsf{T}}\mathbf{v}=1$。因此可以得到如下最优化问题

$$\begin{aligned} \max_{\mathbf{v}} \quad & \sum_{n=1}^{N} \mathbf{v}^{\mathsf{T}} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} \mathbf{v} \\ \text{s.t.} \quad & \mathbf{v}^{\mathsf{T}}\mathbf{v} = 1 \end{aligned}$$

使用拉格朗日乘子，可知最优的$\mathbf{v}$满足

$$\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} \mathbf{v} = \lambda \mathbf{v}$$

（推导过程：引入拉格朗日乘子λ，可知优化问题转为$\mathcal{L} = \sum \mathbf{v}^{\mathsf{T}} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} \mathbf{v} - \lambda(1-\mathbf{v}^{\mathsf{T}}\mathbf{v})$。$\nabla \mathcal{L}_{\mathbf{v}} = 2\sum \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}} \mathbf{v} - 2\lambda \mathbf{v}$，令$\nabla \mathcal{L}_{\mathbf{v}} = 0$就可以得到上面这个关系）

而$\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^{\mathsf{T}}$是一个矩阵，如果记为$\mathrm{X}^{\mathsf{T}}\mathrm{X}$，则根据特征值与特征向量的定义，最优的$\mathbf{v}$实际上就是$\mathrm{X}^{\mathsf{T}}\mathrm{X}$的第一个特征向量。推而广之，最优的$\mathrm{V}=\{\mathbf{v}_j\}_{j=1}^{\tilde{d}}$就是$\mathrm{X}^{\mathsf{T}}\mathrm{X}$的前$\tilde{d}$个特征向量

因此，线性自动编码器就是要把$\mathrm{X}^{\mathsf{T}}\mathrm{X}$求出来，得到前$\tilde{d}$个特征向量的方向，就是应该投影，做特征变换的方向。算法如下

> 计算$\mathrm{X}^{\mathsf{T}}\mathrm{X}$的前$\tilde{d}$个特征向量$\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_{\tilde{d}}$
>
> 做特征变换$\boldsymbol{\Phi}(\mathbf{x}) = \mathrm{W}(\mathbf{x})$

由上面的推导，可知线性编码器保证输入在投影后的空间里范数总和最大。如果换一个说法，说输入在投影后的空间里方差（变化量）总和最大，那么就是统计学里**主成分分析** (PCA) 的做法。PCA算法与线性自动编码器算法大同小异，不过由于其关注的是变化量总和，而变化量实际上是数据相对于平均数的差的平方，因此需要先把数据做一个相对于平均值的变换。PCA算法总体过程如下

> 令$\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$，令$\mathbf{x}_n \leftarrow \mathbf{x}_n - \bar{\mathbf{x}}$
>
> 计算$\mathrm{X}^{\mathsf{T}}\mathrm{X}$的前$\tilde{d}$个特征向量$\mathbf{w}_1, \mathbf{w}_2, \dots, \mathbf{w}_{\tilde{d}}$
>
> 做特征变换$\boldsymbol{\Phi}(\mathbf{x}) = \mathrm{W}(\mathbf{x} - \bar{\mathbf{x}})$

自动线性编码器和PCA都是非常好用的线性维度缩减的工具，可以用来处理数据。不过实践中PCA更常用一些

NTUML 30. 径向基函数网络

- **本文作者：** Tingxun Shi

- **本文链接：** http://txshi-mt.com/2017/10/22/NTUML-30-RBF-Network/

径向基函数网络假设函数

前面讲使用高斯核的SVM时曾经提到过这种模型的一个性质：在高斯核的帮助下，这种模型可以在无限维的空间里学习到一个最大间隔超平面。该模型的定义如下

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n y_n \exp(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2) + b\right)$$

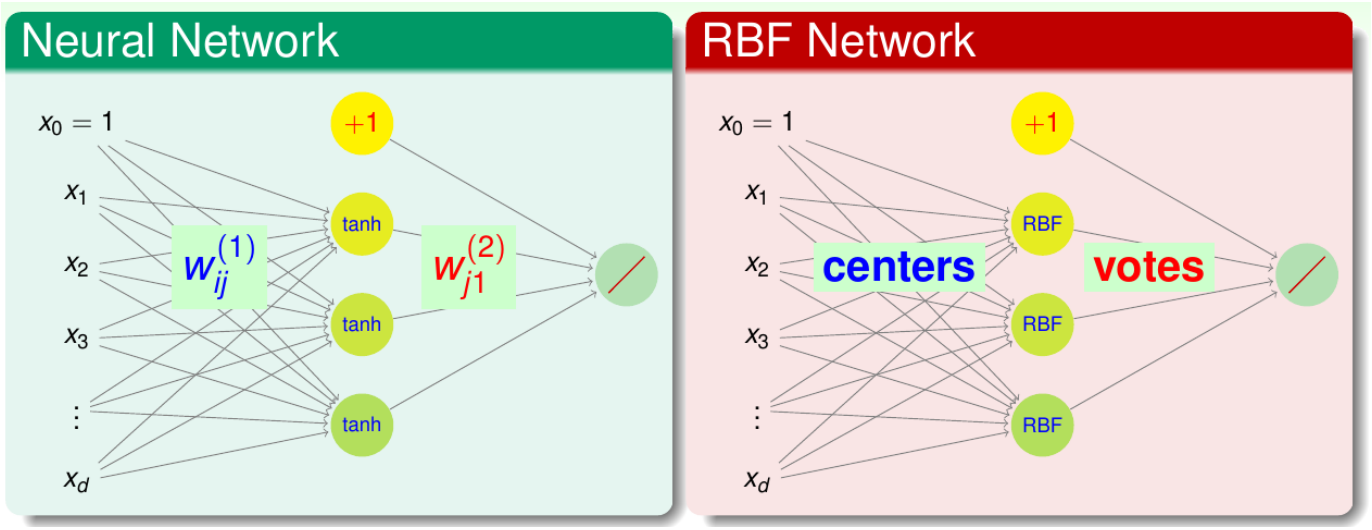
从最后得到的假设函数来看，它做的事情还可以表述为，使用一系列高斯核函数做线性组合，它们各自的中心就是在支持向量上。高斯核函数又称为径向基函数（Radial Basis Function，RBF），这个词可以做如下分解：

