

1. 矩阵的基本运算

1.1 基本运算

1.2 特殊的矩阵

2. 行列式、逆、秩、迹**3. 特征值与特征向量****4. 矩阵的分解**

4.1 与求解线性方程组有关的分解

4.2 基于特征值及相关概念的分解

1. 矩阵的基本运算

1.1 基本运算

- 加法 / 减法

$$\begin{aligned} A + B &= B + A \\ (A + B) + C &= A + (B + C) \end{aligned}$$

- 数乘

$$\begin{aligned} \lambda(\mu A) &= \mu(\lambda A) \\ \lambda(\mu A) &= (\lambda\mu)A \\ (\lambda + \mu)A &= \lambda A + \mu A \\ \lambda(A + B) &= \lambda A + \lambda B \end{aligned}$$

矩阵的加减法和数乘合成为矩阵的线性运算。

- 转置：把矩阵 A 的行和列互相交换所产生的矩阵称为 A 的转置矩阵，记为： A^T 。

$$\begin{aligned} (A^T)^T &= A \\ (\lambda A)^T &= \lambda A^T \\ (AB)^T &= B^T A^T \end{aligned}$$

- 共轭：把矩阵 A 的各个元素换为其对应的共轭复数生成的矩阵称为 A 的共轭矩阵，记为： \bar{A} 。
- 共轭转置：对矩阵先转置再共轭或者先共轭再转置得到的矩阵，记为 A^* or A^H 。

1.2 特殊的矩阵

	名称	实数域	名称	复数
$m \times n$	正规矩阵	$A^T A = AA^T$	正规矩阵	$A^H A = AA^H$
$n \times n$	对称矩阵	$A^T = A$	厄米特矩阵	$A^H = A$
$n \times n$	斜对称矩阵	$A^T = -A$	斜厄米特矩阵	$A^H = -A$
$n \times n$	正交矩阵	$Q^T Q = QQ^T = I$ $Q^T = Q^{-1}$	酉矩阵	$U^H U = UU^H = I$ $U^H = U^{-1}$

2. 行列式、逆、秩、迹

- **行列式:** 一个 $n \times n$ 的矩阵 A 的行列式记为 $\det(A)$ or $|A|$, 表示对线性变换的度量。一个矩阵的行列式等于其任意行（或列）的元素与对应的代数余子式乘积之和。

$$\det(A) = |A| = \sum_{i=1}^n a_{ik} A_{ik} = \sum_{j=1}^n a_{kj} A_{kj}$$

$$A_{ij} = (-1)^{i+j} M_{ij}$$

其中：代数余子式为 A_{ij} ; 余子式为 M_{ij} 。

- **逆:** 一个 $n \times n$ 的矩阵 A , 若在相同数域上存在另一个 n 阶矩阵 B , 使得: $AB = BA = I$, 则我们称 B 是 A 的逆矩阵, 而 A 则被称为可逆矩阵。 $(A^*:$ 伴随矩阵)

$$A^{-1} = \frac{A^*}{|A|}, \quad A^* = (A_{ij})^T$$

初等变换 : $[A|I] \rightarrow$ 行 (或列) 初等变换 $\rightarrow [I|A^{-1}]$

- **秩:** 一个矩阵 A 的列秩是 A 的线性无关的纵列的极大数目; 类似地, 行秩是 A 的线性无关的横行的极大数目。表示的是: 变换后空间的维数, 即列空间的维数, 记为: $\text{rank}(A)$ or $r(A)$ 。

以下三种表示等效 (针对方阵) : 行列式不为零, 矩阵可逆, 矩阵满秩。

- **迹:** 一个 $n \times n$ 的矩阵 A 的主对角线上各个元素的总和称为矩阵 A 的迹。

$$\text{tr}(A) = a_{11} + a_{22} + \dots + a_{nn} = \sum_{i=1}^n a_{ii}$$

3. 特征值与特征向量

- **特征值与特征向量:** 向量 \vec{v} 在变换 A 下没有离开自己张成的空间。用数学表达式表示为:

$$A\vec{v} = \lambda\vec{v}$$

其中: λ 为特征值, \vec{v} 为特征向量。矩阵的特征值有如下性质:

$$\det(A) = \prod \lambda_i$$

$$r(A) = \text{非零特征值的个数}$$

$$\text{tr}(A) = \sum \lambda_i$$

4. 矩阵的分解

我们一般考虑矩阵为实矩阵。

4.1 与求解线性方程组有关的分解

- **三角分解：**设矩阵 $A \in F^{n \times n}$, 若 $L, U \in F^{n \times n}$ 分别是下三角和上三角矩阵, 满足 $A = LU$, 则称矩阵 A 可作三角分解; 若 $L, V \in F^{n \times n}$ 分别是对角线元素为 1 的下三角和上三角矩阵, D 为对角矩阵, 满足 $A = LDV$, 则称矩阵 A 可作 LDV 分解。LDV 分解是 LU 分解的更进一步, 即把 L, U 对角线上的元素提取出来组成 D 。三角分解可以通过高斯消元法求解, 过程如下:

$$(A | I) \rightarrow \text{行初等变换} \rightarrow (U | P)$$
$$L = P^{-1}$$

故可解得 L, U , 即 $A = LU$, 然后再把对角线提取出来, 即可得到 $A = LDV$ 。

如果方阵 A 是非奇异的, 即行列式不为 0, 三角分解总是存在的。

- **满秩分解：**设矩阵 $A \in F^{m \times n}$, $\text{rank}(A) = r$, 若存在秩为 r 的矩阵 $B \in F^{m \times r}$ (列满秩), $C \in F^{r \times n}$ (行满秩), 满足 $A = BC$, 则称矩阵 A 可做满秩分解。计算方法如下:

方法一:

$$(A | I_m) \rightarrow \text{行初等变换} \rightarrow \begin{pmatrix} C & P \\ 0 & \end{pmatrix}$$

B 为 P^{-1} 的前 r 列, C 为 A 化为阶梯形的非零行, $A = BC$ 。

方法二:

首先定义 **Hermite 标准型**: 就是阶梯型中的每一个行第一个非零元素为 1, 而且该元素所在列中其他元素为 0 的特殊的一种。计算步骤如下:

- 用行初等变换把 A 化为 Hermite 标准型;
- 依 Hermite 标准型中的列向量 e_i 所在列的位置即 j_i 列, 相应取出 A 的第 j_i 列 a_{j_i} , 得到 A 的极大线性无关组 $\{a_{j_1}, a_{j_2}, \dots, a_{j_r}\}$; 则矩阵 $B = \{a_{j_1}, a_{j_2}, \dots, a_{j_r}\}$ 。
- Hermite 标准型中非零行构成矩阵 C , 即得到 A 的满秩分解 $A = BC$ 。

满秩分解一定存在, 但不唯一。

- **QR 分解：**设矩阵 $A \in F^{m \times n}$, 若存在正交矩阵 $Q \in F^{m \times m}$, 上三角矩阵 $R \in F^{m \times n}$, 满足 $A = QR$, 则称矩阵 A 可做 QR 分解。

如果矩阵 A 列满秩且要求矩阵 R 的对角线元素为正, 则 QR 分解存在且唯一。

4.2 基于特征值及相关概念的分解

- **特征分解：**特征分解, 又称谱分解, 是将矩阵分解为由其特征值和特征向量表示的矩阵之积的方法。只有可对角化矩阵才能进行特征分解。(可对角化矩阵: 矩阵 A 有 n 个线性无关的特征向量)

设 $A \in F^{n \times n}$, 且有 n 个线性无关的特征向量 q_i 。这样, 矩阵 A 可以被分解为:

$$A = Q\Lambda Q^{-1}$$

其中 $Q \in F^{n \times n}$, 且且第 i 列为 A 的特征向量 q_i ; Λ 是对角矩阵, 其对角线上的元素为对应的特征值。s

- **奇异值分解**: 矩阵值分解, 又称 SVD 分解。设矩阵 $A \in C^{m \times n}$, $\text{rank}(A) = r$, $\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_r > 0$ 是矩阵 A 的奇异值, 则存在酉矩阵 $U \in C^{m \times m}$, $V \in C^{n \times n}$, 分块矩阵 $\Sigma = \begin{pmatrix} \Delta & 0 \\ 0 & 0 \end{pmatrix} \in C^{m \times n}$, 满足

$$A = U\Sigma V = U \begin{pmatrix} \Delta & 0 \\ 0 & 0 \end{pmatrix} V^H$$

其中:

$$\Delta = \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{pmatrix}$$

计算过程如下:

- 求矩阵 $A^H A$ 的特征值, 大于 0 的特征值按从大到小的顺序组成对角矩阵 Δ , 得到矩阵 Σ ;
- 求矩阵 $A^H A$ 的特征向量, 然后进行标准正交化, 得到矩阵 V ;
- 令 $u_i = \frac{1}{\sigma_i} Av_i, i = 1, 2, \dots, r$, 得到 C^m 中的 $\{u_1, u_2, \dots, u_r\}$, 把它扩充为 C^m 中的标准正交基, 就得到矩阵 U 。计算完成。

- 1. 初识机器学习
- 2. 什么是机器学习
 - 2.1 监督学习
 - 2.2 无监督学习

1. 初识机器学习

- **Artificial Intelligence:** Human Intelligence Exhibited by Machines.
 - 人工智能（AI）：机器展示的人类智能。
- **Machine Learning:** An Approach to Achieve Artificial Intelligence.
 - 机器学习（ML）：一种实现人工智能的方法。
- **Deep Learning:** A Technique for Implementing Machine Learning.
 - 深度学习（DL）：一种实现机器学习的技术。

2. 什么是机器学习

什么是机器学习，这个问题目前还没有明确、统一的回答。本课程中列举了两种定义：

- 在没有明确设置的情况下，使计算机具有学习能力的研究领域。—— **Arthur Samuel**
Machine Learning: The field of study that gives computers the ability to learn without being explicitly programmed.
- 计算机程序从经验 **E** 中学习，解决某一任务 **T**，进行某一性能度量 **P**，通过 **P** 测定在任务 **T** 上的表现因经验 **E** 而提高。—— **Tom Mitchell**
Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

目前，机器学习主要的两类方式：

- **Supervised learning:** 监督学习
- **Unsupervised learning:** 无监督学习
- **Others:**
 - **Semi-supervised learning:** 半监督学习
 - **Reinforcement learning:** 强化学习

2.1 监督学习

监督学习：我们已经得到了一个数据集、并且已经知道正确的输出是什么样子（数据含有标签），而且预设输入和输出之间一定存在某种联系。

- **regression problem**: 回归问题，我们尝试预测出**连续的**输出，类似房价问题。
- **classification problem**: 分类问题，我们尝试预测出**离散的**输出，类似肿瘤问题。

2.2 无监督学习

无监督学习: 使用的数据集**没有任何标签**，在我们不知道输入 / 输出是什么的前提下，从数据中提取出结构 (structure) 。

- **clustering problem**: 聚类问题，自动将数据聚类。

1. 监督学习

2. 线性回归 (Linear Regression)

- 2.1 代价函数 (cost function)
- 2.2 梯度下降法 (gradient descent algorithm)
 - 1. 批量梯度下降法 (batch gradient descent)
 - 2. 随机梯度下降法 (stochastic gradient descent)
- 2.3 正规方程 (The normal equations)
 - 1. 矩阵倒数
 - 2. 矩阵的迹
 - 3. 最小二乘法
- 2.3 概率解释 (Probabilistic interpretation)
- 2.4 局部加权线性回归 (Locally weighted linear regression)

3. 分类和逻辑回归 (Classification and logistic regression)

- 3.1 逻辑回归 (logistic regression)
- 3.2 题外话: 感知器学习算法 (The perceptron learning algorithm)
- 3.3 取 $l(\theta)$ 最大值的另外一个算法—牛顿法

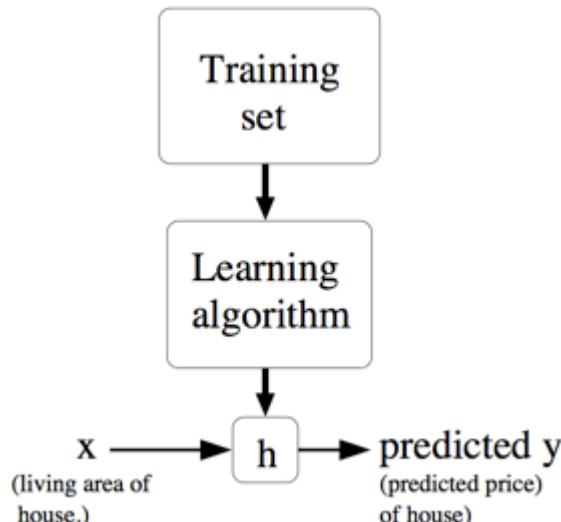
4. 广义线性模型 (Generalized Linear Models)

- 4.1 指数族 (The exponential family)
- 4.2 构建广义线性模型 (Constructing GLMs)
 - 1. 线性回归 (Linear Regression)
 - 2. 逻辑回归 (Logistic Regression)
 - 3. Softmax 回归

参数拟合

1. 监督学习

如果我们用规范的方式来描述监督学习问题，我们的目标是：给定一个训练集，来让机器学习一个函数 $h : X \rightarrow Y$ ，让 $h(x)$ 能是一个与真实 y 值比较接近的评估值。这个函数 h 就被叫做假设 (**hypothesis**)。用图标表示的话，这个过程大概就是下面这样：



如果我们要预测的目标变量 y 是连续的，这种学习问题就被称为回归问题；如果 y 是离散的值，这样的问题就叫做分类问题。

2. 线性回归 (Linear Regression)

为了让我们的房屋问题更加具有代表性，我们对数据集增加多个输入特征，如下：

居住面积 (平方英尺)	卧室数目 (个)	价格 (千美元)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
...

现在，输入特征 x 是一个在 R^2 范围内取值的一个二维向量了，例如 $x_1^{(i)}$ 就是训练集中第 i 个房屋的面积，而 $x_2^{(i)}$ 就是训练集中第 i 个房屋的卧室数目。要进行这个监督学习，我们必须确定好合适的假设 h 进行表示。在这里，我们选择简单的线性函数表示：

$$h(x) = h_{\theta}(x) = \theta_0 + \theta_1 \times x_1 + \theta_2 \times x_2$$

这里的 θ_i 是参数，是从 x 到 y 的线性函数映射的空间参数。在不至于引起混淆的情况下，我们可以把 $h_{\theta}(x)$ 里面的 θ 省略掉，就简写成 $h(x)$ 。另外，为了简化公式，假设 $x_0 = 1$ (截距项 **intercept term**)， n 为输入特性的维数，简化成向量表示如下：

$$h(x) = \sum_{i=0}^n \theta_i x_i = \theta^T x$$

2.1 代价函数 (cost function)

给定了训练集和假设 h ，我们如何选择合适的学习参数 θ 呢？一个看上去比较合理的方法就是让 $h(x)$ 尽量逼近 y ，用公式的表示的话，就要定义一个函数，来衡量不同的 θ 值时， $h(x^{(i)})$ 与 $y^{(i)}$ 的距离。用这样的方式定义的函数就叫做**成本函数 (cost function)**。在当前问题中，我们采用**最小均方算法 (LMS algorithm)** 作为代价函数，如下：

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

其中， m 表示训练集中样本的个数。

2.2 梯度下降法 (gradient descent algorithm)

我们希望选择一个能让 $J(\theta)$ 最小的 θ 值。怎么做呢，我们使用一个搜索的算法，首先对 θ 进行初始化，然后通过某种方法对 θ 值不断进行调整，使得 $J(\theta)$ 逐渐变小直至收敛。这里我们使用梯度下降法 (gradient descent algorithm)，初始化 θ 值，使用如下公式进行重复更新直至收敛，其中 α 表示学习速率：

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

注：本文中 $:=$ 表示的是计算机程序中的一种赋值操作，是把等号右边的计算结果赋值给左边的变量， $a := b$ 就表示用 B 的值覆盖 A 原有的值。要注意区分，如果写的是 $a = b$ 则表示的是判断二者相等的关系。

要实现这个算法，咱们需要解决等号右边的倒数项。首先考虑只有一组训练样本 (x, y) 的情况，计算过程如下：

$$\begin{aligned}\frac{\partial}{\partial \theta_j} J(\theta) &= \frac{\partial}{\partial \theta_j} \frac{1}{2} (h_\theta(x) - y)^2 \\ &= 2 \times \frac{1}{2} (h_\theta(x) - y) \times \frac{\partial}{\partial \theta_j} (h_\theta(x) - y) \\ &= (h_\theta(x) - y) \times \frac{\partial}{\partial \theta_j} \left(\sum_{i=0}^n \theta_i x_i - y \right) \\ &= (h_\theta(x) - y) \times x_i\end{aligned}$$

对第 i 个训练样本，更新规则如下所示：

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)}$$

当训练集有超过一个训练样本的时候，有两种对这个规则的修改方法。

1. 批量梯度下降法 (batch gradient descent)

批量梯度下降法更新规则如下：

重复直至收敛 {

$$\theta_j := \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_\theta(x^{(i)})) x_j^{(i)} \quad (\text{对每个 } j)$$

}

批量梯度下降法：更新规则实际就是对原始的代价函数 J 进行简单的梯度下降。在每个步长内检查整个训练集中的所有样本。

然而，由于每次进行一个批量梯度下降算法，都需要遍历整个训练集。当训练集 m 变得很大时，因此引起的性能开销就很不划算了。

2. 随机梯度下降法 (stochastic gradient descent)

随机梯度下降法更新规则如下：

循环： {

i=1:m {

$$\theta_j := \theta_j - \alpha (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)} \quad (\text{对每个 } j)$$

}

}

随机梯度下降法：我们对整个训练集进行循环遍历，每次遇到一个训练样本，根据每个单一训练样本的误差梯度来对参数进行更新。即在每个步长内使用单个样本。对于训练集很大的数据，随机梯度下降法速度很快。

然而，随机梯度下降法通常不会精确地收敛到全局的最小值，会在最小值附近震荡，通常情况下在最小值附近的这些值大多数其实也足够逼近了，足以满足咱们的精度要求，所以也可以用。不过更常见的情况是我们事先对数据集已经有了描述，并且选择了合适的学习速率 α ；然后来运行随机梯度下降，同时更新学习速率 α 随着算法的运行逐渐趋于 0，这样也能保证我们的参数会收敛到最小值。基于以上种种原因，通常推荐使用的都是随机梯度下降法，而不是批量梯度下降法，尤其是在训练用的数据集比较大的情况下。

2.3 正规方程 (The normal equations)

1. 矩阵倒数

假如有一个函数 $f : R_{m \times n} \rightarrow R$ 从 $m \times n$ 大小的矩阵映射到实数域，那么就可以定义当矩阵为 A 的时候有导函数 f 如下所示：

$$\nabla_A f(A) = \begin{bmatrix} \frac{\partial f}{\partial A_{11}} & \vdots & \frac{\partial f}{\partial A_{1n}} \\ \vdots & \vdots & \vdots \\ \frac{\partial f}{\partial A_{m1}} & \vdots & \frac{\partial f}{\partial A_{mn}} \end{bmatrix}$$

因此，这个梯度 $\nabla_A f(A)$ 本身也是一个 $m \times n$ 的矩阵，其中的第 (i, j) 个元素是 $\frac{\partial f}{\partial A_{ij}}$ 。

2. 矩阵的迹

对于一个给定的 $n \times n$ 方形矩阵 A，它的迹定义为对角项和：

$$trA = tr(A) = \sum_{i=1}^n A_{ii}$$

- 迹函数的线性

$$\begin{aligned} tr(A + B) &= trA + trB \\ tr(aA) &= atrA \end{aligned}$$

- 迹函数的矩阵乘积

$$\begin{aligned} trA &= trA^T \\ tr(AB) &= tr(BA) \\ tr(ABC) &= tr(CAB) \end{aligned}$$

- 迹函数的矩阵导数

$$\nabla_A trAB = B^T \quad (1)$$

$$\nabla_{A^T} f(A) = (\nabla_A f(A))^T \quad (2)$$

$$\nabla_A trABA^TC = CAB + C^T AB^T \quad (3)$$

$$\nabla_A |A| = |A|(A^{-1})^T \quad (4)$$

结合 (2) 和 (3)，我们可以得到公式 (5)：

$$\nabla_A \text{tr}ABA^TC = B^TA^TC^T + BA^TC \quad (5)$$

3. 最小二乘法

首先我们把 J 用矩阵一向量的记号来重新你描述。给定一个训练集，把设计矩阵 (**design matrix**) X 设置为一个 $m \times (n + 1)$ 矩阵 (考虑到 θ_0)，这个矩阵里包含了训练样本的输入值作为一行：

$$X = \begin{bmatrix} -(x^{(1)})^T - \\ -(x^{(2)})^T - \\ \vdots \\ -(x^{(m)})^T - \end{bmatrix}$$