- “径向”表示要求的函数值与某个距离有关，这个距离是新的点$\mathbf{x}$和已知点$\mathbf{x}_n$之间的距离
- “基函数”表示这些函数要用来做线性组合。对于上式，系数就是$\alpha_n y_n$

如果定义$g_n(\mathbf{x}) = y_n \exp(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2)$，这个函数就表示新的点与已知点够不够近，越近给的票越多，这个票数就是y_n。这样一来，上面的SVM可以表示为

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n g_n(\mathbf{x}) + b\right)$$

这样更清楚地表明g_{SVM}其实就是g_n的线性组合，每个g_n代表一个径向假设。而径向基函数网络（RBF网络）就是这种模型的延伸。之所以说它是一个“网络”，就是因为其结构与之前讲过的神经网络有类似的地方。下图给出了两者的对比



神经网络与RBF网络的对比

可以看出，两者的输出相同，都是线性组合。不同的地方在隐含层：神经网络是计算权重和输入的内积，然后做非线性变换$\tanh$；而RBF网络则是计算两点之间的距离，然后使用高斯函数做变换。从历史上讲，RBF网络实际是神经网络的分枝

总体来看，RBF网络的假设函数可以写成如下形式

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m) + b\right)$$

其包括以下几个部分：

- 径向基函数RBF，注意高斯函数只是其中一种，可以有其它选择
- 中心点$\boldsymbol{\mu}_m$
- 投票β_m
- 输出转换Output

其中$\boldsymbol{\mu}_m$和β_m是两个关键变量

如果把高斯核SVM看作是RBF网络的一个特例，则存在以下对应关系：

- RBF：高斯函数
- Output：sign函数
- M：支持向量的数量

- $\boldsymbol{\mu}_m$：SVM的各个支持向量
- β_m：由对偶问题解出的$\alpha_m y_m$

因此RBF网络的问题就是，对给定的RBF和Output，求出$\boldsymbol{\mu}_m$和β_m

前面提到过，核函数是在变换后的空间$\mathcal{Z}$中的内积，因此能保证满足Mercer定理即可。核函数实际上描述了向量之间的相似性，而RBF描述的也是相似性，只不过这种相似性可以直接通过向量之间的距离来计算，因此可以定义不同的RBF，只要能满足对距离的定义就可以。一般情况下，随着距离的变大，RBF的值应该单调不增。也就是说，核函数和RBF是两种对距离的定义，而高斯函数就属于这两者的交集之中。除此以外，神经网络中每个神经元也可以看作是描述了一种距离的计算方式，为$\tanh(\gamma \mathbf{x}^{\mathsf{T}} \mathbf{x}' + 1)$。在DNA序列的应用中，可以使用编辑距离

总而言之，RBF说明相似性是一种很好的定义特征变换的方法，这种相似性通过计算点到中心的距离可以获得

径向基函数网络学习

前面讲过，RBF网络的目的是找到一些中心，以及将RBF值线性组合起来的系数。在最简单的例子中，可以把看到过的所有数据点都当做中心。这种RBF网络称为全RBF网络，满足$M = N$以及每个$\boldsymbol{\mu}_m = \mathbf{x}_m$。其物理意义可以解释为，训练集中的每条数据$\mathbf{x}_m$对其周围的数据都应该有一定的影响，而影响力由$\beta_m$决定。例如，假设每个点的影响力都一样，则对于新点，训练集中的每个点都会根据其到新点距离的远近做投票：总体来讲，一人一票，但是离得越近，系数越大。即$\beta_m = 1 \cdot y_m$。这种假设函数的形式为

$$g_{\mathrm{uniform}}(\mathbf{x}) = \mathrm{sign}\left(\sum_{m=1}^{N} y_m \exp(-\gamma \|\mathbf{x} - \mathbf{x}_m\|^2)\right)$$

即训练集中的每条数据都可以对新数据的所属类别发表意见，模型将所有人的意见收集起来，根据相似性分配票数，加权求和，最后输出。显然，对离新点$\mathbf{x}$最近的数据点，它与新点的高斯函数值最大，说话最有分量。由于高斯函数随距离增大衰减很快，因此实际上往往这个点的标签可以决定最后结果。从这个角度来说，可以避免求和项，只需要找到距离新点最近的那个点就可以了。此时，模型的聚合退化成了模型的选择，假设函数变为

$$g_{\mathrm{nbor}}(\mathbf{x}) = y_m \text{ such that } \mathbf{x} \text{ closest to } \mathbf{x}_m$$

这种方法称为**最近邻法**。在这种算法的基础上可以做一些扩展：不只找最近的邻居，而是找k个最近的邻居，将它们的结果做聚合。这种扩展的方法称为k-最近邻法（kNN）。kNN是一种比较“懒”的算法，但是却非常符合直觉。不过这种算法在测试时会比较费事

全RBF有什么好处呢？考虑将这种网络用在回归问题中，误差函数使用平方误差，而且不再指定β_m，而是让算法去优化这些系数。那么模型其实就是对经RBF变换过的数据求解线性回归。变换后的数据为

$$\mathbf{z}_n = [\mathrm{RBF}(\mathbf{x}_n, \mathbf{x}_1), \mathrm{RBF}(\mathbf{x}_n, \mathbf{x}_2), \ldots, \mathrm{RBF}(\mathbf{x}_n, \mathbf{x}_N)] \in \mathbb{R}^N$$

由前面的结论，可以轻易求出最优的系数$\boldsymbol{\beta} = (\mathrm{Z}^{\mathsf{T}}\mathrm{Z})^{-1}\mathrm{Z}^{\mathsf{T}}\mathbf{y}$，其中Z是对称方阵。进一步地，如果使用高斯函数做RBF，且所有$\mathbf{x}_n$都不同，则这个Z还是可逆的，系数$\boldsymbol{\beta}$可以继续化简为$\boldsymbol{\beta} = \mathrm{Z}^{-1}\mathbf{y}$

得到这个结果以后，如果要用得到的g送入原来的训练数据，有

$$g_{\mathrm{RBF}}(\mathbf{x}_1) = \boldsymbol{\beta}^{\mathsf{T}}\mathbf{z}_1 = \mathbf{y}^{\mathsf{T}}\mathrm{Z}^{-1}(\mathrm{Z}\text{的第一列}) = \mathbf{y}^{\mathsf{T}}[1 \quad 0 \quad \ldots \quad 0]^{\mathsf{T}} = y_1$$

以此类推，$g_{\mathrm{RBF}}(\mathbf{x}_n) = y_n$，$E_{\mathrm{in}}(g_{\mathrm{RBF}}) = 0$，意味着有过拟合的危险。如果使用岭回归，则$\boldsymbol{\beta} = (\mathrm{Z}^{\mathsf{T}}\mathrm{Z} + \lambda\mathrm{I})^{-1}\mathrm{Z}^{\mathsf{T}}\mathbf{y}$就不会产生前面这种“完美”的结果。考虑到Z是两个点的高斯函数值，也就是使用高斯核得到的核矩阵K，则这个模型跟前面讲过的核岭回归有异曲同工之妙。在核岭回归中，$\boldsymbol{\beta}_{\mathrm{kernel_ridge_regression}} = (\mathrm{K} + \lambda\mathrm{I})^{-1}\mathbf{y}$。只不过核岭回归是在无限多维空间中做正则，而正则化的全RBF是在N维空间中做正则

另外，回顾前面SVM的做法，它其实没有用到所有数据点做中心点，因此可以将中心点数量减少使得$M \ll N$，也可以起到正则化的作用。从物理意义上看，相当于不再让每个点都有话语权，而是选择一些代表出来去投票，代表所有人的意见。那么接下来的问题就是，如何找到好的代表呢？

K均值聚类算法

要寻找好的代表，首先可以看一个比较简单的情况。假设在训练集里，有$\mathbf{x}_1 \approx \mathbf{x}_2$，那么它俩的RBF值应该也接近，所以没有必要在RBF网络里保存两份近似的值$\mathrm{RBF}(\mathbf{x}, \mathbf{x}_1)$和$\mathrm{RBF}(\mathbf{x}, \mathbf{x}_2)$，而是让这两个数据点共用一个代表，也就是把这两个点**聚类**起来，得到$\boldsymbol{\mu} \approx \mathbf{x}_1 \approx \mathbf{x}_2$

在本系列课程最开始的时候就提到过聚类问题，这是一种非常典型的非监督学习问题。稍微形式化地说，它是要把数据集合$\{\mathbf{x}_n\}$划分为M个不相交的集合$S_1, S_2, \ldots, S_M$，而且为每个S_m选择一个代表$\boldsymbol{\mu}_m$，使得对任意$\mathbf{x}_n \in S_m \Leftrightarrow \boldsymbol{\mu}_m \approx \mathbf{x}_n$。因此，聚类问题通常使用的误差函数通常是看数据点到自己所属类别的代表的距离的平方总和，写作如下形式

$$E_{\text{in}}(S_1,\dots,S_M;\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_M)=\frac{1}{N}\sum_{n=1}^{N}\sum_{m=1}^{M}[\![\mathbf{x}_n\in S_m]\!]\|\mathbf{x}_n-\boldsymbol{\mu}_m\|^2$$

聚类算法就是要最小化这样的目标函数，即解决如下最优化问题

$$\min_{\{S_1,\dots,S_M\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_M\}}\sum_{n=1}^{N}\sum_{m=1}^{M}[\![\mathbf{x}_n\in S_m]\!]\|\mathbf{x}_n-\boldsymbol{\mu}_m\|^2$$

这并不是一个容易解决的问题，因为它是一种组合最优化问题，而且变量$S_1,\dots,S_M$都是类型变量，$\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_M$却是数值变量。不过因为这些变量可以分成两类$S$和$\boldsymbol{\mu}$，可以尝试使用各个击破的方法，先固定一组变量，对另一组变量做最优化，然后再反过来