然后，我们设 \vec{y} 是一个 m 维向量，包含了训练样本中的所有目标值：

$$y = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

由于 $h_\theta(x^{(i)}) = (x^{(i)})^T\theta$ ，可以证明存在下面这种等量关系：

$$\begin{aligned} X\theta - \vec{y} &= \begin{bmatrix} (x^{(1)})^T\theta \\ \vdots \\ (x^{(m)})^T\theta \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \\ &= \begin{bmatrix} h_\theta(x^1) - y^{(1)} \\ \vdots \\ h_\theta(x^m) - y^{(m)} \end{bmatrix} \end{aligned}$$

利用点积公式，可以推出：

$$\frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) = \frac{1}{2} \sum_{i=1}^m (h_\theta(x^{(i)}) - y^{(i)})^2$$

最后，要让 J 的值最小，就要找到导数为 0 的点，因此就有：

$$\begin{aligned}\nabla_\theta J(\theta) &= \nabla_\theta \frac{1}{2}(X\theta - \vec{y})^T(X\theta - \vec{y}) \\ &= \frac{1}{2} \nabla_\theta (\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T \vec{y}) \\ &= \frac{1}{2} \nabla_\theta \text{tr}(\theta^T X^T X\theta - \theta^T X^T \vec{y} - \vec{y}^T X\theta + \vec{y}^T \vec{y}) \\ &= \frac{1}{2} \nabla_\theta (\text{tr}\theta^T X^T X\theta - 2\text{tr}\vec{y}^T X\theta) \\ &= \frac{1}{2} (X^T X\theta + X^T X\theta - 2X^T \vec{y}) \\ &= X^T X\theta - X^T \vec{y}\end{aligned}$$

在第三步，我们利用定理：一个实数的迹就是这个实数本身；第四步中，用到了 $\text{tr}A = \text{tr}A^T$ 这个定理；第五步中，利用等式（1）和（5），其中 $A^T = \theta$ 、 $B = B^T = X^T X$ 、 $C = I$ 。

然后令导数为 0，得到下面的法线方程：

$$\begin{aligned}X^T X\theta &= X^T \vec{y} \\ \theta &= (X^T X)^{-1} X^T \vec{y}\end{aligned}$$

所以让 $J(\theta)$ 取得最小值的 θ 就是 $\theta = (X^T X)^{-1} X^T \vec{y}$ 。

2.3 概率解释 (Probabilistic interpretation)

当面对回归问题时，我们会有疑问，为什么选择最小二乘法作为代价函数？在本节中，我们会给出一系列的概率基本假设，基于这些假设，就可以推出最小二乘法回归是一种非常自然的算法。

1. 首先我们假设目标变量和输入值存在下面这种等量关系：

$$y^{(i)} = \theta^T x^{(i)} + \epsilon^{(i)}$$

其中 $\epsilon^{(i)}$ 是误差项，用以描述建模忽略的变量和随机噪声。

2. 其次假设 $\epsilon^{(i)}$ 服从高斯分布，即 $\epsilon^{(i)} \sim N(0, \sigma^2)$ ，概率密度函数就是：

$$p(\epsilon^{(i)}) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(\epsilon^{(i)})^2}{2\sigma^2}\right)$$

这也意味着在给定参数 θ 下， $y^{(i)}$ 相对于输入 $x^{(i)}$ 服从高斯概率分布：

$$\begin{aligned}y^{(i)} | x^{(i)}; \theta &\sim N(\theta^T x^{(i)}, \sigma^2) \\ p(y^{(i)} | x^{(i)}; \theta) &= \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right)\end{aligned}$$

根据上面讲的设计矩阵 X , 包含了全部的 $x^{(i)}$, 然后再给定 θ , 那么 $y^{(i)}$ 的分布是什么样子呢? 数据的概率以 $p(\vec{y}|X; \theta)$ 的形式给出。为了衡量 θ 对向量 \vec{y} 的贡献, 我们定义一个似然函数 (**likelihood function**) $L(\theta)$:

$$\begin{aligned} L(\theta) &= L(\theta; X, \vec{y}) \\ &= p(\vec{y}|X; \theta) \\ &= \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \end{aligned}$$

θ "越好", $L(\theta)$ 分布越集中, 值越大, 所以接下来我们求最大值, 此次对应的 θ 就是最佳的, 这种方法称为极大似然法 (**maximum likelihood**)。我们使用对数方便计算:

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \log \prod_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= \sum_{i=1}^m \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(y^{(i)} - \theta^T x^{(i)})^2}{2\sigma^2}\right) \\ &= m \times \log \frac{1}{\sqrt{2\pi}\sigma} - \frac{1}{\sigma^2} \times \frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2 \end{aligned}$$

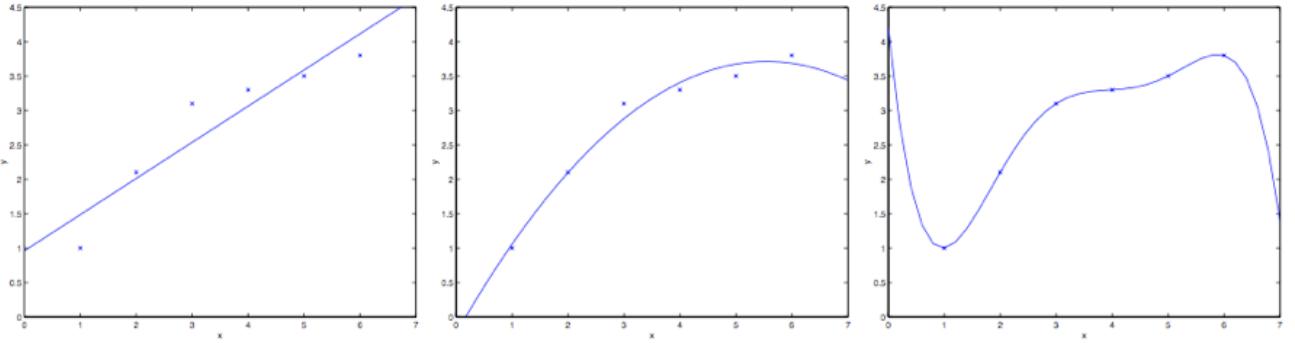
因此, 对 $l(\theta)$ 取得最大值也就意味着下面这个子式取到最小值:

$$\frac{1}{2} \sum_{i=1}^m (y^{(i)} - \theta^T x^{(i)})^2$$

到这里我们能发现这个子式实际上就是 $J(\theta)$, 也就是最原始的最小二乘函数。总结一下也就是: 在对数据进行概率假设的基础上, 最小二乘回归得到的 θ 和最大似然法估计的 θ 是一致的。所以这是一系列的假设, 其前提是认为最小二乘回归 (least-squares regression) 能够被判定为一种非常自然的方法。

2.4 局部加权线性回归 (**Locally weighted linear regression**)

假如问题还是根据从实数域内取值的 $x \in R$ 来预测 y 。左下角的图显示了使用 $y = \theta_0 + \theta_1 x$ 来对一个数据集进行拟合。我们明显能看出这个数据的趋势并不是一条严格的直线, 所以用直线进行的拟合就不是好的方法。



那么这次不用直线，而增加一个二次项，用 $y = \theta_0 + \theta_1 x + \theta_2 x^2$ 来拟合，(看中间的图)很明显，我们对特征补充得越多，效果就越好。不过，增加太多特征也会造成危险的：最右边的图就是使用了五次多项式 $y = \sum_{j=0}^5 \theta_j x^j$ 来进行拟合。看图就能发现，虽然这个拟合曲线完美地通过了所有当前数据集中的数据，但我们明显不能认为这个曲线是一个合适的预测工具，比如针对不同的居住面积 x 来预测房屋价格 y 。先不说这些特殊名词的正规定义，咱们就简单说，最左边的图像就是一个欠拟合 (**under fitting**) 的例子，比如明显能看出拟合的模型漏掉了数据集中的结构信息；而最右边的图像就是一个过拟合 (**over fitting**) 的例子。

正如前文谈到的，也正如上面这个例子展示的，一个学习算法要保证能良好运行，特征的选择是非常重要的。在本节，咱们就简要地讲一下局部加权线性回归 (locally weighted linear regression , LWR)，这个方法是假设有足够多的训练数据，对不太重要的特征进行一些筛选。在原始版本的线性回归算法中，要对一个查询点 x 进行预测，比如要衡量 $h(x)$ ，要经过下面的步骤：

1. 使用参数 θ 进行拟合，让数据集中的值与拟合算出的值的差值平方 $(y^{(i)} - \theta^T x^{(i)})^2$ 最小（最小二乘法的思想）；参数 θ 一旦确定，无论选择哪个 x ，都不会变。
2. 输出 $\theta^T x$ 。

相应地，在LWR局部加权线性回归方法中，步骤如下：

1. 使用参数 θ 进行拟合，让加权距离 $w^{(i)}(y^{(i)} - \theta^T x^{(i)})^2$ 最小；此时参数 θ 随着 $w^{(i)}$ 变化，而 $w^{(i)}$ 随着 x 变化，即对于不同的 x ，每次衡量 $h(x)$ 时，都要重新计算 θ ；**训练集要保留**
2. 输出 $\theta^T x$ 。

上面式子中的 $w^{(i)}$ 是非负的权值。直观点说就是，如果对应某个 i 的权值 $w^{(i)}$ 特别大，那么在选择拟合参数 θ 的时候，就要尽量让这一点的 $(y^{(i)} - \theta^T x^{(i)})^2$ 最小。而如果权值 $w^{(i)}$ 特别小，那么这一点对应的 $(y^{(i)} - \theta^T x^{(i)})^2$ 就基本在拟合过程中忽略掉了。对于权值的选取可以使用下面这个比较标准的公式：

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^2}{2\tau^2}\right)$$

如果 x 是有值的向量，那就要对上面的式子进行泛化，得到的是

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^T (x^{(i)} - x)}{2\tau^2}\right)$$

或者：

$$w^{(i)} = \exp\left(-\frac{(x^{(i)} - x)^T \Sigma^{-1} (x^{(i)} - x)}{2}\right)$$

这就看是选择用 τ 还是 Σ 。

要注意的是，权值是依赖每个特定的点 x 的，而这些点正是我们要去进行预测评估的点。此外，如果 $|x^{(i)} - x|$ 非常小，那么权值 $w^{(i)}$ 就接近 1；反之如果 $|x^{(i)} - x|$ 非常大，那么权值 $w^{(i)}$ 就变小。所以可以看出， θ 的选择过程中，查询点 x 附近的训练样本有更高得多的权值。（ θ is chosen giving a much higher “weight” to the (errors on training examples close to the query point x .)）（还要注意，当权值的方程的形式跟高斯分布的密度函数比较接近的时候，权值和高斯分布并没有什么直接联系，尤其是当权值不是随机值，且呈现正态分布或者其他形式分布的时候。）随着点 $x^{(i)}$ 到查询点 x 的距离降低，训练样本的权值的也在降低，参数 τ 控制了这个降低的速度； τ 也叫做带宽参数，这个也是在你的作业中需要来体验和尝试的一个参数。局部加权线性回归是咱们接触的第一个非参数算法。而更早之前咱们看到的无权重的线性回归算法就是一种参数学习算法，因为有固定的有限个数的参数（也就是 θ_i ），这些参数用来拟合数据。我们对 θ_i 进行了拟合之后，就把它们存了起来，也就不再需要再保留训练数据样本来进行更进一步的预测了。与之相反，如果用局部加权线性回归算法，我们就必须一直保留着整个训练集。这里的非参数算法中的“non-parametric”是粗略地指：为了呈现出假设 h 随着数据集规模的增长而线性增长，我们需要以一定顺序保存一些数据的规模。

3. 分类和逻辑回归 (Classification and logistic regression)

分类问题与回归问题很像，只不过我们要预测的 y 的值仅局限于若干个有限的离散值。接下来我们关注简单的二值化分类问题。

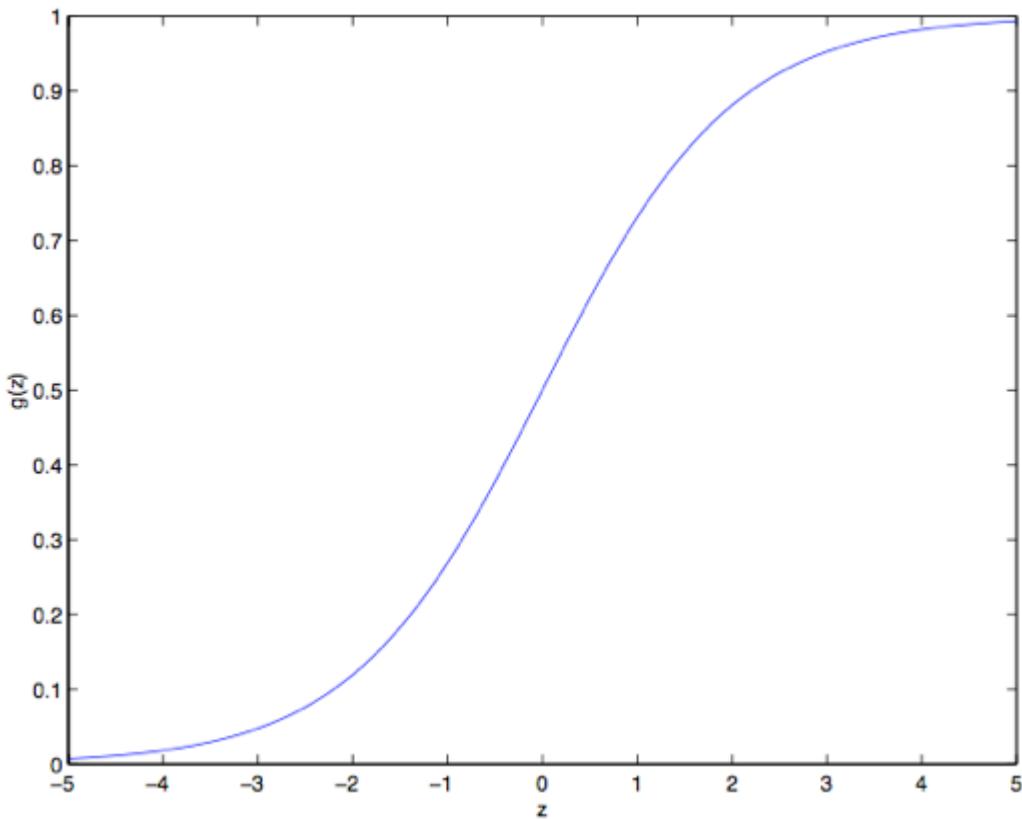
3.1 逻辑回归 (logistic regression)

首先不同于分类问题，我们改变假设函数 $h_\theta(x)$ 的形式，用一个逻辑函数表示，如下：

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}}$$

其中 $g(z)$ 函数的方程、导数和图像如下所示：

$$\begin{aligned} g(z) &= \frac{1}{1 + e^{-z}} \\ \frac{d}{dz} g(z) &= \frac{d}{dz} \times \frac{1}{1 + e^{-z}} \\ &= \frac{1}{(1 + e^{-z})^2} \times (e^{-z}) \\ &= \frac{1}{(1 + e^{-z})} \times \left(1 - \frac{1}{(1 + e^{-z})}\right) \\ &= g(z)(1 - g(z)) \end{aligned}$$



那么给定了逻辑回归模型，我们如何选择一个合适的 θ 呢？在上述的概率解释中，我们得知可以通过极大似然估计来推出。故我们先写出 θ 下的 $y^{(i)}$ 的概率分布，然后写出似然函数，然后求最大似然估计，可以求倒数、利用梯度求 θ 的变化规律。

1. 概率分布

$$p(y^{(i)} | x^{(i)}; \theta) = (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}}$$

2. 似然函数，假设 m 个训练样本是各自独立生成的，即通过累乘可以得到似然函数

$$\begin{aligned} \max : L(\theta) &= \prod_{i=1}^m p(y^{(i)} | x^{(i)}; \theta) \\ &= \prod_{i=1}^m (h_\theta(x^{(i)}))^{y^{(i)}} (1 - h_\theta(x^{(i)}))^{1-y^{(i)}} \end{aligned}$$

跟以前一样，取个对数更容易计算最大值：

$$\begin{aligned} l(\theta) &= \log L(\theta) \\ &= \sum_{i=1}^m [y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log(1 - h_\theta(x^{(i)}))] \end{aligned}$$

3. 如何取得最大似然函数呢？我们使用梯度上升法（**gradient ascent**），写成向量的形式，也就是
 $\theta := \theta + \alpha \nabla_{\theta} l(\theta)$ 。

$$\begin{aligned}
\frac{\partial}{\partial \theta_j} l(\theta) &= \sum_{i=1}^m [y^{(i)} \times \frac{1}{h_{\theta}(x^{(i)})} - (1-y) \times \frac{1}{h_{\theta}(x^{(i)})}] \frac{\partial}{\partial \theta_j} g(\theta^T x^{(i)}) \\
&= \sum_{i=1}^m [y^{(i)} \times \frac{1}{h_{\theta}(x^{(i)})} - (1-y) \times \frac{1}{h_{\theta}(x^{(i)})}] g(\theta^T x^{(i)}) (1 - g(\theta^T x^{(i)})) \frac{\partial}{\partial \theta_j} \theta^T x^{(i)} \\
&= \sum_{i=1}^m [y^{(i)} \times (1 - g(\theta^T x^{(i)})) - (1-y) \times g(\theta^T x^{(i)})] x_j^{(i)} \\
&= \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)} \\
\theta_j &:= \theta_j + \alpha \sum_{i=1}^m (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}
\end{aligned}$$

这里写累和符号是批量梯度法，不写累和的话是随机梯度法，一般选择随机梯度法，所以公式可以简化为如下形式：

$$\theta_j := \theta_j + \alpha (y^{(i)} - h_{\theta}(x^{(i)})) x_j^{(i)}$$

3.2 题外话：感知器学习算法（The perceptron learning algorithm）

现在咱们来岔开一下话题，简要地聊一个算法，这个算法的历史很有趣，并且之后在我们讲学习理论的时候还要讲到它。设想一下，对逻辑回归方法修改一下，“强迫”它输出的值要么是 0 要么是 1。要实现这个目的，很自然就应该把函数 g 的定义修改一下，改成一个阈值函数（**threshold function**）：

$$g(z) = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

如果我们还像之前一样令 $h_\theta(x) = g(\theta^T x)$, 但用刚刚上面的阈值函数作为 g 的定义, 然后如果我们用了下面的更新规则:

$$\theta_j := \theta_j + \alpha(y^{(i)} - h_\theta(x^{(i)}))x_j^{(i)}$$

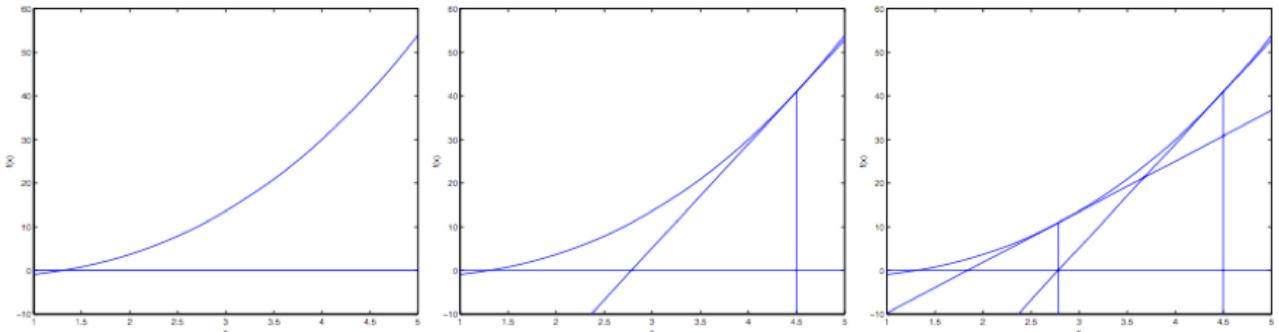
这样我们就得到了感知器学习算法。但一定要注意, 虽然这个感知器学习算法可能看上去表面上跟我们之前讲的其他算法挺相似, 但实际上这是一个和逻辑回归以及最小二乘线性回归等算法在种类上都完全不同的算法; 更重要的是, 很难对感知器的预测赋予有意义的概率解释, 也很难作为一种极大似然估计算法来推出感知器学习算法。