首先，固定$\boldsymbol{\mu}_1,\dots,\boldsymbol{\mu}_M$，为每个$\mathbf{x}_n$挑选一个最好的$S_m$。由于每个$\mathbf{x}_n$都必须且只能对应一个最好的$S_m$，而$\boldsymbol{\mu}_m$固定所以可以知道$\mathbf{x}_n$到每个$\boldsymbol{\mu}_m$的距离，因此这个最优化动作很简单，只需要把离$\mathbf{x}_n$最近的那个$\boldsymbol{\mu}_m$分给它就好了

接下来，固定$S_1,\dots,S_M$，选出各个类（簇）的代表。此时原始最优化问题退化成一个与$\boldsymbol{\mu}_m$有关的无限值最优化问题，因此只需要求目标函数的梯度然后令其等于0即可。有

$$\nabla_{\boldsymbol{\mu}_m}E_{\text{in}}=-2\sum_{n=1}^{N}[\![\mathbf{x}_n\in S_m]\!](\mathbf{x}_n-\boldsymbol{\mu}_m)=-2\left(\left(\sum_{\mathbf{x}_n\in S_m}\mathbf{x}_n\right)-|S_m|\boldsymbol{\mu}_m\right)$$

令上式为0，则$\boldsymbol{\mu}_m$是属于S_m的$\mathbf{x}_n$的均值

上述过程其实就是非常有名的聚类算法——K均值聚类算法的核心过程（这里的K其实就是前面说的M）。其完整流程如下：

1. 初始化$\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\dots,\boldsymbol{\mu}_K$。通常做法是在$\mathbf{x}_n$里随机选$K$个点
2. 交替最优化E_{in}，即重复以下过程直至收敛（收敛的判定方法通常是$S_1,\dots,S_K$不再变化
   - 最优化$S_1,S_2,\dots,S_K$，即把$\mathbf{x}_n$分给离它最近的那个$\boldsymbol{\mu}_i$所属的类簇
   - 最优化$\boldsymbol{\mu}_1,\boldsymbol{\mu}_2,\dots,\boldsymbol{\mu}_K$，即对每个$S_k$，求属于该类簇所有点的均值

由于这个过程一直在使E_{in}变小，因此算法一定会收敛

K均值聚类算法可能是所有聚类算法里最有名的一个，其核心思路就是做交互最优化

可以把K均值聚类算法与RBF网络结合起来，算法如下

1. 使用K均值聚类算法（这里$K=M$）得到M个中心点$\{\boldsymbol{\mu}_m\}$
2. 使用RBF（例如高斯函数）围绕$\boldsymbol{\mu}_m$构建变换$\boldsymbol{\Phi}(\mathbf{x})$
   $$\boldsymbol{\Phi}(\mathbf{x})=[\text{RBF}(\mathbf{x},\boldsymbol{\mu}_1),\text{RBF}(\mathbf{x},\boldsymbol{\mu}_2),\dots,\text{RBF}(\mathbf{x},\boldsymbol{\mu}_M)]$$
3. 在变换后的数据$\{(\boldsymbol{\Phi}(\mathbf{x}_n),y_n)\}$上运行线性模型，得到$\boldsymbol{\beta}$
4. 返回$g_{\text{RBFNET}}(\mathbf{x})=\text{LinearHypothesis}(\boldsymbol{\beta},\boldsymbol{\Phi}(\mathbf{x}))$

就像自动编码器一样，这里RBF网络也是使用无监督学习算法来萃取特征

K均值聚类和径向基函数网络实战

实验说明K均值聚类算法的效果对K的设置和初始点的设置比较敏感。以及全RBF网络和kNN由于需要遍历所有数据集，因此预测时效率不高，所以不是很常用

NTUML 31. 矩阵分解

线性网络假设函数

在本系列课程的第一讲里，曾讲到过一种称为“推荐系统”的问题，也就是根据用户对电影的评分历史，估计ta对一部新电影可能的评分。也就是说，对于第m部电影，其数据集$\mathcal{D}_m$可以表示为

$$\{(\tilde{\mathbf{x}}_n = (n), y_n = r_{nm}) : \text{ user } n \text{ rated movie } m\}$$

这种数据与前面介绍的数据最大的不同是，$\tilde{\mathbf{x}}_n$是抽象特征，只是一个用户的编号ID。我们不知道ta背后是男是女，是长是幼，对电影是什么口味，只知道ta打过了哪些分数。那么如何从这些数据里学到每个用户的喜好呢？

首先，要考虑如何表示这种特征。需要注意的是，用户ID只是一个离散的数值，没有什么实际意义。此外，对用户ID的比较、四则运算等也是没有意义的（一般来说，用户ID越小，说明用户注册时间越早，所以可能会透露出一些额外的信息。不过这里认为评分和注册时间没有什么关系）。类似的特征有很多，例如人的血型，编程语言种类等等。这种没有数值关联性的特征通常称为**类别特征**。目前为止，介绍过的大部分模型基本都是接受数值特征，例外情况只有决策树及其延伸（例如随机森林）。因此，要把这些类别特征利用到更广泛的模型中，就需要将它们转成数值特征，这也是特征变换的一种。不过这种特征变换通常是预先定义好的，这种预先定义的动作就称为**编码**。一种比较简单的编码方式称为二值向量编码方式（binary vector encoding，更常见的称呼方法是独热编码 one hot encoding）：假设对于该变量有t种取值，那么就建立一个长度为t的数组，每一位都分给一种可能的取值情况。当该特征属于某种取值时，其对应的那一位置为1，其余位置为0。例如，人的血型有A、B、O和AB四种取值情况，那么这四个值可以分别表示为

$$\begin{aligned}\texttt{A} &= [1,0,0,0]^\mathsf{T}\\ \texttt{B} &= [0,1,0,0]^\mathsf{T}\\ \texttt{O} &= [0,0,1,0]^\mathsf{T}\\ \texttt{AB} &= [0,0,0,1]^\mathsf{T}\end{aligned}$$

因此，对第m个电影，编码后的数据$\mathcal{D}_m$可以表示为

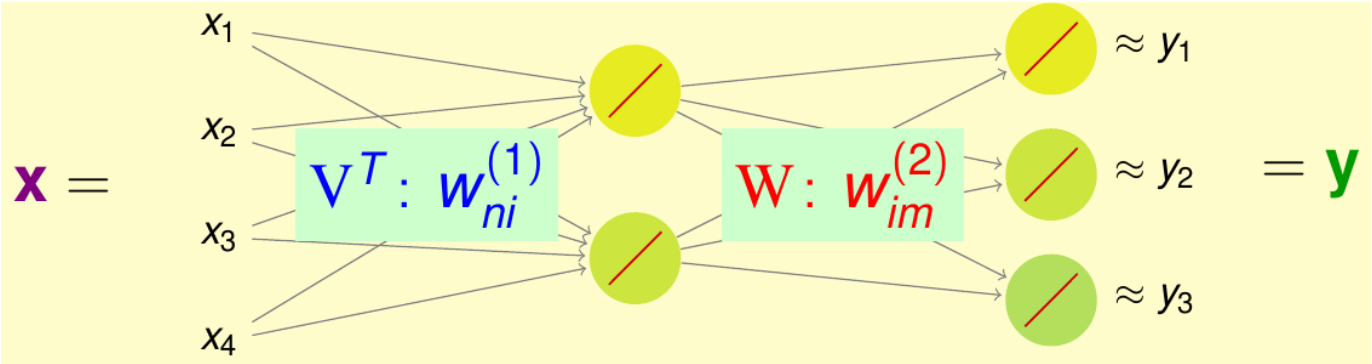
$$\{(\mathbf{x}_n = \text{OneHotEncoded}(n), y_n = r_{nm}) : \text{ user } n \text{ rated movie } m\}$$

进一步地，可以把用户对所有电影的评分收集起来，得到一个联合数据$\mathcal{D}$

$$\{(\mathbf{x}_n = \text{OneHotEncoded}(n), \mathbf{y}_n = [\, r_{n1} \quad ? \quad ? \quad r_{n4} \quad r_{n5} \quad \dots \quad r_{nM} \,]^\mathsf{T})\}$$

由于用户基本不可能对所有电影都作出过评分，因此可以将用户没有评过分的项用“？”标记起来

这个特征仍然抽象，还是需要考虑怎么做进一步的特征提取和变换。由于前面讲到的神经网络有这方面的功能，因此可以用神经网络来解决这个问题。假设网络使用如下结构构建：输入对应使用者，是一个N维独热编码过的向量，中间使用$\tilde{d}$个神经元做隐藏层，最后输出是一个M维的向量对应电影评分，那么这个网络很像前面的自动编码器，看上去应该是有提取特征的功能。注意为了简单起见，网络里不带常数项$\mathbf{x}_0^{(\ell)}$。此外，这个应用里隐藏层不需要做非线性变换$\tanh$，因为输入向量太稀疏了。综合起来，网络的结构可以用下图表示。由于隐藏层都使用的是线性变换，因此该网络也可以称为线性网络



求解推荐系统问题的线性网络

按照上图所示，将两层权重分别矩阵化

$$\begin{aligned}\mathrm{V}^\mathsf{T} &= [w_{ni}^{(1)}] \in \mathbb{R}^{N\times\tilde{d}}\\ \mathrm{W} &= [w_{im}^{(2)}] \in \mathbb{R}^{\tilde{d}\times M}\end{aligned}$$

则线性网络的假设函数可以写为

$$h(\mathbf{x}) = \mathrm{W}^\mathsf{T}\mathrm{V}\mathbf{x}$$

其中每个用户n的输出为$h(\mathbf{x}_n) = \mathrm{W}^\mathsf{T}\mathbf{v}_n$，这里利用了稀疏向量的性质，$\mathbf{v}_n$是V的第$n$列