3.3 取 $l(\theta)$ 最大值的另外一个算法—牛顿法

再回到用 S 型函数 $g(z)$ 来进行逻辑回归的情况, 咱们来讲一个让 $l(\theta)$ 取最大值的另一个算法。开始之前, 咱们先想一下求一个方程零点的牛顿法。假如我们有一个从实数到实数的函数 $f : R \rightarrow R$, 然后要找到一个 θ , 来满足 $f(\theta) = 0$, 其中 $\theta \in R$ 是一个实数。牛顿法就是对 θ 进行如下的更新:

$$\theta := \theta - \frac{f(\theta)}{f'(\theta)}$$

这个方法可以通过一个很自然的解释, 我们可以把它理解成用一个线性函数来对函数 f 进行逼近, 这条直线是 f 的切线, 而猜测值是 θ , 解的方法就是找到线性方程等于零的点, 把这一个零点作为 θ 设置给下一次猜测, 然后依次类推。下面是对牛顿法的图解:



牛顿法的给出的解决思路是让 $f(\theta) = 0$ 。如果咱们要用它来让函数 l 取得最大值能不能行呢? 函数 l 的最大值的点应该对应着是它的导数 $l'(\theta)$ 等于零的点。所以通过令 $f(\theta) = l'(\theta)$, 咱们就可以同样用牛顿法来找到 l 的最大值, 然后得到下面的更新规则:

$$\theta := \theta - \frac{l'(\theta)}{l''(\theta)}$$

最后, 在咱们的逻辑回归背景中, θ 是一个有值的向量, 所以我们要对牛顿法进行扩展来适应这个情况。牛顿法进行扩展到多维情况, 也叫牛顿-拉普森法 (Newton-Raphson method), 如下所示:

$$\theta := \theta - H^{-1} \nabla_\theta l(\theta)$$

上面这个式子中的 $\nabla_\theta l(\theta)$ 和之前的样例中的类似, 是关于 θ_i 的 $l(\theta)$ 的偏导数向量; 而 H 是一个 $n \times n$ 矩阵, 实际上如果包含截距项的话, 应该是 $(n+1) \times (n+1)$, 也叫做 Hessian, 其详细定义是:

$$H_{ij} = \frac{\partial^2 l(\theta)}{\partial \theta_i \partial \theta_j}$$

牛顿法通常都能比（批量）梯度下降法收敛得更快，而且达到最小值所需要的迭代次数也低很多。然而，牛顿法中的单次迭代往往要比梯度下降法的单步耗费更多的性能开销，因为要查找和转换一个 $n \times n$ 的 Hessian 矩阵；不过只要这个 n 不是太大，牛顿法通常就还是更快一些。当用牛顿法来在逻辑回归中求似然函数 $l(\theta)$ 的最大值的时候，得到这一结果的方法也叫做 Fisher 评分（Fisher scoring）。

4. 广义线性模型 (Generalized Linear Models)

到目前为止，我们已经看过了回归案例和分类案例。在回归的案例中，我们得到的函数是： $y|x; \theta - N(\mu, \sigma)$ ；而在分类的案例中，我们得到的函数是： $y|x; \theta - Bernoulli(\phi)$ 。这里面的 μ 和 ϕ 分别是 x 和 θ 的某种函数。在本节中，我们会发现这两种方法都是一个更广泛使用的模型的特例，这种更广泛使用的模型就叫做广义线性模型。我们还会讲一下广义线性模型中的其他模型是如何推出的，以及如何应用到其他的分类和回归问题上。

4.1 指数族 (The exponential family)

在学习 GLMs 之前，我们要先定义指数族分布 (exponential family distributions)。如果一个分布能用下面的方式写出来，我们就说这个分布属于**指数族**：

$$p(y; \eta) = b(y) \exp(\eta^T T(y) - a(\eta))$$

上面的式子中，各参数如下所示：

- η : 该分布的自然参数 (natural parameter) (我的理解：不一定对： $\eta = \theta^T x$)
- $T(y)$: 充分统计量 (sufficient statistic)，一般 $T(y) = y$ 。
- $b(y)$:
- $a(\eta)$: 一个对数分割函数 (log partition function)

对 T 、 b 、 a 固定选择后，就定义了一个用 η 进行参数化的分布族；通过改变 η ，我们就能得到这个分布族中的不同分布。

现在看到的高斯分布和伯努利分布都属于指数分布族。首先我们来分析**伯努利分布**： $\phi - (x; \theta)$

$$\begin{aligned} p(y; \phi) &= \phi^y (1 - \phi)^{1-y} \\ &= \exp(y \log \phi + (1 - y) \log(1 - \phi)) \\ &= \exp\left(\log\left(\frac{\phi}{1 - \phi}\right)\right)y + \log(1 - \phi) \end{aligned}$$

所以，我们可以得到自然参数 $\eta = \log\frac{\phi}{1-\phi}$ 。有趣的是，如果我们反转定义，求解 ϕ ，就会发现一下有趣的事：

$$\begin{aligned} \phi &= \frac{1}{1 + e^{-\eta}} \\ T(y) &= y \\ b(y) &= 1 \\ a(\eta) &= -\log(1 - \phi) = \log(1 + e^\eta) \end{aligned}$$

观察上面的形式，可以看到 ϕ 正是我们之前见过的 S 型函数。而且上面的 T 、 b 、 a 说明了伯努利分布可以写成指数族的形式。

注意：这里 ϕ 可以看成是关于 $(x; \theta)$ 的函数，令 $\eta = \theta^T x$ ，我们可以得到这个方程：**观察最右边**

$$\phi(x; \theta) = \frac{1}{1 + e^{-\eta}} = \frac{1}{1 + e^{-\theta^T x}} = h_\theta(x)$$

这个就是逻辑回归的方程形式，下面的线性回归也可以写成类似的形式。

接下来我们来看高斯分布， $\mu - (x; \theta)$ ，在推导线性回归的概率解释时，我们知道 σ 的选择对我们最终的 θ 和 $h_\theta(x)$ 都没有影响。所以我们可以给 σ^2 取一个任意值。为了简化推导过程，就令 $\sigma^2 = 1$ 。然后就有了下面的等式：

$$\begin{aligned} p(y; \mu) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}(y - \mu)^2\right) \\ &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \times \exp(\mu y - \frac{1}{2}\mu^2) \end{aligned}$$

所以，我们可以得到自然参数 $\eta = \mu$ 。类比上面的伯努利分布，有以下结论：

$$\begin{aligned} \mu &= \eta \\ T(y) &= y \\ b(y) &= \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{1}{2}y^2\right) \\ a(\eta) &= \frac{1}{2}\mu^2 = \frac{1}{2}\eta^2 \end{aligned}$$

注意： $\mu = \eta$ 正是线性回归的方程，我的解释如下，类比逻辑回归：**观察最右边**

$$\mu(x; \theta) = \eta = \theta^T x = h_\theta(x)$$

等价如下推论：第一行左边是假设的分布的均值，符合假设 2；第二行参数替换，符合假设 3。

$$\begin{aligned} \phi \text{ or } \mu &= f(\eta) && \text{我们要求的假设的一种形式} \\ &= f(\theta^T x) && \text{利用 } \eta = \theta^T x \\ &= h_\theta(x) && \text{假设 } h \text{ 的直观形式} \end{aligned}$$

指数族里面还有很多其他的分布：多项式分布（multinomial）；泊松分布（Poisson），用于对计数类数据进行建模； γ 和 指数分布（the gamma and the exponential），这个用于对连续的、非负的随机变量进行建模，例如时间间隔； β 和 狄利克雷分布（the beta and the Dirichlet），这个是用于概率的分布。

4.2 构建广义线性模型（Constructing GLMs）

设想一个分类或者回归问题，要预测一些随机变量 y 的值，作为 x 的一个函数。要想导出适用于这个问题的广义线性模型，就需要对我们的模型、即给定 x 下的 y 的条件分布，来做出以下三个假设（与我上面的推论类似）：

1. $y|x; \theta \sim ExponentialFamily(\eta)$ ：即给定 x 、 θ ， y 的分布属于指数族分布，是一个参数为 η 的指数族。
2. 给定 x ，目的是预测对应这个 x 的充分统计量的期望值，即 $E[T(y)|x]$ 。**一个好的期望值对应的 θ 和分布公式使我们的目标。**咱们的例子中绝大部分情况都是 $T(y) = y$ ，这也就意味着我们的学习假设 h 输出的预测值 $h(x)$ 要满足 $h(x) = E[y|x]$ 。（然后我们假设的高斯分布均值为 μ ，伯努利分布均值为 ϕ ，符合上面的推论。）
3. 自然参数 η 和输入值 x 线性相关： $\eta = \theta^T x$ ；或者如果 η 是有值的向量，则有 $\eta_i = \theta_i^T x$ 。

上面几个假设中，第三个可能看上去证明得最差，所以也更适合把这第三个假设看作是一个我们在设计广义线性模型时候的一种“设计选择”，而不是一个假设。那么这三个假设 or 设计中，可以推导出一个非常适合的学习算法类别，也就是**广义线性模型 GLMs**，这个模型有很多特别友好又理想的性质，而且很容易学习。此外，这类模型对一些关于 y 的分布的不同类型建模来说效率都很高；例如，我们下面介绍用广义线性模型来推导的最小二乘法和逻辑回归。

1. 线性回归（Linear Regression）

我们这一节要讲的是普通最小二乘法实际上是广义线性模型中的一个特例，设想如下的背景设置：目标变量 y 是连续的，然后我们将给定 x 的 y 的分布以高斯分布 $N(\mu, \sigma^2)$ 来建模。这样我们根据指数族分布和三个假设条件，可以得到下面的等式：

$$\begin{aligned} h_{\theta}(x) &= E[y|x; \theta] && \text{假设 2} \\ &= \mu && \text{高斯分布均值} \\ &= \eta && \text{假设 1} \\ &= \theta^T x && \text{假设 3} \end{aligned}$$

即推导出线性回归的假设 h ，然后利用梯度下降法求解。

2. 逻辑回归 (Logistic Regression)

接下来我们分析逻辑回归，选择伯努利分布来对给定 x 的 y 的分布进行建模，如果有 $y|x; \theta \sim Bernoulli(\phi)$ ，那么 $E[y|x; \theta] = \phi$ 。所以就跟刚刚推导普通最小二乘法的过程类似，有以下等式：

$$\begin{aligned} h_{\theta}(x) &= E[y|x; \theta] \\ &= \phi \\ &= 1/(1 + e^{-\eta}) \\ &= 1/(1 + e^{-\theta^T x}) \end{aligned}$$

与刚才推导的高斯分布一致。

再解释一点术语，这里给出分布均值的函数 g 是一个关于自然参数的函数， $g(\eta) = E[T(y); \eta]$ ，这个函数也叫做**规范响应函数** (canonical response function)，它的反函数 g^{-1} 叫做**规范链接函数** (canonical link function)。因此，对于高斯分布来说，它的规范响应函数正好就是识别函数 (identify function)；而对于伯努利分布来说，它的规范响应函数则是逻辑函数 (logistic function)。

注：很多教科书用 g 表示链接函数，而用反函数 g^{-1} 来表示响应函数；但是咱们这里用的是反过来的，这是继承了早期的机器学习中的用法，我们这样使用和后续的其他课程能够更好地衔接起来。

3. Softmax 回归

咱们来看一个广义线性模型的例子吧。设想这样一个分类问题，其中响应变量 y 的取值可以是 k 值中的任意一个，即 $y \in \{1, 2, \dots, k\}$ 。因此我们就要用**多项式分布** (multinomial distribution) 来进行建模。下面我们就通过这种多项式分布来推出一个广义线性模型。要实现这一目的，首先还是要把多项式分布也用指数族分布来进行描述。

1. 用 $k - 1$ 个参数对概率分布进行参数化；

虽然 y 有 k 个取值，但考虑到概率分布之和要等于 1，所以我们仅需要 $k - 1$ 个参数 ϕ_1, \dots, ϕ_k 进行描述，如下：

$$\begin{aligned} \phi_1 + \phi_2 + \dots + \phi_k &= 1 \\ \phi_k &= 1 - \sum_{i=1}^{k-1} \phi_i \\ p(y = i; \phi) &= \phi_i, \quad p(y = k; \phi) = 1 - \sum_{i=1}^{k-1} \phi_i \end{aligned}$$

2. 定义充分统计量 $T(y)$;

$$T(1) = \begin{bmatrix} 1 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(2) = \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, T(3) = \begin{bmatrix} 0 \\ 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix}, T(k-1) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}, T(k) = \begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

这次和之前的样例都不一样，现在， $T(y)$ 是一个 $k - 1$ 维的向量，而且有 k 个样式，而不是一个实数了。把样式统一起来表示为：向量 $T(y)$ 中的第 i 个元素可以写成 $T(y)_i$ 。

下面介绍一个非常有用的符号， $1\{\cdot\}$ 指示函数 (indicator function)，这个 1 可以等效为 `if`，如果参数为真，则等于 1；反之则等于 0。则向量 $T(y)$ 可以简化为 $T(y)_i = 1\{y = i\}$ 。

3. 表示 $E[T(y)|x]$ ，1、2 步可以表述为为假设 1 准备，这一步是为假设 2 准备；

$$E[T(y)_i|x] = P(y = i) = \phi_i$$

4. 准备就绪，可以把多项式写成指数族分布了；

$$\begin{aligned} p(y; \phi) &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \cdots \phi_{k-1}^{1\{y=k-1\}} \phi_k^{1\{y=k\}} \\ &= \phi_1^{1\{y=1\}} \phi_2^{1\{y=2\}} \cdots \phi_{k-1}^{1\{y=k-1\}} \phi_k^{1-\sum_{i=1}^{k-1} 1\{y=i\}} \\ &= \phi_1^{T(y)_1} \phi_2^{T(y)_2} \cdots \phi_{k-1}^{T(y)_{k-1}} \phi_k^{1-\sum_{i=1}^{k-1} T(y)_i} \\ &= \exp(T(y)_1 \log(\phi_1) + T(y)_2 \log(\phi_2) + \dots + T(y)_{k-1} \log(\phi_{k-1}) + (1 - \sum_{i=1}^{k-1} T(y)_i) \log(\phi_k)) \\ &= \exp\left(\sum_i^{k-1} \frac{\log(\phi_i)}{\log(\phi_k)} T(y)_i + \log(\phi_k)\right) \\ &= b(y) \exp(\eta^T T(y) - a(\eta)) \\ &= p(y; \eta) \end{aligned}$$

观察第 5 行，这是一个内积表达式，所以我们得到的 η 是一个向量，各参数表达式如下：

$$\eta = \begin{bmatrix} \log(\phi_1/\phi_k) \\ \log(\phi_2/\phi_k) \\ \vdots \\ \log(\phi_{k-1}/\phi_k) \end{bmatrix}$$

$$b(y) = 1$$

$$a(\eta) = -\log(\phi_k)$$

这样我们把这个多项式分布作为一个指数族写了出来，与 $i (i \in \{1, 2, \dots, k-1\})$ 对应的链接函数为：

$$\eta_i = \log \frac{\phi_i}{\phi_k}$$

$$\eta_k = \log \frac{\phi_k}{\phi_k} = 0 \quad \text{方便计算}$$

5. 计算响应函数；

$$\begin{aligned}
e^{\eta_i} &= \frac{\phi_i}{\phi_k} \\
\phi_k e^{\eta_i} &= \phi_i \\
\phi_k \sum_{i=1}^k e^{\eta_i} &= \sum_{i=1}^k \phi_i = 1 \\
\phi_k &= \frac{1}{\sum_{i=1}^k e^{\eta_i}} = \frac{1}{1 + \sum_{i=1}^{k-1} e^{\eta_i}}
\end{aligned}$$

这样就得到了响应函数：

$$\phi_i = \frac{e^{\eta_i}}{\sum_{j=1}^k e^{\eta_j}} = \frac{e^{\eta_i}}{1 + \sum_{j=1}^{k-1} e^{\eta_j}}$$

上面这个函数从 η 映射到了 ϕ , 称为 Softmax 函数。

6. 为了完成我们的建模, 利用假设 3, 也就是 η_i 是一个 x 的线性函数。所以就有了 $\eta_i = \theta_i^T x$ (*for* : $i = 1, \dots, k - 1$), 其中的 $\theta_1, \dots, \theta_{k-1} \in R^{n+1}$ 就是我们建模的参数。为了表述方便, 我们这里还是定义 $\theta_k = 0$, 这样就有 $\eta_k = \theta_k^T x = 0$, 跟前文提到的相符。因此, 我们的模型假设了给定 x 的 y 的条件分布为:

$$\begin{aligned}
p(y = i|x; \theta) &= \phi_i \\
&= \frac{e^{\eta_i}}{1 + \sum_{j=1}^{k-1} e^{\eta_j}} \\
&= \frac{e^{\theta_i^T x}}{1 + \sum_{j=1}^{k-1} e^{\theta_j^T x}}
\end{aligned}$$

这个适用于解决 $y \in \{1, \dots, k\}$ 的分类问题的模型, 就叫做 Softmax 回归。这种回归是对逻辑回归的一种扩展泛化。

7. 假设 (**hypothesis**) h 则有如下形式:

$$\begin{aligned}
h_\theta(x) &= E[T(y)|x; \theta] \\
&= E \left[\begin{array}{c} 1(y=1) \\ 1(y=2) \\ \vdots \\ 1(y=k-1) \end{array} \mid x; \theta \right] \\
&= E \left[\begin{array}{c} \phi_1 \\ \phi_2 \\ \vdots \\ \phi_{k-1} \end{array} \right] \\
&= E \left[\begin{array}{c} \frac{exp(\theta_1^T x)}{1+\sum_{j=1}^{k-1} exp(\theta_j^T x)} \\ \frac{exp(\theta_2^T x)}{1+\sum_{j=1}^{k-1} exp(\theta_j^T x)} \\ \vdots \\ \frac{exp(\theta_{k-1}^T x)}{1+\sum_{j=1}^{k-1} exp(\theta_j^T x)} \end{array} \right]
\end{aligned}$$

也就是说：我们的假设函数会对每一个 $i = 1, 2, \dots, k$, 给出 $p(y = i|x; \theta)$ 概率的估计值。（虽然咱们在前面假设的这个 $h_\theta(x)$ 只有 $k - 1$ 维，但很明显 $p(y = y|x; \theta)$ 可以通过用 1 减去其他所有项目概率的和来得到，即 $1 - \sum_{i=1}^{k-1} \phi_i$ 。）

注意：这里的 θ 是一个 $(n + 1) \times k$ 的矩阵， $n + 1$ 表示的是输入 x 的 n 维特征； k 表示输出 y 的分布；矩阵 θ 的每一列对应着一个 θ_i ，一行输入 x 与每一列 θ_i 相乘，输出一个 $1 \times k$ 的向量，其中最大的那个值的下标就表示 y 该取的那个值。例如判断一个数字是 0 – 9 中的哪一个，则 $k = 10$ ；如果输出的向量第一个元素比较大，就意味着这个数字很大概率是 0，即 $y = 0$ 。

参数拟合

这和我们之前推导的线性回归和逻辑回归的原始推导类似，假设我们有 m 个训练样本，我们可以先写出其似然函数的对数：

$$\begin{aligned} l(\theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) \\ &= \sum_{i=1}^m \log \prod_{l=1}^k \left(\frac{e^{\theta_l^T x^{(i)}}}{\sum_{j=1}^k e^{\theta_j^T x^{(i)}}} \right)^{1(y^{(i)}=l)} \end{aligned}$$

要得到上面等式的第二行，要用到步骤 4 中的设定 $p(y|x; \theta)$ 。现在就可以通过对 $l(\theta)$ 取最大值得到的 θ 而得到对参数的最大似然估计，使用的方法就可以用梯度上升法或者牛顿法了。

1. 生成式学习算法 (Generative Learning algorithms)

2. 高斯判别分析 (Gaussian Discriminant Analysis)

2.1 多元正态分布 (multivariate normal distribution)

2.2 高斯判别分析模型 (Gaussian Discriminant Analysis model)

2.3 讨论：高斯判别分析 (GDA) 与逻辑回归 (logistic regression)

3. 朴素贝叶斯法 (Naive Bayes)

3.1 拉普拉斯光滑 (Laplace smoothing)