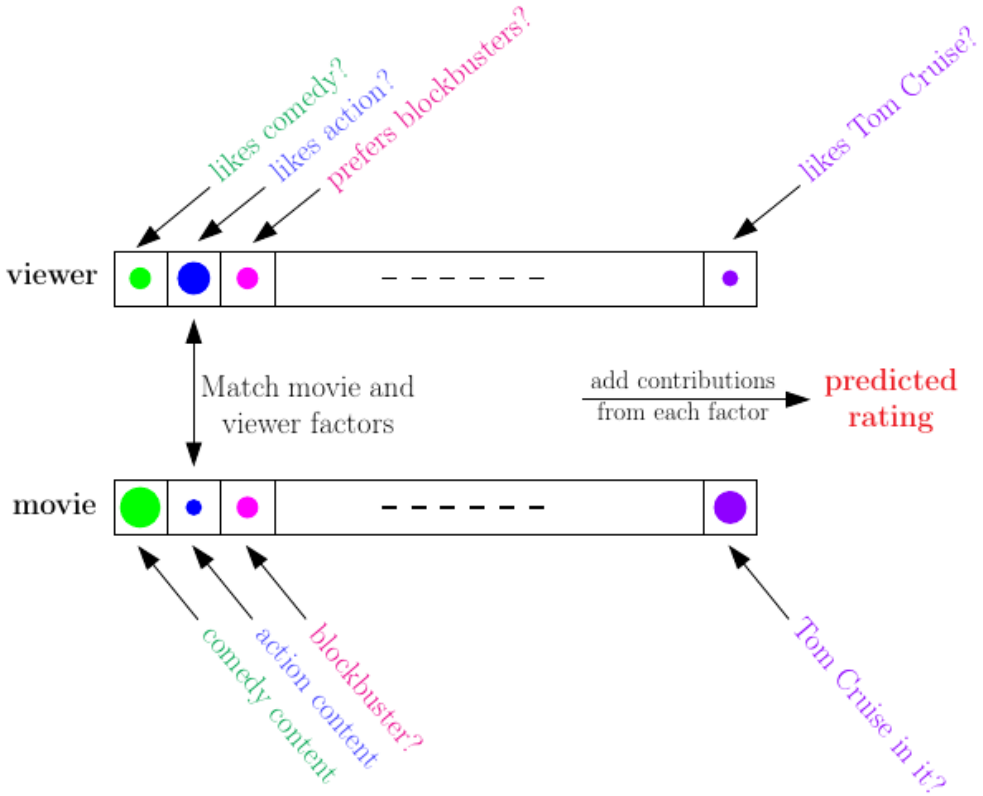
这样，用于推荐系统的线性网络，就是要最优化V和W

基本矩阵分解

前面给出的假设函数中，$\mathrm{V}\mathbf{x}$这一项可以看作是对$\mathbf{x}$做的特征变换$\mathbf{\Phi}(\mathbf{x})$。这样，如果单独看第$m$个电影，其对应的实际上就是一个线性模型$h_m(\mathbf{x}) = \mathbf{w}_m^{\mathsf{T}}\mathbf{\Phi}(\mathbf{x})$，训练的目标就是希望这个模型对每个$\mathcal{D}_m$都能达到$\mathbf{w}_m^{\mathsf{T}}\mathbf{v}_n \approx y_n = r_{nm}$的效果。因此，可以使用平方误差来做损失函数，即

$$E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \frac{1}{\sum_{m=1}^{M}|\mathcal{D}_m|} \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^{\mathsf{T}}\mathbf{v}_n)^2$$

由于内积的对称性，有$r_{nm} \approx \mathbf{w}_m^{\mathsf{T}}\mathbf{v}_n = \mathbf{v}_n^{\mathsf{T}}\mathbf{w}_m$。将这个式子矩阵化，有$\mathrm{R} \approx \mathrm{V}^{\mathsf{T}}\mathrm{W}$，其中$\mathrm{R}$是评分矩阵，$\mathrm{V}$是所有用户特征组成的矩阵，$\mathrm{W}$是所有电影特征（权重）组成的矩阵。如果能通过已知的一部分R（不需要知道所有的元素）将其分解为V和W的乘积，那么就能得到使用者特征，和每部电影应该怎么做线性组合。也就是说，在这个模型里做的模型是由已知评分反推那些特征影响用户评分，以及每部电影有哪些元素。例如，某个维度可能对应用户是否喜欢喜剧片，以及电影是否是喜剧片，如下图所示



矩阵分解的物理意义

矩阵分解经常用来从抽象特征中提取具体特征

矩阵分解的最优化问题可以进一步推演

$$\begin{aligned}\min_{\mathrm{W},\mathrm{V}} E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) &\propto \sum_{\text{user } n \text{ rated movie } m} (r_{nm} - \mathbf{w}_m^{\mathsf{T}}\mathbf{v}_n)^2 \\ &= \sum_{m=1}^{M} \left(\sum_{(\mathbf{x}_n, r_{nm}) \in \mathcal{D}_m} (r_{nm} - \mathbf{w}_m^{\mathsf{T}}\mathbf{v}_n)^2 \right)\end{aligned}$$

这里可以看到又出现了两组变量，难以同时优化，但是可以交替优化：先固定$\mathbf{v}_n$，最小化$\mathbf{w}_m$，此时就是求解一组线性回归模型，不过不带常数项；然后，固定$\mathbf{w}_m$，最小化$\mathbf{v}_n$，也是求解一组线性回归模型，只是此时是求解关于用户的线性回归模型，而且也不带常数项。这种算法称为交替最小二乘法。其完整表述如下

> 1. 随机初始化$\tilde{d}$维向量$\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$
> 2. 交替最小化E_{in}：重复如下过程直到收敛
>   - 最优化$\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M$，对第$m$个电影，对数据集$\{(\mathbf{v}_n, r_{nm})\}$求解一个线性回归模型，更新$\mathbf{w}_m$
>   - 最优化$\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$，对第$n$个用户，对数据集$\{(\mathbf{w}_m, r_{nm})\}$求解一个线性回归模型，更新$\mathbf{v}_n$