3.2 针对文本分类的事件模型 (Event models for text classification)

1. 生成式学习算法 (Generative Learning algorithms)

- 判别式学习算法 (Discriminative Learning algorithms) : 对 $p(y|x; \theta)$ 进行建模, 得到 $h_\theta(x)$; 也就是给定 x 下的 y 的条件分布 (似然函数累乘的项就是条件分布), 以 θ 为参数。常见的判别式模型有:

- 线性回归 (Linear Regression)
- 逻辑回归 (Logistic Regression)
- 支持向量机 (SVM)
- 神经网络 (Neural Network)

- 生成式学习算法 (Generative Learning algorithms) : 对 $p(x|y)$ 和 $p(y)$ 进行建模, 也就是说对 $p(x, y)$ 进行建模, 即求 x, y 的联合分布 (似然函数累乘的项就是联合分布); 解释是: 一个生成模型在给定了样本所属的类的条件下, 对样本特征建立概率模型。在对 $p(x|y)$ 和 $p(y)$ 进行建模之后, 我们用贝叶斯公式 (Bayes rule) 可以

$$p(xy) = p(y|x)p(x) = p(x|y)p(y) \quad \text{条件概率公式}$$

$$p(y|x) = \frac{p(xy)}{p(x)} = \frac{p(x|y)p(y)}{p(x)}$$

这里的分母为: $p(x) = p(x|y=1)p(y=1) + p(x|y=0)p(y=0)$ (其实就是条件概率)。实际上如果我们计算 $p(y|x)$ 来进行预测, 那就并不需要去计算这个分母, 因为有下面的等式关系:

$$\begin{aligned} \arg \max_y p(y|x) &= \arg \max_y \frac{p(x|y)p(y)}{p(x)} \\ &= \arg \max_y p(x|y)p(y) \end{aligned}$$

常见的生成式模型有:

- 高斯判别分析 (GDA)
- 朴素贝叶斯法 (Naive Bayes)
- 隐形马尔可夫模型 (HMM)

2. 高斯判别分析 (Gaussian Discriminant Analysis)

我们要学的第一个生成学习算法是高斯判别分析 (**Gaussian Discriminant analysis, GDA**)。在这个模型里面，我们假设 $p(x|y)$ 是一个多元正态分布。所以咱们首先简单讲一下多元正态分布的一些特点，然后再继续讲 GDA。

2.1 多元正态分布 (multivariate normal distribution)

n 维多元正态分布，也叫做多变量高斯分布，参数为一个均值向量 $\mu \in R^n$ ，以及一个协方差矩阵 $\Sigma \in R^{n \times n}$ ，其中 $\Sigma >= 0$ 是一个对称的半正定矩阵。写成概率密度形式为：

$$p(x|y) \sim N(\mu, \Sigma)$$
$$p(x; \mu, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^T \Sigma^{-1} (x - \mu)\right)$$

在上面的等式中， $|\Sigma|$ 的意思是矩阵 Σ 的行列式 (determinant)。

随机变量 Z 是一个有值的向量 (vector-valued random variable)， Z 的协方差 (covariance) 定义是：

$$\begin{aligned} Cov(Z) &= E[(Z - E[Z])(Z - E[Z])^T] \\ &= E[ZZ^T] - (E[Z])(E[Z])^T \end{aligned}$$

如果 X 是一个多变量正态分布，即 $X \sim N(\mu, \Sigma)$ ，则有：

$$\begin{aligned} E[X] &= \mu \\ Cov(X) &= \Sigma \end{aligned}$$

2.2 高斯判别分析模型 (Gaussian Discriminant Analysis model)

假设我们有一个分类问题，其中输入特征 x 是一系列的连续随机变量，那就就可以使用高斯判别分析 (GDA) 模型，其中对 $p(x|y)$ 用多元正态分布来进行建模。这个模型为：

$$\begin{aligned} y &\sim Bernoulli(\phi) \\ x|y = 0 &\sim N(\mu_0, \Sigma) \\ x|y = 1 &\sim N(\mu_1, \Sigma) \end{aligned}$$

分布写成的具体形式为：

$$\begin{aligned} p(y) &= \phi^y (1 - \phi)^{1-y} \\ p(x|y = 0) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_0)^T \Sigma^{-1} (x - \mu_0)\right) \\ p(x|y = 1) &= \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_1)^T \Sigma^{-1} (x - \mu_1)\right) \end{aligned}$$

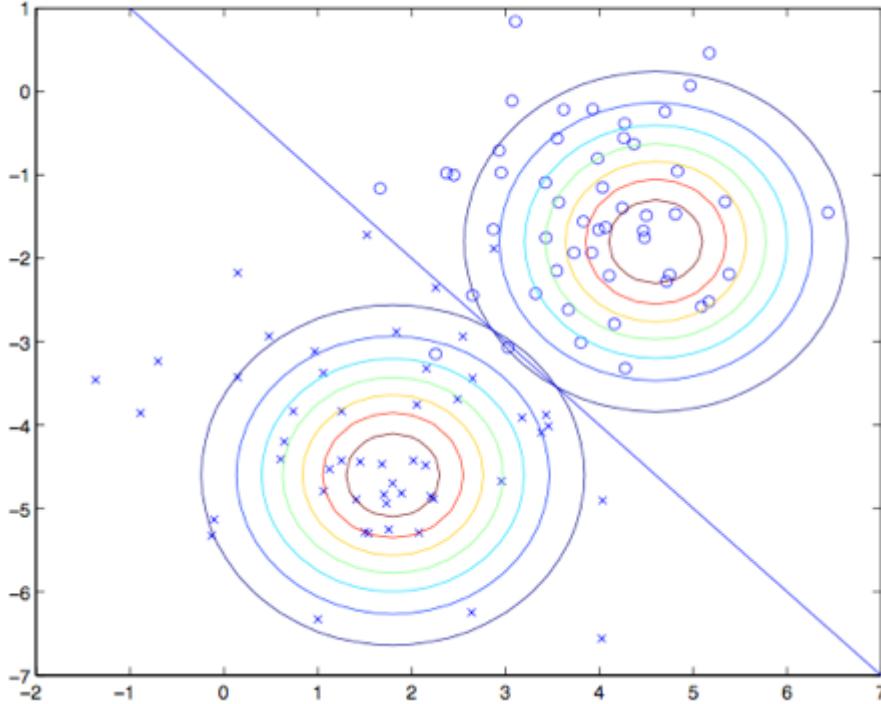
在上面的等式中，模型的参数包括 $\phi, \mu_0, \mu_1, \Sigma$ （要注意，虽然这里有两个不同方向的均值向量 μ_0 和 μ_1 ，针对这个模型还是一般只是用一个协方差矩阵 Σ ）。取对数的似然函数（log-likelihood）如下所示：

$$\begin{aligned} l(\phi, \mu_0, \mu_1, \Sigma) &= \log \prod_{i=1}^m p(x^{(i)}, y^{(i)}, \phi, \mu_0, \mu_1, \Sigma) \\ &= \log \prod_{i=1}^m p(x^{(i)} | y^{(i)}; \phi, \mu_0, \mu_1, \Sigma) p(y^{(i)}; \phi) \end{aligned}$$

通过使 l 取得最大值，找到对应的参数组合（即将 l 对四个参数依次求导，令导数为 0，求解相应的参数），然后就能找到该参数组合对应的最大似然估计，如下所示：

$$\begin{aligned} \phi &= \frac{1}{m} \sum_{i=1}^m \{y^{(i)} = 1\} \\ \mu_0 &= \frac{\sum_{i=1}^m 1\{y(i) = 0\}x^{(i)}}{\sum_{i=1}^m 1\{y(i) = 0\}} \\ \mu_1 &= \frac{\sum_{i=1}^m 1\{y(i) = 1\}x^{(i)}}{\sum_{i=1}^m 1\{y(i) = 1\}} \\ \Sigma &= \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu_{y^{(i)}})(x^{(i)} - \mu_{y^{(i)}})^T \end{aligned}$$

用图像化的方式来表述，这个算法可以按照下面的图示所表示：



图中展示的点就是训练数据集，图中的两个高斯分布就是针对两类数据各自进行的拟合。要注意这两个高斯分布的轮廓图有同样的形状和拉伸方向，这是因为他们都有同样的协方差矩阵 Σ ，但他们有不同的均值 μ_0 和 μ_1 。此外，图中的直线给出了 $p(y = 1|x) = 0.5$ 这条边界线。在这条边界的一侧，我们预测 $y = 1$ 是最大可能搞得结果；而另一侧，就估计 $y = 0$ 。

2.3 讨论：高斯判别分析（GDA）与逻辑回归（logistic regression）

高斯判别分析模型与逻辑回归有很有趣的相关性。如果我们把变量（quantity） $p(y=1|x; \phi, \mu_0, \mu_1, \Sigma)$ 作为一个 x 的函数，就会发现可以用如下的形式来表达：

$$p(y=1|x; \phi, \mu_0, \mu_1, \Sigma) = \frac{1}{1 + \exp(-\theta^T x)}$$

其中的 θ 是对 $\phi, \mu_0, \mu_1, \Sigma$ 的某种函数；这就是逻辑回归用来对 $p(y=1|x)$ 建模的形式。

这两个模型中什么时候该选哪一个呢？一般来说，高斯判别分析（GDA）和逻辑回归，对同一个训练集，可能给出的判断曲线是不一样的。哪一个更好呢？

事实上，二者的关系有如下推论：

- 如果 $p(x|y)$ 是一个多变量的高斯分布（且具有一个共享的协方差矩阵 Σ ），那么 $p(y|x)$ 则必然符合一个逻辑函数；
- 然而，反过来，这个命题是不成立的。例如：假如 $p(y|x)$ 是一个逻辑函数，这并不能保证 $p(x|y)$ 一定是一个多变量的高斯分布（实际上，第一个命题只要是可以指数族分布描述的，都可以推导出 $p(y|x)$ 符合一个逻辑函数）。

这就表面高斯判别模型比逻辑回归对数据进行更强的建模和假设（stronger modeling assumptions）。1) 这也就意味着，在这两种模型假设都可用的时候，高斯判别分析法去拟合数据是更好的，是一个更好的模型。尤其当 $p(x|y)$ 已经确定是一个高斯分布（有共享的协方差矩阵 Σ ），那么高斯判别分析是渐进有效的（asymptotically efficient）。2) 实际上，这也意味着，在面对非常大的训练集（训练样本规模 m 特别大）的时候，严格来说，可能就没有什么别的算法能比高斯判别分析更好（比如考虑到对 $p(y|x)$ 估计的准确度等等）。3) 所以在这种情况下就表明，高斯判别分析（GDA）是一个比逻辑回归更好的算法；再扩展一下，即便对于小规模的训练集，我们最终也会发现高斯判别分析（GDA）是更好的。

总结一下：当两种模型都可用的情况下，高斯判别分析有着更强的建模和假设：高斯判别分析拟合数据更好；而且无论大规模样本或者小规模样本，高斯判别分析都更好。

奈何事有正反，由于逻辑回归做出的假设要明显更弱一些，所以因此逻辑回归给出的判断也更健壮（robust），也对错误的建模假设不那么敏感。事实上，有很多不同的假设集合都能够将 $p(y|x)$ 引向逻辑回归函数。例如，如果 $x|y=0 \sim Poisson(\lambda_0)$ 、 $x|y=1 \sim Poisson(\lambda_1)$ 都是一个泊松分布，那么 $p(y|x)$ 也将是适合逻辑回归的。逻辑回归也正适用于这类的泊松分布的数据。但对这样的数据，如果我们强行使用高斯判别分析，然后用高斯分布来拟合这些非高斯数据，那么结果的可预测性就会降低，而且GDA这种方法也许可行，也有可能是不能用。

总结一下：很多分布都能将 $p(y|x)$ 引向逻辑回归函数，当数据分布不符合高斯分布时，逻辑回归对不同分布都很健壮，预测结果更加准确。

即：

- 高斯判别分析（GDA）能够建立更强的模型假设，并且在数据利用上更加有效；前提是模型尽可能接近正确。
- 逻辑回归建立的假设更弱，因此对偏离的模型来说更加健壮：如果训练集数据的确是非高斯分布的，而且是有有限的大规模数据，那么逻辑回归几乎总是比GDA要更好的。因此，在实际中，逻辑回归的使用频率要比高斯判别分析（GDA）高得多。

3. 朴素贝叶斯法（Naive Bayes）

在高斯判别分析（GDA）中，特征向量 x 的值是连续的、实数的向量。下面我们要讲的是当 x_i 是离散值的时候来使用的另一种学习方法——朴素贝叶斯法（NB）。

假设我们有一个训练集（也就是一堆已经标好了是否为垃圾邮件的邮件）。要构建垃圾邮件分类器，我们应该先确定用来描述一封邮件的特征 x_i 有哪些。我们将用一个特征向量来表示一封邮件，这个向量的长度等于字典中单词的个数。如果邮件中包含了字典中的第 i 个单词，那么就令 $x_i = 1$ ；反之则 $x_i = 0$ 。例如下面这个向量：

$$x = \begin{bmatrix} 1 & \text{a} \\ 0 & \text{aardvark} \\ 0 & \text{aardwolf} \\ \vdots & \vdots \\ 1 & \text{buy} \\ \vdots & \vdots \\ 0 & \text{zygmurgy} \end{bmatrix}$$

选好了特征向量了，接下来就是建立一个生成模型（generative model），所以我们必须对 $p(x|y)$ 进行建模。要给 $p(x|y)$ 建模，先来做一个非常强的假设。我们假设特征向量 x_i 对于给定的 y 是独立的。这个假设也叫做朴素贝叶斯假设（Naive Bayes，NB assumption），基于此假设衍生的算法也就叫做朴素贝叶斯分类器（Naive Bayes classifier）。然后我们就得到了等式：

$$\begin{aligned} p(x_1, x_2, \dots, x_n | y) &= p(x_1 | y)p(x_2 | y, x_1) \dots p(x_n | y, x_1, x_2, \dots, x_{n-1}) \\ &= p(x_1 | y)p(x_2 | y) \dots p(x_n | y) \\ &= \prod_{i=1}^n p(x_i | y) \end{aligned}$$

值得注意的是，贝叶斯假设是一个很强的假设，产生的这个算法可以适用于很多种问题。我们这个模型的参数为 $\phi_{i|y=1} = p(x_i = 1 | y = 1)$, $\phi_{i|y=0} = p(x_i = 1 | y = 0)$, 而 $\phi_y = p(y = 1)$, 公式表达为：

$$\begin{aligned} p(y) &= \phi_y^y (1 - \phi_y)^{1-y} \\ p(x_i | y = 0) &= \phi_{i|y=0}^{x_i} (1 - \phi_{i|y=0})^{1-x_i} \\ p(x_i | y = 1) &= \phi_{i|y=1}^{x_i} (1 - \phi_{i|y=1})^{1-x_i} \end{aligned}$$

即这三个分布都是伯努利分布， $x_i, y \in \{0, 1\}$ ，可以类比写出来，然后就可以写出下面的联合似然函数：

$$\begin{aligned} L(\phi_y, \phi_{j|y=0}, \phi_{j|y=1}) &= \prod_{i=1}^m p(x^{(i)}, y^{(i)}) \\ &= \prod_{i=1}^m p(x^{(i)} | y^{(i)}; \phi_{j|y=0}, \phi_{j|y=1}) p(y^{(i)}; \phi_y) \end{aligned}$$

找到使联合似然函数取得最大值的对应参数组合 ϕ_y , $\phi_{j|y=0}$ 和 $\phi_{j|y=1}$ 就给出了最大似然估计：

$$\phi_y = \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m}$$

$$\phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{x^{(i)} = 1 \text{ and } y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}}$$

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m 1\{x^{(i)} = 1 \text{ and } y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\}}$$

这些参数有一个非常自然的解释。例如 $\phi_{j|y=1}$ 正是单词 j 出现的邮件中垃圾邮件所占 ($y = 1$) 的比例。拟合好了全部参数后，要对一个新样本的特征向量 x 进行预测，只需要进行如下简单地运算：

$$p(y = 1|x) = \frac{p(x|y = 1)p(y = 1)}{p(x)}$$

$$= \frac{(\prod_{i=1}^n p(x_i|y = 1)) \times p(y = 1)}{(\prod_{i=1}^n p(x_i|y = 1)) \times p(y = 1) + (\prod_{i=1}^n p(x_i|y = 0)) \times p(y = 0)}$$

然后选择有高后验概率的分类。

最后，我们要注意，刚刚我们对朴素贝叶斯算法的使用中，特征向量 x_i 都是二值化的，其实特征向量也可以是多个离散值，比如 $\{1, 2, \dots, k_i\}$ 这样也都是可以的。只需要把 $p(x_i|y)$ 的建模从伯努利分布改成多项式分布。实际上，即使一些原始的输入值是连续值（例如房屋面积），也可以转换成一个小规模的离散值的集合，然后再使用朴素贝叶斯方法。例如我们用特征向量 x_i 来表示住房面积，那么就可以按照下面所示的方法来对这一变量进行离散化：

居住面积	<400	400-800	800-1200	1200-1600	>1600
离散值 x_i	1	2	3	4	5

这样，我们可以根据上面的集合中对应的值得到离散化的特征向量，然后就可以对 $p(x_i|y)$ 用多项式分布来进行建模。然后就给前面讲过的内容一样了（也可以类似 Softmax 处理方式）。当原生的连续值的属性不太容易用一个多元正态分布来进行建模的时候，**将其特征向量离散化然后使用朴素贝叶斯法（NB）来替代高斯判别分析法（GDA）**，通常能形成一个更好的分类器。

3.1 拉普拉斯光滑（Laplace smoothing）

刚刚讲述的朴素贝叶斯的方法能够解决很多问题了，但还能对这种方法进行一点调整来进一步提高效果，尤其是应对文本分类的情况。我们来简要讨论一下一个算法当前状态的一个问题，然后在讲一下如何解决这个问题。

还是考虑垃圾邮件分类的过程，设想你学完了 CS229 的课程，然后做了很棒的研究项目，之后你决定在 2003 年 6 月把自己的作品投稿到 NIPS 会议，这个 NIPS 是机器学习领域的一个顶级会议，递交论文的截止日期一般是六月末到七月初。你通过邮件来对这个会议进行了讨论，然后你也开始受到带有 nips 四个字母的信息。但这个是你第一个 NIPS 论文，而在此之前，你从来没有接到过任何带有 nips 这个单词的邮件；尤其重要的是，nips 这个单词就从来都没有出现在你的垃圾/正常邮件训练集里面。加入这个 nips 是你字典中的第 35000 个单词那么你的朴素贝叶斯垃圾邮件筛选器就要对参数 $\phi_{35000|y}$ 进行最大似然估计，如下所示：

$$\phi_{35000|y=1} = \frac{\sum_{i=1}^m 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 1\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} = 0$$

$$\phi_{35000|y=0} = \frac{\sum_{i=1}^m 1\{x_{35000}^{(i)} = 1 \wedge y^{(i)} = 0\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\}} = 0$$

也就是说，因为之前程序从来没有在别的垃圾邮件或者正常邮件的训练样本中看到过 nips 这个词，所以它就认为看到这个词出现在这两种邮件中的概率都是0.因此当要决定一个包含 nips 这个单词的邮件是否为垃圾邮件的时候，他就检验这个类的后验概率，然后得到了：

$$p(y=1|x) = \frac{\prod_{i=1}^n p(x_i|y=1)p(y=1)}{\prod_{i=1}^n p(x_i|y=1)p(y=1) + \prod_{i=1}^n p(x_i|y=0)p(y=0)}$$

$$= \frac{0}{0}$$

这是因为对于 $\prod_{i=1}^n p(x_i|y)$ 中包含了 $p(x_{35000}|y) = 0$ 的都加了起来，也就还是 0。所以我们的算法得到的就是 $\frac{0}{0}$ ，也就是不知道该做出怎么样的预测了。然后进一步概况这个问题就是，统计学上来说，**只因为你在自己以前的有限的训练数据集中没见过一件事，就估计这个事件的概率为 0，明显是一个坏注意。**例如上面朴素贝叶斯最大似然估计的 ϕ_y 参数公式如下：

$$\phi_y = \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m}$$

如果我们使用这个最大似然估计，假设我们的 m 个独立观测中没有出现该事件，我们单纯的利用公式的得到的 $\phi_y = 0$ ，这就会出现刚才的问题，即 $\frac{0}{0}$ 。为了避免这个问题，我们可以引入**拉普拉斯光滑 (Laplace smoothing)**，即对所有的事件都加上一个 1 避免这个问题，新的估计公式如下：

$$\phi_j = \frac{\sum_{i=1}^m 1\{z(i) = j\} + 1}{m + k}$$

这里首先是对分子加 1，然后对分母加 k (指 y 的 k 个可能的取值， $\sum_{j=1}^k \phi_j = 1$)，这样就能保证对所有的 $\phi_j \neq 0$ ；这样就解决了概率估计为零的问题。