其实，前面讲的自动编码器也隐含了矩阵分解的思想。类比于矩阵分解的假设函数$\mathrm{R} \approx \mathrm{V}^{\mathsf{T}}\mathrm{W}$，自动编码器的假设函数也可以写为$\mathrm{X} \approx \mathrm{W}(\mathrm{W}^{\mathsf{T}}\mathrm{X})$。两者都是一种线性神经网络，在隐含层使用$\tilde{d}$个节点。只不过自动编码器的误差函数是在所有$x_{ni}$上做测量，而矩阵分解只在已知$r_{nm}$上做测量，而且前者能得到全局最优解，后者只能得到局部最优解。总而言之，自动编码器可以看作是矩阵分解的一种特殊形式

随机梯度下降

上述问题也可以用随机梯度下降（SGD）来求解：从数据集中选取一条数据，在这条数据上算梯度，做梯度下降。这种算法的效率非常高，而且易于实现，易于扩展到其它误差函数上

首先，写出对每一条数据的误差函数

$$\text{err}(\text{user } n, \text{movie } m, \text{rating } r_{nm}) = (r_{nm} - \mathbf{w}_m^\mathsf{T}\mathbf{v}_n)^2$$

其梯度非常好计算

$$\nabla_{\mathbf{v}_n}\text{err} = -2(r_{nm} - \mathbf{w}_m^\mathsf{T}\mathbf{v}_n)\mathbf{w}_m$$
$$\nabla_{\mathbf{w}_m}\text{err} = -2(r_{nm} - \mathbf{w}_m^\mathsf{T}\mathbf{v}_n)\mathbf{v}_n$$

有趣的是，这两项梯度都正比于残差与另一组变量向量的乘积

套用前面讲过的SGD算法，使用SGD求解矩阵分解的过程可以写为

> 随机初始化$\tilde{d}$维向量$\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$
>
> 对$t = 0, 1, \ldots, T$
>
> - 在所有已知的r_{nm}里随机选取一个(n, m)
> - 计算残差$\tilde{r}_{nm} = (r_{nm} - \mathbf{w}_m^\mathsf{T}\mathbf{v}_n)$
> - SGD更新
>
> $$\mathbf{v}_n^{\text{new}} \leftarrow \mathbf{v}_n^{\text{old}} + \eta \cdot \tilde{r}_{nm}\mathbf{w}_m^{\text{old}}$$
> $$\mathbf{w}_m^{\text{new}} \leftarrow \mathbf{w}_m^{\text{old}} + \eta \cdot \tilde{r}_{nm}\mathbf{v}_n^{\text{old}}$$

如果要求解的是大规模矩阵分解问题，SGD非常有效

台大团队在参加2011年KDD Cup比赛时用到的就是SGD矩阵分解，不过他们根据问题的场景做了一点有针对性的修改。这次比赛是根据用户之前一段时间的打分去预测后面一段时间的打分，也就是说训练数据的时间戳要比测试数据早一些。在运行SGD时，实际上最后T'次迭代里算法会倾向于把它在这段时间里看到的数据效果做好，因此台大队会故意让时间戳较晚的数据在比较靠后的迭代中训练。这意味着，如果了解了模型使用最优化的方式和里面的技巧，那么就能够在实际应用时修改算法以使其更符合问题的需要

特征提取模型小结

从神经网络开始所讲的模型基本都可以称为是**特征提取模型**。所谓特征提取模型，指的是在送入最终模型之前，将特征变换的过程也纳入学习过程，学习如何表示数据才能有更好的结果。例如在神经网络里，对数据的变换是隐藏层的权重$w_{ij}^{(\ell)}$，在RBF网络里是$\boldsymbol{\mu}_m$（其中KNN是RBF网络的特例），在矩阵分解里则是$\mathbf{v}_n$（也可以是$\mathbf{w}_m$）。从另一个角度看，AdaBoost或者GBDT也有一点特征提取模型的味道，这里对数据的变化是所有基分类器

不同的模型也有一些提取特征的技巧：Boosting算法里是函数梯度下降，神经网络和深度学习里是反向传播+SGD以及自动编码器，RBF网络里是K均值聚类，矩阵分解里是交替最小二乘，而KNN是“偷懒学习”（在测试时才遍历训练集作比较）

特征提取模型的优点是容易使用，减轻了人类提取特征的负担，而且如果使用了足够多的隐藏变量，模型的能力通常比较强；缺点是问题通常不是凸优化问题，比较难以优化，难以达到全局最优解，而且容易过拟合，需要合适的正则化方法或者验证方法。因此，这些模型用起来一定要小心

NTUML 32. 结束曲

特征变换技术

关于特征变换，最开始讲的是**核技巧**。核技巧将可能非常多的特征变换都埋藏在了一个核操作中，包含了多项式核、高斯核和决策树桩核等。核需要满足Mercer条件，所有满足该条件的函数都可以做核函数，值得一提的是，两个核的和/积都还是核。核的应用非常广

泛，无论是常见的方法诸如SVM、SVR还是概率SVM，或者是少见的模型例如核岭回归和核Logistic回归，其核心都是核方法的应用。它们都延伸了原有的线性模型，通过核得到复杂的特征

运用特征变换的第二种方法是**模型聚合**，这种方法的基本思想是将一些比较基础的预测模型g都看作是一种特征变换，将预测得到的结果聚合起来，使用一些聚合技术，得到一个更大的模型G。常见的g包括决策树桩、决策树、（高斯）RBF等等。如果已经有了若干个g，就可以做模型混合，比如公平投票法（均匀混合）、带权重的投票法（非均匀混合）和有条件的混合等等。如果没有g，可以使用诸如bagging的方法得到不一样的g让它们投票，例如随机森林；也可以一步步找到非常不一样的g，这样就是使用AdaBoost或者梯度Boost；而决策树和KNN有点类似于有条件混合的策略。从另一个角度，概率SVM也有一点模型混合的味道，只不过是先学到SVM，然后将其与一个常数模型做线性聚合