回到我们的朴素贝叶斯分选器问题上，使用了拉普拉斯光滑之后，对参数的估计就可以写成下面这个样子：

$$\phi_{j|y=0} = \frac{\sum_{i=1}^m 1\{x^{(i)} = 1 \text{ and } y^{(i)} = 0\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 0\} + 2}$$

$$\phi_{j|y=1} = \frac{\sum_{i=1}^m 1\{x^{(i)} = 1 \text{ and } y^{(i)} = 1\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 1\} + 2}$$

这里的 2 指的是 x_i 的两种可能的取值 $\{0, 1\}$ ，如果 x_i 有 l 种可能的取值，则将 2 替换成 l 。

(在实际应用中，通常是否对 ϕ_y 使用拉普拉斯并没有太大影响，因为通常我们会对每个垃圾邮件和非垃圾邮件都有一个合适的划分比例，所以 ϕ_y 会是 $p(y=1)$ 的一个合理估计，即我们选取的样本满足 $\phi_y \neq 0$ 。)

3.2 针对文本分类的事件模型 (Event models for text classification)

到这里就要给咱们关于生成学习算法的讨论进行一下收尾了，所以就接着讲一点关于文本分类方面的另一个模型。我们刚已经演示过的朴素贝叶斯方法能够解决很多分类问题了，不过还有另一个相关的算法，在针对文本的分类效果还要更好。

我们上面讲的朴素贝叶斯方法使用的是**一种叫做多元伯努利事件模型 (Multi-Variate Bernoulli event model)** 的方法。这个模型里面，我们假设的主体是：每封收到的邮件是否为垃圾邮件都是随机的。那么收件人首先遍历词典，得到一个表（假设这个表长度为 n ）；然后对收到的每一封邮件查询是否包含单词 i ，各个单词之间独立，且服从概率分布 $p(x_i = 1|y) = \phi_{i|y} (i \in \{1, 2, \dots, n\})$ 。因此，一封邮件的联合概率可以写为：

$$p(x, y) = \left(\prod_{i=1}^n p(x_i|y) \right) p(y)$$

然后还有另外一个模型。叫做**多项式事件模型 (Multinomial event model)**。要描述这个模型，我们需要使用一个不同的记号和特征集来表征各种邮件。首先我们有一个词汇表，其长度为 $|V|$ 。然后，对于收到的每封邮件，用 x_i 表示邮件中的第 i 个单词在词汇表中的位置（即 $x_i \in \{1, 2, \dots, |V|\}$ ）；用 n 表示单词总数，这样我们可以用一个长度为 n 的向量 (x_1, x_2, \dots, x_n) 表征该邮件。例如，如果一个邮件的开头是 “A NIPS …”，那么 $x_1 = 1$ (“a” 是词典中的第一个），而 $x_2 = 35000$ （这是假设 “nips” 是词典中的第 35000 个）。这里要注意，对于不同的邮件内容， n 的取值可以不同。

在多项式事件模型中，我们的参数假设还是和以前一样，即 $\phi_y = p(y)$ 、 $\phi_{k|y=0} = p(x_j = k|y = 0)$ 以及 $\phi_{k|y=1} = p(x_j = k|y = 1)$ 。表面上看这和之前的多元伯努利事件模型的概率很相似，但实际上意义完全不同，这里的 $p(x_j = k|y)$ 是一个**多项式分布**，而不是伯努利分布了。其中 k 的意义为： x_j 处的单词在词汇表中的位置 $k \in \{1, 2, \dots, |V|\}$ 。

如果给定一个训练集 $\{(x^{(i)}, y^{(i)}); i = 1, 2, \dots, m\}$ ，其中 $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \dots, x_{n_i}^{(i)})$ （这里的 n_i 是在第 i 个训练样本中的单词总数），那么这个数据的似然函数如下所示：

$$\begin{aligned} L(\phi, \phi_{k|y=0}, \phi_{k|y=1}) &= \prod_{i=1}^m p(x^{(i)}, y^{(i)}) \\ &= \prod_{i=1}^m \left(\prod_{j=1}^{n_i} p(x_j^{(i)}|y; \phi_{i|y=0}, \phi_{i|y=1}) \right) p(y^{(i)}; \phi_y) \end{aligned}$$

让上面的这个函数最大化就可以产生对参数的最大似然估计：

$$\begin{aligned} \phi_y &= \frac{\sum_{i=1}^m 1\{y^{(i)} = 1\}}{m} \\ \phi_{k|y=0} &= \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{y^{(i)} = 0 \wedge x_j^{(i)} = k\}}{\sum_{i=1}^m 1\{y^{(i)} = 0\} n_i} \\ \phi_{k|y=1} &= \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{y^{(i)} = 1 \wedge x_j^{(i)} = k\}}{\sum_{i=1}^m 1\{y^{(i)} = 1\} n_i} \end{aligned}$$

等式含义解释：

- ϕ_y 的值表示样本中垃圾邮件的百分比；
- $\phi_{k|y=0}$ 的分母：表示样本中正常邮件的单词总量；分子：词汇表中的第 k 个单词在正常邮件中出现的总次数；

- $\phi_k|y=1$ 的分母：表示样本中垃圾邮件的单词总量；分子：词汇表中的第 k 个单词在垃圾邮件中出现的总次数；

如果使用拉普拉斯光滑（实践中会用这个方法来提高性能）来估计 $\phi_{k|y=0}$ 和 $\phi_{k|y=1}$ ，即在分子上加 1，分母上加 k 的可能的取值的个数 $|V|$ ，就得到了下面的等式：

$$\phi_{k|y=0} = \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{y^{(i)} = 0 \wedge x_j^{(i)} = k\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 0\} n_i + |V|}$$

$$\phi_{k|y=1} = \frac{\sum_{i=1}^m \sum_{j=1}^{n_i} 1\{y^{(i)} = 1 \wedge x_j^{(i)} = k\} + 1}{\sum_{i=1}^m 1\{y^{(i)} = 1\} n_i + |V|}$$

当然，这个不一定就是一个最好的分类算法，不过朴素贝叶斯分选器通常用起来都是出乎意料的那么好。所以，这个方法可以作为一个很好的“首发选择”，很简单有很好实现。

1. 支持向量机 (Support Vector Machines)

2. 间隔 (Margins)

2.1 间隔: 直觉 (Margins: Intuition)

2.2 记号 (Notation)

2.3 函数间隔和几何间隔 (Function and geometric margins)

1. 函数间隔

2. 几何间隔

3. 最优间隔分类器 (optimal margin classifier)

3.1 最优间隔分类器问题的形式化

3.2 拉格朗日对偶性 (Lagrange duality)

1. 原始问题 (primal problem)

2. 对偶问题 (dual problem)

3.2 最优边界分类器 (Optimal margin classifiers)

4. 核与 SMO 算法

4.1 核 (Kernel)

1. 特征映射与核

2. 有效核函数

4.2 正则化和不可区分的情况 (Regularization and the non-separable case)

4.3 SMO 算法 (The SMO algorithm)

1. 坐标上升算法 (coordinate ascent algorithm)

2. SMO 算法 (The SMO algorithm)

1. 支持向量机 (Support Vector Machines)

本章的讲义主要讲述的是支持向量机 (**Support Vector Machines, SVM**) 学习算法。SVM 算得上现在最好的现成的监督学习算法之一，很多人实际上认为这里没有“之一”这两个字的必要，认为 SVM 就是最好的现成的监督学习算法。讲这个 SVM 的来龙去脉之前，我们需要先将一些关于间隔的内容，以及对数据进行分割成大的区块 (**gap**) 的思路。接下来，我们要讲一些最优间隔分类器 (**optimal margin classifier**)，其中还会引入一些关于拉格朗日对偶 (**Lagrange duality**) 的内容。然后我们还会接触到核 (**Kernels**)，这提供了一种在非常高的维度（例如无穷维度）中进行 SVM 学习的高效率方法。最终本章结尾部分会讲 **SMO 算法**，也就是 SVM 算法的一个有效实例。

2. 间隔 (Margins)

主要分为三部分：

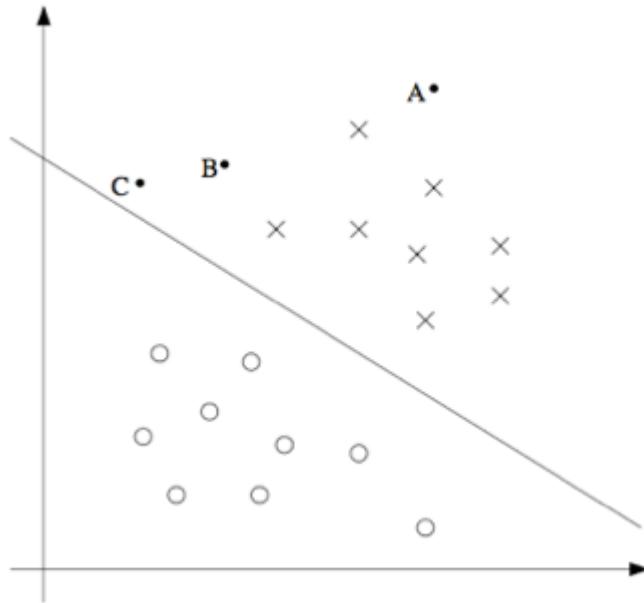
- 间隔: 直觉
- 记号
- 函数间隔和几何间隔

2.1 间隔: 直觉 (Margins: Intuition)

学 SVM 算法，先从间隔 (**Margins**) 开始，本节主要介绍间隔的一些直观展示、以及对于我们做出的预测的“信心”；在第三小节中，会对这些概念进行更正式化的表述：

考虑逻辑回归，其中的概率分布 $p(y=1|x; \theta)$ 是基于 $h_\theta(x) = g(\theta^T x)$ 而建立的模型。当且仅当 $h_\theta(x) \geq 0.5$ ，也就是 $\theta^T x \geq 0$ 的时候，我们才会预测 1；反过来，对于训练样本 $y=1$ ，那么 $\theta^T x$ 越大， $h_\theta(x) = p(y=1|x; \theta)$ 也就越大，我们对预测 Label 为 1 的“信心”也就越强；总结来说，如果 $y=1 \wedge \theta^T x >> 0$ ，我们就对这时候进行的预测非常有信心。同理对于负样本，如果有 $y=0 \wedge \theta^T x << 0$ ，我们也对这时候给出的预测很有信心。因此可以用一种不太正确的方式来说：如果我们找到一个合适的 θ ，满足： $y=1 \rightarrow \theta^T x >> 0$ and $y=0 \rightarrow \theta^T x << 0$ ，我们就是这个对训练数据的拟合很好。为了描述这种思路，稍后我们引入函数间隔符号 (**notion of function margins**)。

此外还有另外一种更直观的表示，如下图所示：x 表示正样本；o 表示负样本；中间直线是分类边界 (**decision boundary**)，也叫分类超平面(**separating hyperplane**)，通过 $\theta^T x = 0$ 确定。



可以发现 A 点距离分界线较远，我们对 A 点进行预测的话，我们很有信心认为该位置 $y=1$ ，反之对于 C 点，这个点距离边界线很近，我们进行预测的话，不是有很大把握。由此，我们可以推出：如果一个点距离分类超平面 (**separating hyperplane**) 比较远，我们就可以对给出的预测很有信心。那么给定一个训练集，如果我们能够找到一个分类边界，利用这个边界我们可以对所有的训练样本给出正确并且有信心（也就是数据点距离分类边界要都很大）的预测，那这就是我们想要达到的状态了。当然上面这种说法还是很不正规，后面我们会使用几何间隔记号 (**notion of geometric margins**) 来更正规地来表达。

2.2 记号 (Notation)

在讨论 SVM 的时候，我们需要引入新的记号，用来表示分类。例如我们针对一个二值化分类的问题建立一个线性分类器，其中用来分类的标签 (**label**) 为 $y \in \{-1, 1\}$ ，分类特征 (feature) 为 x ，线性分类器的参数更改为 \vec{w}, b ，这样分类器可以写为：

$$h_{w,b}(x) = g(w^T x + b)$$

$$g(z) = \begin{cases} 1, & z \geq 0 \\ -1, & z < 0 \end{cases}$$

注意： $w \in R^n, b \in R, x \in R^n$ ，即 b 相当于参数 θ_0 ， w 相当于参数 $[\theta_1, \dots, \theta_n]^T$ 。

2.3 函数间隔和几何间隔 (Function and geometric margins)

1. 函数间隔

给定一个训练集的第 i 个样本 $(x^{(i)}, y^{(i)})$, 我们用下面的方法来定义对应该样本的函数间隔 (w, b) :

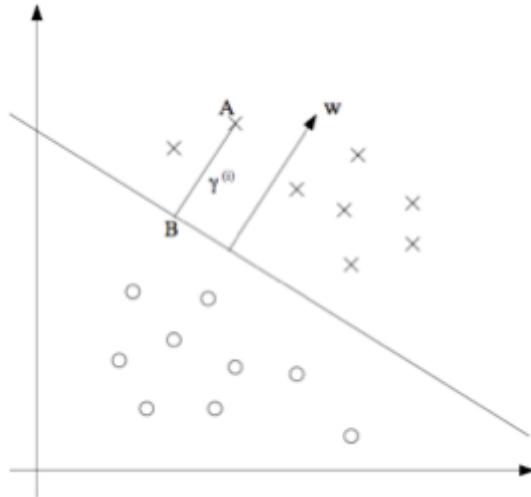
$$\hat{\gamma}^{(i)} = y^{(i)}(w^T x^{(i)} + b)$$

要注意, 如果 $y^{(i)} = 1$, 那么为了让函数间隔很大, 我们就需要 $w^T x + b >> 0$; 如果 $y^{(i)} = -1$, 为了让函数间隔变大, 我们就需要 $w^T x + b << 0$ 。而且, 只要满足 $y^{(i)}(w^T x + b) \geq 0$, 那我们针对这个样本的预测就是正确的。因此, 一个大的函数间隔就表示了一个可信且正确的预测。对于一个训练集 $S = (x^{(i)}, y^{(i)}); i \in \{1, \dots, m\}$, 我们如何衡量整个训练集的函数间隔呢? 我们将对应 S 的函数间隔 (w, b) 定义为每个训练样本的函数间隔的最小值, 记作 $\hat{\gamma}$ (我们的目的是求得合适的 (w, b) , 使得 $\hat{\gamma}$ 最大, 即让最小的样本的函数间隔最大化, 那样整个训练集也会有一个很大的函数间隔)。可以写成:

$$\hat{\gamma} = \min_{i=1, \dots, m} \hat{\gamma}^{(i)} = \min_{i=1, \dots, m} y^{(i)}(w^T x^{(i)} + b)$$

然而, 对于一个线性分类器, 函数间隔的一个性质却使得这个分类器并不具有良好的置信度, 例如成比例的缩放 (w, b) , 对分类结果没有影响, 却可以人为的改变函数间隔。为了解决这个问题, 我们需要引入某种归一化条件, 例如 $\|w\| = 1$ 固定化函数间隔的度量。

2. 几何间隔



图中斜线给出了对应 (w, b) 的分类边界, 其法线方向为向量 w 的方向 (这里的 w 是与分类超平面垂直的)。假设有一个点 A, 此点表示的是训练样本的输入特征 $x^{(i)}$, 对应标签为 $y^{(i)} = 1$ 。假设这个点到分类超平面的距离为 $\gamma^{(i)}$, 也称为几何间隔 (我们先假设其长度, 可以算出来)。如何计算 $\gamma^{(i)}$ 的值呢? 由于 B 点位于分类边界上, 其满足方程 $w^T x + b = 0$, 即我们可以得到方程:

$$w^T(x^{(i)} - \gamma^{(i)} \frac{w}{\|w\|}) + b = 0$$

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

这个解是针对 A 这个正样本的结果, 如果对公式更泛化一下, 就可以定义对应训练样本 $(x^{(i)}, y^{(i)})$ 的几何边界 (w, b) 为:

$$\begin{aligned}\gamma^{(i)} &= y^{(i)} \left(\left(\frac{w}{\|w\|} \right)^T x^{(i)} + \frac{b}{\|w\|} \right) \\ &= y^{(i)} \left(\frac{w^T x^{(i)}}{\|w\|} + \frac{b}{\|w\|} \right)\end{aligned}$$

这里注意，**几何间隔是不受参数缩放影响的**。且，如果 $\|w\| = 1$ ，那么几何间隔就等于函数间隔。最后，给定一个训练集 $S = (x^{(i)}, y^{(i)}); i \in \{1, \dots, m\}$ ，我们也可以将对应 S 的**几何间隔** (w, b) 定义为每个训练样本的几何间隔的最小值：

$$\gamma = \min_{i=1, \dots, m} \gamma^{(i)}$$

3. 最优间隔分类器 (optimal margin classifier)

主要分为三部分：

- 最优间隔分类器问题的形式化
- 拉格朗日对偶性 (Lagrange duality)
- 最优间隔分类器 (Optimal margin classifier)

3.1 最优间隔分类器问题的形式化

给定一个训练集，根据我们前文的讨论，似乎很自然地第一要务就是要尝试找出一个分类边界，使（几何）间隔能够最大。这样生成的分类器，能够把正向和负向的训练样本分隔开，中间有一个空白区，也就是几何间隔。到目前为止，我们都是假定给定的训练集是**线性可分的** (linearly separable) (即一定可以找到一个超平面来进行划分)。那么如何找到使几何间隔最大的那组参数呢？我们提出下面的这样一个优化问题 (optimization problem)：

$$\begin{aligned}& \max_{\gamma, w, b} \gamma \\& \text{s.t. } y(i)(w^T x^{(i)} + b) \geq \gamma, \quad i = 1, \dots, m; \|w\| = 1\end{aligned}$$

也就是说，我们要让 γ 取最大值，必须满足每一个训练样本的函数间隔都至少为 γ ，并且满足 $\|w\| = 1$ 这个约束条件（保证函数间隔与几何间隔相等）。因此，对上面这个优化问题进行求解，就能得出对应训练集合的最大可能几何间隔的 (w, b) 。

然而， $\|w\| = 1$ 这个约束条件是**非凸的**，不能简单的通过标准优化软件解决，所以我们要对这个优化问题进行改善，让它更好解，这就引出第二种表示方法：

$$\begin{aligned}& \max_{\gamma, w, b} \frac{\hat{\gamma}}{\|w\|} \\& \text{s.t. } y(i)(w^T x^{(i)} + b) \geq \hat{\gamma}, \quad i = 1, \dots, m\end{aligned}$$

这时候，我们要让 $\hat{\gamma}/\|w\|$ 的取值最大（几何间隔和函数间隔通过这个公式联系： $\gamma = \hat{\gamma}/\|w\|$ ），使得函数间隔都至少为 $\hat{\gamma}$ ，这样我们就能得到我们想要的结果。这样，我们的**约束条件就没有非凸的问题了**，然而我们的**目标函数** $\hat{\gamma}/\|w\|$ 却**变成了非凸的**，依然不好解决。

那我们继续往下调整，记得我们之前说的可以对参数 w, b 进行任意缩放，而不会有任何实质性改变。那我们引入一个缩放约束参数，使得 (w, b) 的函数间隔相对训练集为 1，即：

$$\hat{\gamma} = 1$$

我们总能找到合适的缩放参数，满足上述假设，这样我们的问题就简化成了如下问题：

$$\begin{aligned} \max_{\gamma, w, b} \frac{1}{||w||} &= \min_{\gamma, w, b} \frac{1}{2} ||w||^2 \\ s.t. \quad y(i)(w^T x^{(i)} + b) &\geq 1, \quad i = 1, \dots, m \end{aligned}$$

(对第一行等式不理解的话，对 $1/||w||$ 求导，分母变为 $||w||^2$ ，然后对左边的最大化问题等于对右边的最小化问题) 这样问题就变得容易解决了。上面的问题变成了一个具有**凸二次对象** (**a convex quadratic objective**) 的目标和仅受**线性约束** (**only linear constraints**) 的优化问题。