第三种方法是**找出潜藏特征**，把特征看作是一些隐藏的变量，然后使用传统的权重法将其组合起来。优化时，寻找特征和最后组合特征的过程是一起优化的，而特征的寻找有时还要借助一些无监督学习的方法。这部分常见的方法包括神经网络（神经元的权重）、RBF网络（找到中心点）和矩阵分解（例如找到用户/电影的隐藏特征）。其中RBF网络会使用k均值聚类方法，神经网络会使用自动编码器或PCA等方法

这些模型里还有一个共同的特点，但是容易被人所忽略，即它们都做了一些“特征降维”的方法，例如将特征从高维度到低维度的投影。例如决策树桩是把特征投影到$\mathbb{R}$上，PCA或者自动编码器是在最大程度保留信息的基础上做特征压缩，矩阵分解是把特征从抽象特征（用户ID）变换到具体特征上。而随机森林不仅做了一些随机的低维投影，还通过一些特征选择的方法选出最有用的若干特征

误差优化技术

接着，从最优化的角度来对前面学过的知识做一总结。最基本的最优化方法是**梯度下降**，即如果能够（大致）知道某个梯度的方向，就能把这个方向作为误差函数的一次逼近：往负梯度方向走一小步就可以。梯度下降及其变种随机梯度下降在很多模型中都有应用，例如Logistic回归、神经网络等。梯度下降的延伸方法，函数梯度下降+最陡下降（steepest descent）则是在AdaBoost和梯度Boost中使用

一些更困难的最优化问题需要经过一些数学推导，变型成已知的最优化问题。例如在对偶SVM中，是通过对偶方法推出对偶问题，并说明其等价于凸二次规划问题。而在核Logistic回归和核岭回归中，使用的是表示定理。PCA则是使用了特征值与特征向量方面的知识

另外一些比较困难的方法则是将其分解为一些子问题，分别求解。例如模型混合法，通常是先求解g，再求解G。自动编码器则是一层一层地推进。K均值聚类和矩阵分解使用的是交替求解法，而决策树则是使用了“分而治之”的思想

过拟合的避免

最后，再来看一个很重要的问题，就是如何消除或减小过拟合的影响。当模型很复杂时，这个步骤就更加重要。避免过拟合的一种方法是使用正则化，这种方法在很多算法中都有使用。例如SVM和AdaBoost是使用大边界的分类器（AdaBoost是隐含了这方面的思想），SVR、核模型等使用的是L2正则项，模型混合法加入了模型投票的机制，自动编码器使用的是降噪机制（类似于人类接种疫苗），决策树的剪枝，神经网络的dropout或提早停止（其中后者可以应用到任何基于梯度下降的算法），以及对模型做一些预设的限制，例如限制RBF网络的中心点数量

避免过拟合的另一种方法是验证法。例如随机森林使用OOB来做类似的检查，SVM和SVR可以检查支持向量的数量

机器学习实战

首先来看一下台大队在过去几次KDD Cup中使用的方法

- 2010年KDD Cup，使用了模型混合的方法。混合的基模型一部分是Logistic回归，其输入为大量经过编码的原始特征；另一部分是随机森林，其输入为人类运用领域知识设计出来的特征
- 2011年KDD Cup Track 1，基模型为矩阵分解模型（包括概率PCA）、受限玻尔兹曼机（自动编码器的一种扩展）、KNN、概率LSA、线性回归、神经网络、GBDT。对这些基模型，先使用非线性模型混合（例如神经网络和决策树），最后使用一次线性混合
- 2012年KDD Cup Track 2，基模型为线性回归及其变种（例如SVR）、Logistic回归及其变种、矩阵分解及其变种。对这些基模型，先使用非线性模型混合（例如神经网络和GBDT），最后使用一次线性混合
- 2013年KDD Cup Track 1，基模型为随机森林（使用了很多树）、GBDT变种，使用线性混合。对这次比赛，还投入了大量的精力来设计特征

总之，台大队经常获胜的秘诀就是精心设计领域知识+使用模型混合防止过拟合

ICDM在2006年给出过十大数据挖掘算法

- C4.5（决策树的另一种方法）
- K均值聚类
- SVM

- Apriori（一种典型数据挖掘的算法，找出频繁出现项）
- EM算法（可以看作是交替优化法的一种特例）
- PageRank（Google最原始分析网站重要性的方法，类似与矩阵分解）
- AdaBoost
- KNN
- 朴素贝叶斯（一种简单的线性模型，权重通过数据的统计特性决定）
- C&RT

（当然现在最火的就是深度学习了:-P）

到此为止，台大林轩田老师的机器学习课笔记全部连载完毕，感谢您的阅读。当然，更感谢林轩田老师的精彩授课。在我看来，这是一门不可多得的中文机器学习课程，无论是深度和广度都是恰到好处。另外，这门课的作业设计也非常出色，如果能把这些作业做完，无疑能进一步地加深对这些知识点的理解，不管是从理论的角度，还是实现的角度，都是如此。本系列课程的永久网址可见 https://www.csie.ntu.edu.tw/~htlin/course/ml15fall/