这样问题基本得到了解决，接下来我们聊一下拉格朗日对偶性。这样就引出我们这个优化问题的**对偶形式** (**dual form**)，这种形式会在我们后续要使用核 (**kernels**) 的过程中扮演重要角色。核 (**kernels**) 可以有效地对极高维度空间中的数据建立最优边界分类器。通过这种对偶形式，我们还能够推出一种非常有效的算法，来解决上面这个优化问题。

3.2 拉格朗日对偶性 (Lagrange duality)

咱们先把 SVMs 以及最大化边界分类器都放到一边，先来谈一下约束优化问题的求解。例如下面这样的一个问题：

$$\begin{aligned} \min_w f(w) \\ s.t. \quad h_i(w) = 0, \quad i = 1, \dots, l \end{aligned}$$

我们可以使用**拉格朗日乘数法** (**method of Lagrange multipliers**) 来解决。首先定义一个拉格朗日函数为：

$$L(w, b) = f(w) + \sum_{i=1}^l \beta_i h_i(w)$$

上面这个等式中，这个 β_i 就叫做**拉格朗日乘数** (**Lagrange multipliers**)。接下来求偏导数，使其为零：

$$\frac{\partial L}{\partial w_i} = 0; \quad \frac{\partial L}{\partial \beta_i} = 0;$$

然后就可以解出对应的 w, β 。在本节中，我们对此进一步泛化，扩展到**约束优化问题**上，其中包含**不等约束**和**等式约束**。下面这个，我们称之为**主最优化问题** (**primal optimization problem**)：

$$\begin{aligned} \min_w f(w) \\ s.t. \quad g_i(w) \leq 0, \quad i = 1, \dots, k \\ h_i(w) = 0, \quad i = 1, \dots, l \end{aligned}$$

要解决上面这样的问题，首先定义**广义拉格朗日函数** (**generalized Lagrangian**)：

$$L(w, \alpha, \beta) = f(w) + \sum_{i=1}^k \alpha_i g_i(w) + \sum_{i=1}^l \beta_i h_i(w)$$

1. 原始问题 (primal problem)

在上式中， α_i, β_i 都是拉格朗日乘数。设有下面这样一个量（**quantity**）：

$$\theta_P(w) = \max_{\alpha, \beta; \alpha_i \geq 0} L(w, \alpha, \beta) \quad P: \text{primal}$$

这里， P 是 **primal**（原始）的简写。设想我们有一些假定的 w ，如果 w 不能满足某些主要约束（即 *s.t.* 中的条件不满足： $g_i(w) > 0$ ，则 $\theta_P(w) = \infty$, because $\alpha_i \geq 0$; $h_i(w) \neq 0$ ，则 $\theta_P(w) = \infty$, because $\beta_i h_i(w) = \infty$ ）。总结起来就是：

$$\theta_P(w) = \begin{cases} f(w) & \text{如果 } w \text{ 满足约束条件} \\ \infty & \text{otherwise} \end{cases}$$

因此，如果 w 的所有值都满足主要约束条件，那么 θ_p 的值就等于此优化目标的目标量，所以我们可以进一步引出下面这个最小化问题：

$$\min_w f(w) = \min_w \theta_p(w) = \min_w \max_{\alpha, \beta; \alpha_i \geq 0} L(w, \alpha, \beta)$$

第一个等号是指我们将一个约束条件优化问题换成了一种新提出的问题，二者有一样的解。为了方便书写，我们这里定义一个目标量的最优值 p^* ，我们把这个值称为原始优化问题的值（**value of the primal problem**）：

$$p^* = \min_w \theta_p(w)$$

2. 对偶问题（dual problem）

接下来，我们定义一个稍微不同的问题，即 $\theta_D(\alpha, \beta)$ ：

$$\theta_D(\alpha, \beta) = \min_w L(w, \alpha, \beta) \quad D: \text{dual}$$

这里， D 是 **dual**（对偶）的简写。这里要注意， θ_P 是对 (α, β) 优化求最大值； θ_D 是对 (w) 优化求最小值。下面我们就给出这个对偶优化问题了：

$$\max_{\alpha, \beta; \alpha_i \geq 0} \theta_D(\alpha, \beta) = \max_{\alpha, \beta; \alpha_i \geq 0} \min_w L(w, \alpha, \beta)$$

这个形式基本与我们之前见到的主要约束问题是一样的了，唯一不同的是 *max* 和 *min* 的顺序不同。事实上，我们也可对这种对偶问题的最优值进行定义，即：

$$d^* = \max_{\alpha, \beta; \alpha_i \geq 0} \theta_D(\alpha, \beta)$$

原始优化问题如何和对偶优化问题联系在一起的呢，通过下面的公式就很容易发现：

$$d^* = \max_{\alpha, \beta; \alpha_i \geq 0} \min_w L(w, \alpha, \beta) \leq \min_w \max_{\alpha, \beta; \alpha_i \geq 0} L(w, \alpha, \beta) = p^*$$

上面的公式恒成立，在某些特殊的情况下，就会有二者相等的情况： $d^* = p^*$ 。这样我们就可以通过解决对偶约束问题来解决原始约束问题。那么二者相等的条件是什么呢？

假设 f 和 g_i 都是凸函数（**convex**）^[3]， h_i 是仿射的（**affine**）^[4]。进一步假设 g_i 是严格可行的（**strictly feasible**）；这就意味着会存在某个 w ，使得对所有的 i 都有 $g_i(w) < 0$ 。

3. 当 f 有一个海森矩阵(Hessian matrix)的时候，那么当且仅当这个海森矩阵(Hessian matrix)是半正定矩阵， f 才是凸的。例如， $f(w) = w^T w$ 就是凸的；类似的，所有的线性(linear) 以及仿射(affine) 函数也都是凸的。

4. 例如，存在 a_i 和 b_i 满足 $h_i(w) = a_i^T w + b_i$ 。仿射（affine）的意思大概就跟线性（linear）差不多，不同的就是在矩阵进行了线性变换的基础上还增加了一个截距项（extra intercept term） b_i 。

基于上面的假设，可知必然存在 w^* 、 α^* 、 β^* 满足 w^* 为原始问题（primal problem）的解， α^* 、 β^* 是对偶问题（dual problem）的解，此外存在一个 $p^* = d^* = L(w^*, \alpha^*, \beta^*)$ 。另外这三个还会满足 KKT 条件（Karush-Kuhn-Tucker conditions, KKT），如下所示：

$$\frac{\partial}{\partial w_i} L(w^*, \alpha^*, \beta^*) = 0, i = 1, \dots, n \quad (3)$$

$$\frac{\partial}{\partial \beta_i} L(w^*, \alpha^*, \beta^*) = 0, i = 1, \dots, l \quad (4)$$

$$\alpha_i^* g_i(w^*) = 0, i = 1, \dots, k \quad (5)$$

$$g_i(w^*) \leq 0, i = 1, \dots, k \quad (6)$$

$$\alpha_i^* \geq 0, i = 1, \dots, k \quad (7)$$

反过来，如果一组 w^* 、 α^* 、 β^* 满足 KKT 条件，那么这一组解也是原始约束问题和对偶问题的解。

公式 (3) 和 (4) 利用拉格朗日乘数法，即公式 (3) 意义是： $f(w)$ 、 $h_i(w)$ 法向量共线；公式 (4) 意义是：满足约束条件 $h_i(w) = 0$ 。然后我们着重分析公式 (5)，公式 (5) 也叫作 **KKT 对偶互补条件** (dual complementarity condition)。这个公式联合约束条件 (6) (7) 暗示着很多意义，不过先来解释这个公式的形成，如下注释。

1. 如果 $g_i(w) < 0$ ，由于 $\alpha_i \geq 0$ ，那么为了得到最大化的 $\theta_P(w)$ ，我们必须令 $\alpha_i = 0$ ；
2. 如果 $g_i(w) = 0$ （也称为活动条件），那么 α_i 的取值不影响 $\theta_P(w)$ 的结果（我们以后会注意到满足 $g_i(w) = 0$ 的样本非常关键，称为支持向量，这也是支持向量机的由来）。

如果我们将两种结果糅合在一起，那么我们就得到了公式 (5)： $\alpha_i g_i(w) = 0$ 。

公式 (5) 暗示，当 $\alpha_i^* > 0$ 的时候，则有 $g_i(w^*) = 0$ （也就是说， $g_i(w) \leq 0$ 这个约束条件存在的话，则应该是相等关系）。后面的学习中，这个等式很重要，尤其对于表明 SVM 只有少数的支持向量（Support Vectors）；在学习 SMO 算法的时候，还可以用 KKT 对偶互补条件来进行收敛性检测（convergence test）。

3.2 最优边界分类器（Optimal margin classifiers）

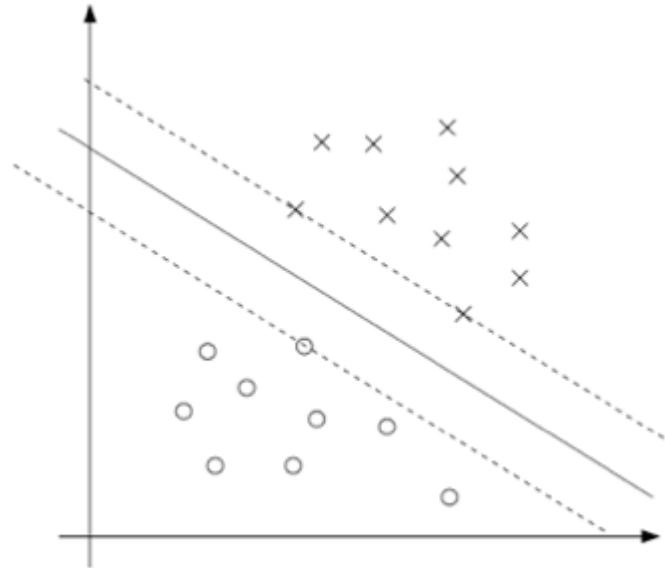
在前面的内容中，我们讲到了寻找最优边界分类器的优化问题形式：

$$\begin{aligned} \max_{\gamma, w, b} \frac{1}{||w||} &= \min_{\gamma, w, b} \frac{1}{2} ||w||^2 \\ s.t. \quad y(i)(w^T x^{(i)} + b) &\geq 1, i = 1, \dots, m \end{aligned}$$

这里的约束条件形式可以写成下面的形式：

$$g_i(w) = -y^{(i)}(w^T x^{(i)} + b) + 1 \leq 0$$

对于训练集中的每一个样本 i ，都要满足上面的约束条件。要注意，通过 **KKT 对偶互补条件** 可知，只有训练样本的函数间隔确定为 1 的情况下，才有 $\alpha_i \geq 0$ （此时这些样本对应的约束条件关系都是等式，即 $g_i(w) = 0$ ）。如下图所示，其中用实线所表示的就是最大间隔分界超平面（maximum margin separating hyperplane）。



具有最小间隔的样本点正好就是距离分类间隔最近的那些点；图中所示，一共有三个这样的点，它们所处的位置即虚线，与分类边界线相平行。因此，在这个优化问题中的最优解里面，只有这三个样本点所对应的 α_i 是非零的（即这些样本是所谓的支持向量）。这种现象就是，**支持向量的规模 (number of support vectors)** 可以比整个训练集的规模 (**size of the training set**) 更小。

那么接下来，我们就要使用拉格朗日对偶性来解决上述的约束优化问题，为了方便书写和表达清晰，我们会以内积的形式给出算法，例如 $\langle x^{(i)}, x^{(i)} \rangle$ 来表示 $(x^{(i)})^T x^{(i)}$ ，即输入特征空间的点相乘得到的内积。当使用核技巧 (**kernel trick**) 的时候，内积形式就显得极为重要。

1. 在构建拉格朗日函数的时候，有下面的等式：

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^m \alpha_i [y^{(i)} (w^T x^{(i)} + b) - 1] \quad (8)$$

这里的拉格朗日乘数只有 α_i ，没有 β_i 。因为问题中只有不等式约束条件。

2. 然后我们写出这个问题的对偶问题：

$$\max_{\alpha; \alpha_i \geq 0} \theta_D(\alpha_i) = \max_{\alpha; \alpha_i \geq 0} \min_{w, b} L(w, b, \alpha)$$

即首先固定 α ，调整 w 、 b ，使 $L(w, b, \alpha)$ 取最小值；然后选择合适的 α ，得到对偶问题的解。

3. 如何使 $L(w, b, \alpha)$ 取最小值呢，即对参数 w 、 b 求偏导数并等于 0，结果如下

$$\begin{aligned} \nabla_w L(w, b, \alpha) &= w - \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} = 0 \\ \nabla_b L(w, b, \alpha) &= - \sum_{i=1}^m \alpha_i y^{(i)} = 0 \end{aligned}$$

整理得：

$$w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \quad (9)$$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0 \quad (10)$$

我们已经得到求 \max 时的 w 、 b 参数了，将公式 (9)、(10) 代入公式 (8) 中，化简得到：

$$\begin{aligned}
L(w, b, \alpha) &= \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} - b \sum_{i=1}^m \alpha_i y^{(i)} + \sum_{i=1}^m \alpha_i \\
&= \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)}
\end{aligned}$$

4. 对偶问题初步总结，我们已经解决了对偶问题的 \min 部分，接下来我们将公式整理得到 \max 部分的对偶优化问题：

$$\begin{aligned}
\max_{\alpha} W(\alpha) &= \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j (x^{(i)})^T x^{(j)} \\
&= \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i=1}^m \sum_{j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\
s.t. \quad &\alpha_i \geq 0, i = 1, \dots, m \\
\sum_{i=1}^m \alpha_i y^{(i)} &= 0, i = 1, \dots, m
\end{aligned}$$

可以证明我们这个优化问题的条件是满足 $p^* = d^*$ 和 KKT 条件的。这样，我们就可以通过求解这个对偶问题来解决掉原来的主优化问题。

具体来说，就是我们构建了一个以 α_i 为参数的取最大值问题，如果能将这个问题解出来（找到 α ，使得 $W(\alpha)$ 取最大值，且满足约束条件），那么我们可以利用公式 (9) 找到最优的 w ，然后再考虑原始优化问题，就能找到 b 的最值了， w, b 参数表达式如下：

$$w = \sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \tag{9}$$

$$b = -\frac{\max_{i:y(i)=-1} w^T x(i) + \min_{i:y(i)=1} w^T x(i)}{2} \tag{11}$$

公式 (11) 的推导过程我个人认为是：我们回到最原始的约束条件 $y(i)(w^T x^{(i)} + b) \geq 1$ 上，此时我们已经找到了合适的 w 了，目的就是求 b ；将 $y^{(i)} = -1$ 和 $y^{(i)} = 1$ 分别带入约束条件中，可以得到如下不等式组：

$$\begin{aligned}
b &\leq -1 - (w^T x^{(i)})_{\max} && (y = -1) \\
b &\geq 1 - (w^T x^{(i)})_{\min} && (y = 1)
\end{aligned}$$

即 b 的取值一定要大于 1 减 $y = 1$ 的最小值了；小于 -1 减 $y = -1$ 的最大值，即：

$$b = -\frac{\max_{i:y(i)=-1} w^T x(i) + \min_{i:y(i)=1} w^T x(i)}{2}$$

好了，让我们看等式 (9) 和 (11)，分析一下代表意义：根据前面的对支持向量的定义，我们可以看出， w 是由支持向量的特征乘以权重构成的（只有支持向量 $\alpha_i \neq 0$ ）；而 b 则是负样本中的最大估计和正样本中的最小估计的均值的相反数的值。我们已知 w, b ，对于新输入的特征 x ，我们可以利用如下公式预测 y ：

$$w^T x + b = \left(\sum_{i=1}^m \alpha_i y^{(i)} x^{(i)} \right)^T x + b \quad (12)$$

$$= \sum_{i=1}^m \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b \quad (13)$$

这样的话，要进行一个预测，如果已经找到了 α_i ，那么预测的值只依赖新的输入特征与训练集中各个样本的内积。另外，我们知道除了支持向量外的 $\alpha_i = 0$ 。那么上面的求和中有很多项都等于 0，这就意味着我们仅需要计算 x 与少量的支持向量的内积即可得到公式 (13)，用于预测。

通过检验优化问题的对偶形式，我们对要解决的问题的结构有了深入的了解，并且还根据输入特征向量之间的内积来编写整个算法。在下一节中，我们将充分利用这些内容，然后对我们的分类问题使用核 (**kernels**) 方法。最终得到的算法，就是支持向量机 (support vector machines) 算法，将能够在非常高的维度空间中有效地学习。

4. 核与 SMO 算法

主要分为三部分：

- Kernel
- Soft margin
- SMO algorithm

4.1 核 (Kernel)

1. 特征映射与核

我们之前讲线性回归的时候，遇到过这样一个问题，其中输入特征 x 是一个房屋的居住面积，然后我们考虑使用特征 x, x^2, x^3 来进行拟合得到一个立方函数。要区分出两组数据集，我们把原始的输入值称为一个问题的输入属性，例如 x ；我们把这些值映射到另外的一些数据集来传递给学习算法的时候，这些新的数据值就成为输入特征。然后我们用 ϕ 来表示特征映射 (**feature mapping**)，即从问题的输入属性到传递给算法的输入特征之间的映射关系。例如上面的例子，写成特征映射就可以表示为：

$$\phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}$$

那么我们就不再简单地利用 SVM 来处理原始数据的输入属性 x 了，而是尝试着利用特征映射产生的新的特征 $\phi(x)$ 。事实上，我们只需要简单回顾之前的算法，然后把所有的 x 替换成 $\phi(x)$ （即对偶问题中求 α 的约束优化问题，只需要把内积形式换成核函数即可）。

由于上面的算法（公式 13）可以用 $\langle x, z \rangle$ 的内积形式给出，这也就意味着只需要把上面的所有内积形式都替换成 $\langle \phi(x), \phi(z) \rangle$ 的内积。更简洁地说，给定一个特征映射 ϕ ，那么就可以定义一个对应的核 (**Kernel**)，如下所示：

$$K(x, z) = \phi(x)^T \phi(z)$$

然后，只需要把上面的算法里用到的 $\langle x, z \rangle$ 全部替换成 $K(x, z)$ ，这样我们的算法就可以使用特征映射 ϕ 来进行机器学习了。

所以，通常的方法是：给定一个特征映射 ϕ ，很容易就可以找出 $\phi(x)$ 和 $\phi(z)$ ，然后取一下内积，就能计算 $K(x, z)$ 了。不过还不仅如此，更有意思的是，通常情况下对这个 $K(x, z)$ 的计算往往都会非常容易，甚至即便 $\phi(x)$ 本身不好计算的时候也如此。在这样的背景下，给定一个特征映射 ϕ ，我们就可以使用 SVM 来对高维特征空间进行机器学习，而可能根本不用麻烦的解出来对应的向量 $\phi(x)$ 。

Kernel 的本质就是给定一个特征映射 ϕ ，用映射后的内积形式（即 $K(x, z) = \phi(x)^T \phi(z)$ ）代替原始的内积形式（即 $\langle x, z \rangle = x^T z$ ）；倒过来看的话，我们可以设计核 (\mathcal{K})，反过来求特征映射 ϕ 。

接下来我们来看一个例子吧，例如 $x, z \in R^n$ ，设我们设计的核形式如下：

$$K(x, z) = (x^T z)^2$$

这个也可以写成下面的形式：

$$\begin{aligned} K(x, z) &= \left(\sum_{i=1}^n x_i z_i \right) \left(\sum_{j=1}^n x_j z_j \right) \\ &= \sum_{i=1}^n \sum_{j=1}^n x_i x_j z_i z_j \\ &= \sum_{i,j=1}^n (x_i x_j)(z_i z_j) \\ &= \phi(x)^T \phi(z) \end{aligned}$$

所以特征映射 ϕ 给出如下所示（例子中 $n = 3$ ）：

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \end{bmatrix}$$

到这里我们就会发现，如果我们构建了一个高维度的 $\phi(x)$ 并进行计算，其复杂度是 $O(n^2)$ 级别的，而计算我们设计的核 $K(x, z)$ 则只需要 $O(n)$ 的事件，也就是与输入属性的维度呈线性相关关系。

与之相关的核也可以设为：

$$\begin{aligned} K(x, z) &= (x^T z + c)^2 \\ &= \sum_{i,j=1}^n (x_i x_j)(z_i z_j) + \sum_{i=1}^n (\sqrt{2c} x_i)(\sqrt{2c} z_i) + c^2 \end{aligned}$$

对应的特征映射为（例子中 $n = 3$ ）：

$$\phi(x) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_1 x_3 \\ x_2 x_1 \\ x_2 x_2 \\ x_2 x_3 \\ x_3 x_1 \\ x_3 x_2 \\ x_3 x_3 \\ \sqrt{2c}x_1 \\ \sqrt{2c}x_2 \\ \sqrt{2c}x_3 \\ c \end{bmatrix}$$

其中，参数 c 控制了 $x_i x_j$ 和 x_i 的相对权重（relative weighting）。

在此基础上进一步扩展，多项式核函数（Kernel） $K(x, z) = (x^T z + c)^d$ ，就对应了一个 $\binom{n+d}{d}$ 维度的特征空间的特征映射（这个空间的维度也可以写成 $\sum_{i=0}^d n^i$ 维）。然而，即使是针对这种 $O(n^d)$ 维度的高维空间，计算 $K(x, z)$ 依然只需要 $O(n)$ 的时间，而且这个高维特征空间中的特征向量也不需要进行具体的表示。

现在，咱们来从另外一个角度来看一下核(Kernel)。凭直觉来看（这种直觉可能有些错误，不过问题不大），如果 $\phi(x)$ 和 $\phi(z)$ 非常接近，那么就可能认为 $K(x, z) = \phi(x)^T \phi(z)$ 很大；如果 $\phi(x)$ 和 $\phi(z)$ 距离很远，比如近似正交，那么就可能认为 $K(x, z) = \phi(x)^T \phi(z)$ 会很小。这样，我们就可以把核 $K(x, z)$ 理解成对 $\phi(x)$ 和 $\phi(z)$ 的近似程度的一种度量手段，也可以说是对 x 和 z 的近似程度的一种度量手段。

有了这种直观认识之后，如果我们尝试某种学习算法，并想建立某个函数 $K(x, z)$ 来衡量 x 和 z 的近似程度。例如，你选择了下面这个函数：

$$K(x, z) = \exp\left(-\frac{\|x - z\|^2}{2\sigma^2}\right)$$

这个函数就对 x 和 z 的近似程度有一个很好的测量，二者相近时函数值接近 1；远离时函数值接近 0。那么我们能不能用这样的一个函数 K 来作为一个 SVM 里面的核（Kernel）呢？在特定的样例里，是可以的（事实上，这个核也叫作高斯核，对应的是一个无穷维的特征映射 ϕ ）。如果我们将这个结论推广一下，那么给定某个函数 K ，我们如何判定这个函数是不是一个有效的核呢，即是否可以找到一个的特征映射，使得 $K(x, z) = \phi(x)^T \phi(z)$ 。

2. 有效核函数

现在暂时假设 K 就是一个有效的核，对应着某种特征映射 ϕ 。然后，我们考虑有 m 个点的有限集合 $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$ （这个集合不一定必须是训练集），然后设一个方形的 $m \times m$ 矩阵 K ，其中 $K_{ij} = K(x^{(i)}, x^{(j)})$ 。这个矩阵就叫做核矩阵（Kernel matrix）。

1. 对称性：如果一个 K 是一个有效的核，那么就有

$$K_{ij} = K(x^{(i)}, x^{(j)}) = \phi(x^{(i)})^T \phi(x^{(j)}) = \phi(x^{(j)})^T \phi(x^{(i)}) = K(x^{(j)}, x^{(i)}) = K_{ji}$$

即 K 是一个对称矩阵。

2. 正半定（也叫半正定）：对于任意向量 \vec{z} ，有：

$$\begin{aligned}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i \phi(x^{(i)})^T \phi(x^{(j)}) z_j \\
&= \sum_i \sum_j z_i \sum_k \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(x^{(i)}) \phi_k(x^{(j)}) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(x^{(i)}) \right)^2 \\
&\geq 0
\end{aligned}$$

这就表明了矩阵 K 是正半定 (positive semidefinite) 的矩阵。

这样，我们就证明了，如果 K 是一个有效核函数 (即该函数对应某种特征映射 ϕ)，那么对应的核矩阵 (Kernel Matrix) $K \in R^{m \times m}$ 就是一个对称正半定矩阵 (symmetric positive semidefinite)。进一步扩展，这个条件不仅仅是 K 是一个有效核函数的必要条件，还是充分条件，这个核函数也叫做默瑟核 (Mercer kernel)。下面要展示的结果都是源自于 Mercer's theorem。

默瑟定理 (Mercer's theorem)：设给定的 $K : R^n \times R^n \rightarrow R$ ，如果 K 为一个有效的默瑟核 (Mercer kernel)，其充要条件为：对任意的 $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\} (m < \infty)$ ，都有对应的核矩阵为对称正半定矩阵。

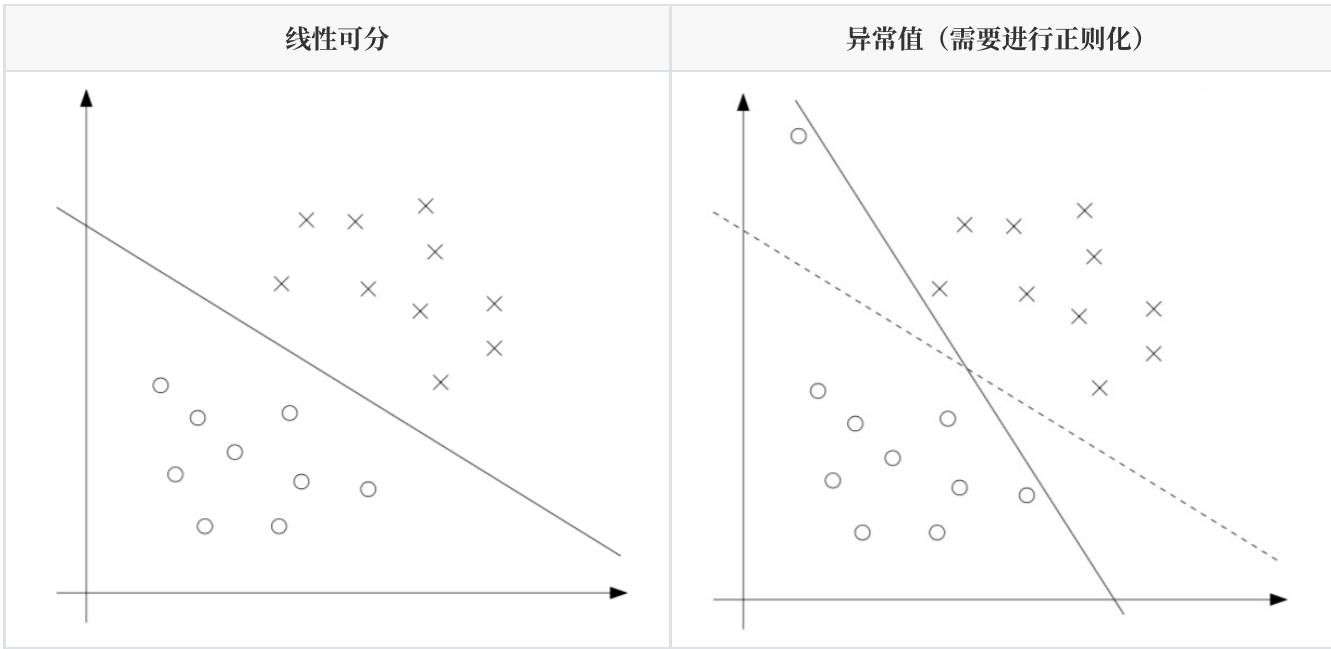
对于一个给定的函数 K ，除了之前的找出对应的特征映射 ϕ 之外，上面的定理还给出了另外一种方法来验证这个函数是否为有效核函数，即判断核矩阵是否为对称正半定矩阵。常用的核函数表示如下：

$K(x, z) = \langle x, z \rangle$	线性核函数
$K(x, z) = (\gamma \langle x, z \rangle + c)^d$	多项式核函数
$K(x, z) = \exp\left(-\frac{\ x - z\ ^2}{2\sigma^2}\right)$	高斯核函数
$K(x, z) = \exp\left(-\frac{\ x - z\ }{\sigma}\right)$	拉普拉斯核函数
$K(x, z) = \tanh(\gamma \langle x, z \rangle + c)$	sigmod 核函数

事实上，核 (Kernel) 的用法远远不仅限于 SVM 算法当中。具体来说，只要你的学习算法，能够写成仅用输入属性向量的内积来表达的形式，那么就可以通过引入核函数 $K(x, z)$ ，来对你的算法“强力”加速，使之能够在与 K 对应的高维度特征空间中有效率地运行。核 (Kernel) 还能与感知器 (Perceptron) 相结合，还可以产生内核感知器算法 (kernel perceptron algorithm)。后文我们要学到的很多算法，也都可以这样处理，这个方法也被称为核技巧 (kernel trick)。

4.2 正则化和不可区分的情况 (Regularization and the non-separable case)

到目前位置，我们对 SVM 进行的推导，都是基于一个假设，也就是所有的数据都是线性可分的。在通过特征映射 ϕ 来将数据映射到高纬度特征空间的过程，通常会增加数据可分割的概率，但是我们不能保证数据一直可分、或者出现异常值（异常值会导致过拟合，需要进行正则化）。例如如下所示的最优边界分类器，如果出现了了一个单独的异常值，这就会导致分界线出现显著的偏移，还会导致分类器的边界缩小。



要想让算法能够使用于非线性可分的数据集，并且使其对异常值的敏感度降低，要对我们的约束优化方法进行重构 (reformulate)，使用 **L1 正则化** (use l_1 regularization)，如下所示：

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

这样就允许数据集里面有（函数）边界小于 1 的情况；然后如果一个样本的函数边界为 $1 - \xi_i (\xi_i \geq 0)$ ，那么 $C\xi_i$ 就是目标函数的成本函数的降低值 (cost of the objective function being increased)。 C 是一个参数，用于在一对目标间控制权重，一个是使得 $\|w\|^2$ 取最小值；另一个是确保绝大部分的样本都有至少为 1 的函数边界。

接下来，我们给出拉格朗日函数：

$$L(w, b, \xi, \alpha, r) = \frac{1}{2} w^T w + C \sum_{i=1}^m \xi_i - \sum_{i=1}^m \alpha_i [y^{(i)}(x^T w + b) - 1 + \xi_i] - \sum_{i=1}^m r_i \xi_i$$

上式中的 α_i 和 r_i 都是拉格朗日乘数（主最优化问题中的 ≤ 0 的约束条件的乘数项），有 $\alpha_i, r_i \geq 0$ 。这次我们不在对偶形式进行详细的推导，过程类似，将上式看作是变量 w, b 的函数，分别对其进行偏导，得到 w, b 的表达式。然后代入公式，求代入后公式的极大值，结果如下（求和的 i, j 是指 i, j 分别变化）：

$$\begin{aligned} \max_{\alpha} \quad & W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)}, x^{(j)} \rangle \\ \text{s.t.} \quad & 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \\ & \sum_{i=1}^m \alpha_i y^{(i)} = 0, \end{aligned}$$

跟以前一样，我们可以用 α_i 表示 w ，和公式 (9) 一样；但是 b 发生了变化，不能用公式 (11) 表示；预测还可以使用公式 (13)。不过有一点不同，在加入了 L1 正则化之后，对偶问题的唯一改变只是约束从原来的 $\alpha_i \geq 0$ 变成了 $0 \leq \alpha_i \leq C$ (这里影响了 b 的取值)。另外，**KKT 对偶互补条件**为：

$$\alpha_i = 0 \implies y^{(i)}(w^T x^{(i)} + b) \geq 1 \quad (14)$$

$$\alpha_i = C \implies y^{(i)}(w^T x^{(i)} + b) \leq 1 \quad (15)$$

$$0 < \alpha_i < C \implies y^{(i)}(w^T x^{(i)} + b) = 1 \quad (16)$$

个人理解：

系数为 0 的样本点，在两条间隔线外或线上（无关向量）；系数为 C 的样本点，在两条间隔线内或线上（离群向量）；系数在 (0, C) 之间的样本点，在两条间隔线上（支持向量）。

4.3 SMO 算法 (The SMO algorithm)

SMO 算法是对序列最小优化算法 (**sequential minimal optimization**) 的缩写，对于从 SVM 推导出的对偶问题，该算法提供了一种有效的解法。不过我们先讲一下坐标上升算法 (**coordinate ascent algorithm**)，这个算法很有趣，而且也是用来推导出 SMO 算法的一步。

1. 坐标上升算法 (coordinate ascent algorithm)

我们首先考虑解决下面这样的无约束优化问题：

$$\max_{\alpha} W(\alpha_1, \alpha_2, \dots, \alpha_m)$$

这里的 W 就是关于参数 α_i 的某种函数，此处暂时忽略掉这个问题和 SVM 的任何关系。之前，我们已经学过了两种优化算法了，梯度法和牛顿法。下面这个新的优化算法，叫做坐标上升算法 (**coordinate ascent**)：

循环直至收敛：{

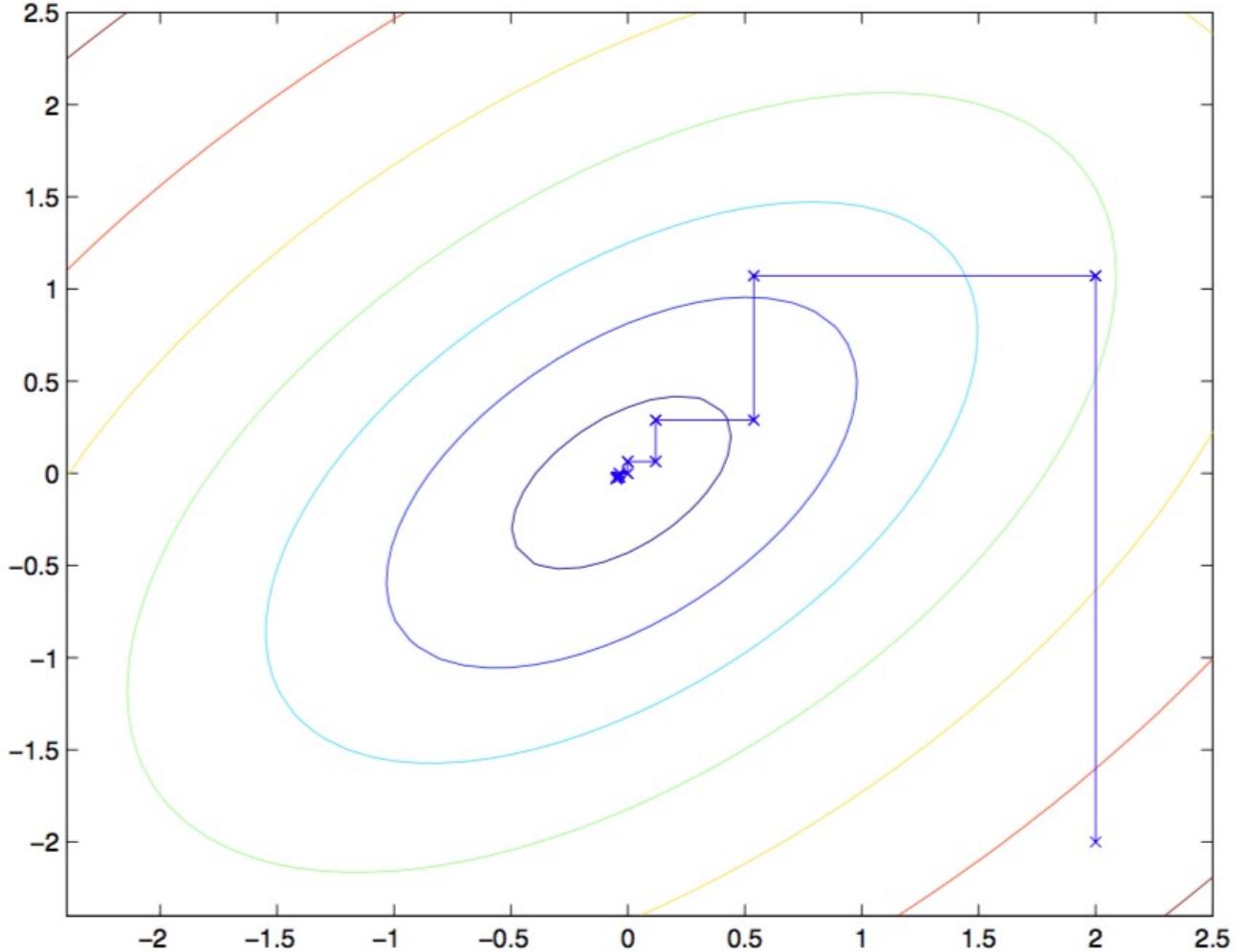
For $i = 1, \dots, m$, {

$$\alpha_i := \arg \max_{\hat{\alpha}_i} W(\alpha_1, \dots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \dots, \alpha_m)$$

}

}

坐标上升算法如上所示，在内层循环中，会对除特定的 α_i 外的变量进行保存，通过优化 $W(\alpha)$ 来调整参数 α_i ，然后剩下的参数进行同样的操作（参数的优化顺序不一定按次序进行）；然后对上述操作进行循环直至参数收敛。下图是一个坐标上升算法的示意图：起始坐标为 $(2, -2)$ ，先对纵轴优化，然后对横轴优化；然后循环操作直至收敛。



2. SMO 算法 (The SMO algorithm)

现在，我们简单地推导一下 SMO 算法。下面是一个对偶优化问题：

$$\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y^{(i)} y^{(j)} \alpha_i \alpha_j \langle x^{(i)} x^{(j)} \rangle \quad (17)$$

$$s.t. \quad 0 \leq \alpha_i \leq C, \quad i = 1, \dots, m \quad (18)$$

$$\sum_{i=1}^m \alpha_i y^{(i)} = 0, \quad (19)$$

考虑到约束条件 (19) 的出现，我们不能简单的利用坐标上升算法固定除 α_i 的参数，重新优化对应 α_i 的目标值，解出 α_i 的新值。根据 (19)，我们可以写出参数 α 之间的依赖关系：

$$\alpha_1 = -\frac{\sum_{i=2}^m \alpha_i y^{(i)}}{y^{(1)}}$$

所以，如果我们对参数 α 进行更新的化，就必须至少同时更新两个，这样才能保证满足约束条件。基于这个情况就衍生出了 SMO 算法，简单来说内容如下所示：

重复直至收敛 {

1. 选择某一对 α_i, α_j 以便在下次迭代中更新（这里需要选择那种能朝向全局最大值方向最大程度靠近的一对值，即 using a heuristic that tries to pick the two that will allow us to make the biggest progress towards the global maximum）；
 2. 保持其他的 $\alpha_k (k \neq i, j)$ 固定，使用对应的 α_i, α_j 来重新优化 $W(\alpha)$ ；
- }

我们可以检查在某些收敛公差参数 tol 范围内，KKT 对偶互补条件是否被满足，以此来检验这个算法的收敛性。这里的 tol 是收敛公差参数，通常都是设定到大概 0.01 到 0.001。

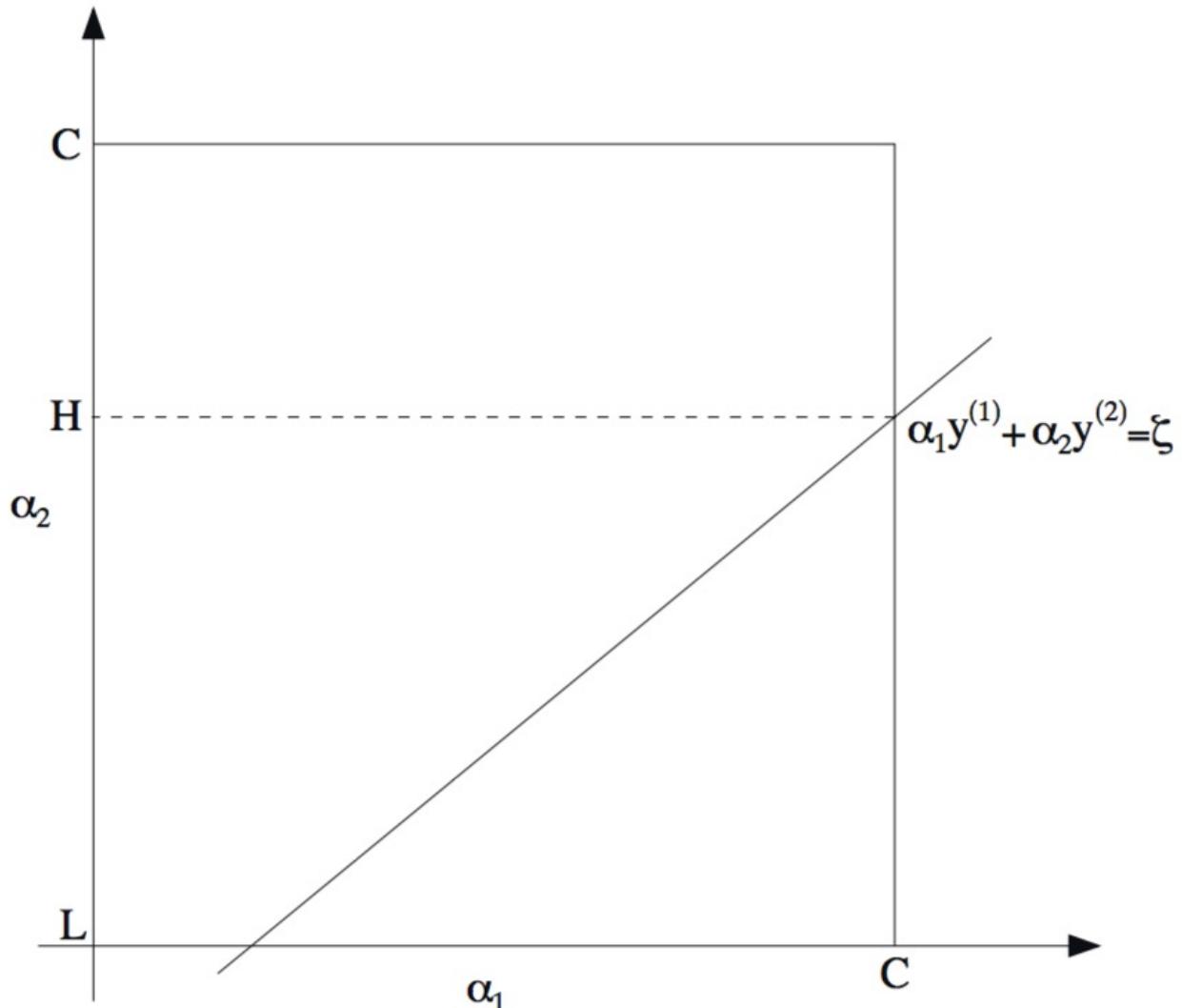
SMO 算法有效的一个关键原因是对 α_i, α_j 的更新计算很有效率。接下来我们简要推导高效率更新的大概思路。假设我们现在有某些 α_i 满足约束条件 (18,19)，如果我们决定要保存固定的 $\alpha_3, \dots, \alpha_m$ 值，然后使用 α_1, α_2 来重新优化 $W(\alpha)$ ，这样成对更新也是为了满足约束条件。根据约束条件 (19)，可以得到：

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = - \sum_{i=3}^m \alpha_i y^{(i)}$$

等号右边是固定的，所以就可把等号右边的项目简写成一个常数 ζ ：

$$\alpha_1 y^{(1)} + \alpha_2 y^{(2)} = \zeta \quad (20)$$

然后根据约束 (18) (方形约束) 和 (20) (线性约束) 可知， α_i, α_j 满足如下关系 (其中 H, L 表示 α_2 在这个约束条件下能取到的极值)：



所以目标函数可以写成：

$$\begin{aligned}\alpha_1 &= (\zeta - \alpha_2 y^{(2)}) y^{(1)} && y = 1 \text{ or } -1 \\ W(\alpha) &= W((\zeta - \alpha_2 y^{(2)}) y^{(1)}, \alpha_2, \dots, \alpha_m)\end{aligned}$$

根据之前的推论，我们可以知道 $W(\alpha)$ 是关于 α_2 的二次函数，利用导数为零求最大值，然后考虑到上图的约束条件，我们可以得到 α_2 的新值：

$$\alpha_2^{new} = \begin{cases} H & \text{if } \alpha_2^{new, unclipped} > H \\ \alpha_2^{new, unclipped} & \text{if } L \leq \alpha_2^{new, unclipped} \leq H \\ L & \text{if } \alpha_2^{new, unclipped} < L \end{cases}$$

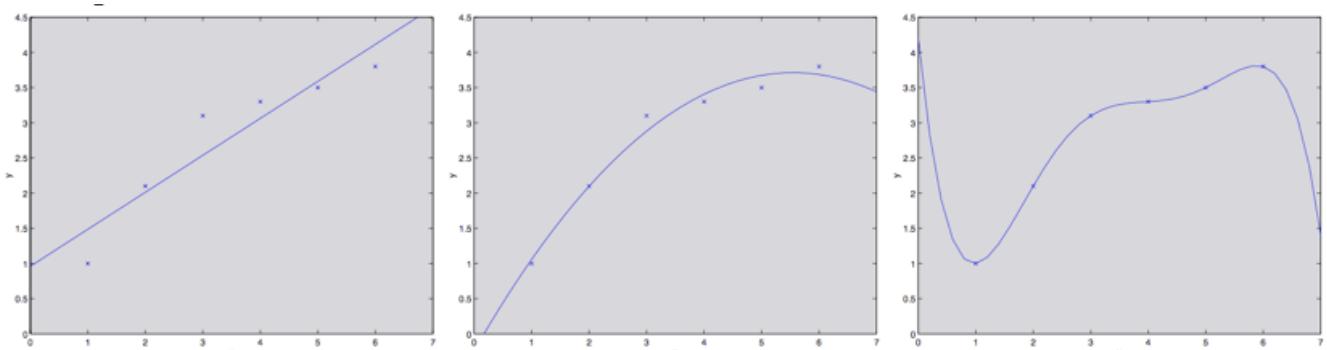
找到 α_2^{new} 之后，就可以利用公式 (20) 回代这个结果，得到 α_1^{new} 。

此外还有一些其他的细节，也都挺简单，不过这里就不讲了，你自己去读一下 John Platt 的论文吧：一个是对后续用于更新的 α_i, α_j 启发式选择；另一个是如何在 SMO 算法运行的同时来对 b 进行更新（ b 依靠 KKT 条件进行更新）。

1. 偏差 / 方差的权衡 (Bias/variance tradeoff)
2. 预先准备 (Preliminaries)
3. 有限个假设的情况 (The case of finite H)
4. 无限个假设的情况 (The case of infinite H)

1. 偏差 / 方差的权衡 (Bias/variance tradeoff)

在讲线性回归的时候，我们讨论过这样的问题：拟合数据的时候，选择如线性 $y = \theta_0 + \theta_1 x$ 的简单模型还是如多项式 $y = \theta_0 + \theta_1 x + \dots + \theta_5 x^5$ 的复杂模型。如下图所示：



如图所示，图中最左边的线性拟合和最右边的高次多项式拟合都有非常大的泛化误差。一个假设 (**hypothesis**) 的泛化误差 (**generalization error**) 是其对训练集中不一定示例的样本的预期误差 (**expected error**)。然而，这两个模型各自出现的问题是不一样的。

- **偏差 (bias)**：描述模型输出结果的期望与样本真实结果的差距；例如左图有很大的偏差，方差很小。
- **方差 (variance)**：描述模型对于给定值的输出稳定性；例如右图有很小的偏差，方差很大。

通常情况下，我们需要在**偏差 (bias)** 和**方差 (variance)** 之间进行权衡妥协。如果我们的模型过于简单，而且参数非常少，那这样可能会有很大的偏差，而方差可能就很小；如果我们的模型过于复杂，有很多的参数，那么可能会有很小的偏差，但方差又特别大。

2. 预先准备 (Preliminaries)

在剩下的内容中，我们开始进入机器学习的**学习理论 (learning theory)** 部分。本章内容非常有趣，而且有启发性，还能帮助我们培养直觉，能够得到在不同背景下如何最佳应用学习算法的经验规则。此外，我们还会探究一些问题：

- 首先，我们刚刚谈到的**偏差 (bias) / 方差 (variance)**，能不能更正规地总结一下？这个问题还会引出关于**模型选择**的方法，这些方法可以在对一个训练集进行拟合的时候来帮助确定要用的多项式应该是几阶的。
- 其次，在机器学习的过程中，我们真正关注的也就是**泛化误差 (generalization error)**，不过绝大部分的学习算法都是将训练集和模型结合的。那么针对训练集的表现好坏程度，为何就能告诉我们泛化误差的信息呢？例如，我们能将训练集的误差和泛化误差联系起来么？
- 最后，是否存在某些条件，我们能否在这些条件下证明某些学习算法能够良好工作？

首先，我们先来给出两个很简单又很有用的引理（lemma）：

- 联合约束（The union bound）：设 A_1, A_2, \dots, A_k 是 k 个不同事件（但不一定互相独立），则有：

$$P(A_1 \cup \dots \cup A_k) \leq P(A_1) + \dots + P(A_k)$$

- Hoeffding 不等式：设 Z_1, Z_2, \dots, Z_m 是 m 个独立的并且遵循伯努利（Bernoulli）分布的随机变量（独立同分布随机变量：independent and identically distributed random variables, iid）。有： $P(Z_i = 1) = \phi$, $P(Z_i = 0) = 1 - \phi$ 。设 $\hat{\phi} = (\frac{1}{m}) \sum_{i=1}^m Z_i$ 是这些随机变量的平均值，然后设任意的 $\gamma \geq 0$ 为某一固定值（fixed），则有：

$$P(|\phi - \hat{\phi}| > q) \leq 2 \exp(-2\gamma^2 m)$$

上面这个引理（也称为切尔诺夫约束，Chernoff bound）表明，如果我们从一个伯努利分布的随机变量中选取平均值 $\hat{\phi}$ 来作为对 ϕ 的估计值，那么只要 m 足够大，我们偏移真实值很远的概率就比较小。

基于上面两个引理，我们就可以开始证明学习理论中的重要结论了。为了简化表述，我们先集中关注一下二分法分类，其中的标签简化为 $y \in \{0, 1\}$ 。然后我们即将讲到的所有内容也都会推广到其它问题中，例如回归问题以及多类别的分类问题等等。

假定我们有一个给定的训练集 $S = \{(x^{(i)}, y^{(i)}); i = 1, \dots, m\}$ ，其样本规模为 m ，集合中的训练样本 $(x_{(i)}, y_{(i)})$ 是符合某个概率分布 D 的独立同分布的随机变量。设一个假设 h ，我们用如下的方法定义训练误差（training error）（在学习理论中也称为经验风险（empirical risk）或经验误差（empirical error））：

$$\hat{\xi}(h) = \frac{1}{m} \sum_{i=1}^m \mathbf{1}\{h(x^{(i)}) \neq y^{(i)}\}$$

这个值只是假设 h 分类错误样本占训练样本总数的比例。如果我们针对某个特定的训练集合 S 的训练误差可以写成 $\hat{\xi}_s(h)$ 。然后我们可以定义泛化误差（generalization error）为：

$$\xi(h) = P_{(x,y) \sim D}(h(x) \neq y)$$

泛化误差意义是：基于分布 D 给出一个新的样本 (x, y) ，假设 h 对该样本分类错误的概率就是泛化误差。

值得注意的是，这里我们有一个预先假设，即训练集和测试集都服从同一个分类 D 。这个假设通常也被认为是 PAC 假设之一。

PAC 是一个缩写，即 probably approximately correct，这是一个框架和一系列假设的集合，在机器学习学习理论中的很多结构都是基于这些假设而证明得到的。这个系列假设中最重要的两个，就是训练样本的独立性和训练集与测试集服从同一分布。

考虑线性分类的情况，假设 $h_\theta(x) = \mathbf{1}\{\theta^T x \geq 0\}$ 。拟合参数 θ 的合理方法是什么呢？一个思路就是最小化训练误差，然后选择最小值时的 θ ：

$$\hat{\theta} = \arg \min_{\theta} \hat{\xi}(h_\theta)$$

我们把上面这个过程称之为经验风险最小化（empirical risk minimization, ERM），这样通过学习算法得到的假设为 $\hat{h} = h_{\hat{\theta}}$ 。我们把 ERM 看作为最基础的学习算法，在这一系列的讲义中我们主要关注的就是这种算法（其他的例如逻辑回归等算法可以看做是对 ERM 的某种近似）。

在我们对学习理论的学习中，有一种做法很有用，就是把假设中的具体的参数抽象出来。我们把学习算法所使用的假设类 H 定义为所有分类器的集合。例如对于线性分类问题， $H = \{h_\theta : h_\theta = \mathbf{1}\{\theta^T x \geq 0\}, \theta \text{ in } R^{n+1}\}$ 是一个对输入特征 x 进行分类的所有分类器的集合。

现在，就可以把经验风险最小化（ERM）看作是对假设类 H 的最小化（利用学习算法选择假设）：

$$\hat{h} = \arg \min_{h \in H} \hat{\xi}(h)$$

3. 有限个假设的情况 (The case of finite H)

我们首先来考虑假设类有限情况下的学习问题，其中假设类 $H = \{h_1, h_2, \dots, h_k\}$ ，由 k 个不同假设组成。因此， H 实际上就是由 k 个从输入特征 x 映射到集合 $\{0, 1\}$ 的函数组成的集合，而经验风险最小化 (ERM) 就是从这样的 k 个函数中选择训练误差最小的作为 \hat{h} 。

我么希望能够保证 \hat{h} 的泛化误差（即泛化误差有界）。这需要两个步骤：首先要表明对所有的 h ，训练误差 $\hat{\xi}(h)$ 是泛化误差 $\xi(h)$ 的一个可靠估计；其次，训练误差 $\hat{\xi}(h)$ 是经验风险最小化 (ERM) \hat{h} 的泛化误差的上界。

训练误差 $\hat{\xi}(h)$ 和 泛化误差 $\xi(h)$ 是相对的：对于假设 h ，训练误差是指在训练集中，假设 h 分类错误的样本占总样本的比例；泛化误差是指基于样本分布 D ，生成的新样本被假设 h 分类错误的概率。

$$\begin{aligned}\hat{\xi}(h) &= \frac{1}{m} \sum_{i=1}^m 1\{h(x^{(i)}) \neq y^{(i)}\} && \text{训练误差} \\ \hat{h} &= \arg \min_{h \in H} \hat{\xi}(h)\end{aligned}$$

$$\begin{aligned}\xi(h) &= P_{(x,y) \sim D}(h(x) \neq y) && \text{泛化误差} \\ h^* &= \arg \min_{h \in H} \xi(h)\end{aligned}$$

这个可以用来解释步骤 (1)，训练误差是泛化误差的一个可靠估计。

经验风险最小化 (ERM) 得到的 \hat{h} 本身就是使得训练误差最小的假设， \hat{h} 的泛化误差即 $\xi(\hat{h})$ 的上界应该与 $\hat{\xi}(\hat{h})$ 有关，而 $\hat{\xi}(\hat{h}) = \min \hat{\xi}(h)$ ，故训练误差 $\xi(h)$ 与 \hat{h} 的泛化误差有关，对应步骤 (2)。

任选一个 $h_i \in H$ 。我们在分布 D 中取 (x, y) ，假如有一个伯努利随机变量 Z ，设 $Z = 1\{h_i(x) \neq y\}$ 。也就是说，我们选择一个样本，并令 Z 表示 h_i 是否进行了错误分类。类似的，我们也定义 $Z_j = 1\{h_i(x^{(j)}) \neq y^{(j)}\}$ 。由于我们的样本都服从分布 D ，则 Z, Z_j 也服从相同的分布。

这样我们就能找到相对假设 h_i ，随机产生的样本分类错误的概率就是 Z 的值，而这个概率本身也代表着泛化误差 $\xi(h_i)$ 。

此外，我们也可以写出训练误差 $\hat{\xi}(h_i)$ 为：

$$\hat{\xi}(h_i) = \frac{1}{m} \sum_{j=1}^m Z_j$$

$\hat{\xi}(h_i)$ 就是 m 个随机变量 Z_j 的平均值，而 Z_j 是服从伯努利分布的独立同分布随机变量，其均值就是 $\xi(h_i)$ 。根据 **Hoeffding 不等式**，得到下面的式子：

$$P(|\xi(h_i) - \hat{\xi}(h_i)| > \gamma) \leq 2 \exp(-2\gamma^2 m)$$

这就表面，对于我们给定的 h_i ，加入训练样本的规模很大的时候，训练误差接近泛化误差的概率还是很高的。但是我们的目标不仅仅满足特定的 h_i ，我们需要证明对所有的 $h \in H$ ，这个结论都成立，以便完成步骤 (1)。为了证明这个结论，我们设 A_i 表示事件 $|\xi(h_i) - \hat{\xi}(h_i)| > \gamma$ ，则有 $P(A_i) \leq 2 \exp(-2\gamma^2 m)$ 。根据联合约束，就可以得出下面的关系：

$$\begin{aligned}
p(\exists h \in H, |\xi(h_i) - \hat{\xi}(h_i)| > \gamma) &= P(A_1 \cup \dots \cup A_k) \\
&\leq \sum_{i=1}^m P(A_i) \\
&\leq \sum_{i=1}^m 2 \exp(-2\gamma^2 m) \\
&= 2k \exp(-2\gamma^2 m)
\end{aligned}$$

如果我们改成否命题，则有：

$$\begin{aligned}
p(\neg \exists h \in H, |\xi(h_i) - \hat{\xi}(h_i)| > \gamma) &= P(\forall h \in H, |\xi(h_i) - \hat{\xi}(h_i)| > \gamma) \\
&\geq 1 - 2k \exp(-2\gamma^2 m)
\end{aligned}$$

如上所示，至少有 $1 - 2k \exp(-2\gamma^2 m)$ 的概率，我们能确保对于所有的 $h \in H$ ，泛化误差 $\xi(h)$ 在训练误差 $\hat{\xi}(h)$ 附近的 γ 范围内，这种结果就叫做一致收敛 (uniform convergence) 结果，因为这是对所有的 $h \in H$ 都成立。

我们不妨假设 $\delta = 2k \exp(-2\gamma^2 m)$ ，那么在上面的讨论中，我们比较感兴趣的变量有三个： m, γ, δ ，我么可以将其中一个变量用另外两个进行约束。

- 例如，我们可以探讨这个问题：给定 $\gamma, \delta \geq 0$ ，如果要保证训练误差在泛化误差附近 γ 范围内的概率不小于 $1 - \delta$ ，那么 m 应该为多大呢？我们可以解出结果为：

$$m \geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta}$$

这种联合约束也说明了至少需要多少数量的训练样本才能对结果有所保证。某些方法或算法为了达到一定的性能水平所需的训练集大小 m 也称为该算法的样本复杂度 (sample complexity)。这个约束条件下的结果有效依赖 k 是一个有限值，这个特性很重要。

- 同理，我们可以假设给定 $m, \delta \geq 0$ ，然后对 γ 求解。即对所有的 $h \in H$ ，都有概率 $1 - \delta$ ，使得训练误差和泛化误差的误差范围在一个范围内：

$$|\xi(h) - \hat{\xi}(h)| \leq \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$$

现在，根据上面的推论，我们可知，对所有的 $h \in H$ ，都有 $|\xi(h) - \hat{\xi}(h)| \leq \gamma$ ，那么对于我们选择的假设 $\hat{h} = \arg \min_{h \in H} \hat{\xi}(h)$ 的泛化误差 $\xi(\hat{h})$ 的范围有什么推论呢（开始进行步骤 (2) 的证明）？

我们假设 $h^* = \arg \min_{h \in H} \xi(h)$ 为 H 中的最佳可能假设 (the best possible hypothesis)。则有：

$$\begin{aligned}
\xi(\hat{h}) &\leq \hat{\xi}(\hat{h}) + \gamma & (1) \\
&\leq \min_{h \in H} \hat{\xi}(h) + \gamma & (2) \\
&\leq \hat{\xi}(h^*) + \gamma & (3) \\
&\leq \xi(h^*) + 2\gamma & (4)
\end{aligned}$$

公式 (1)–(4) 利用的是泛化误差和训练误差的误差范围为 γ ；(2) 用的是 \hat{h} 的定义；(3) 是 $h^* \in H$ 。让我们上面的一大推整理成一个定理：假设 $|H| = k$, m, δ 给定，则在至少 $1 - \delta$ 的概率下，有：

$$\xi(\hat{h}) \leq (\min_{h \in H} \xi(h)) + 2 \sqrt{\frac{1}{2m} \log \frac{2k}{\delta}}$$

这就表明，我们选择的经验风险最小化（ERM）的假设 \hat{h} 的泛化误差 $\xi(\hat{h})$ 最多比全局最优的泛化误差 $\xi(h^*) = \min_{h \in H} \xi(h)$ 多两倍的 γ 。

这也对我们之前提到过的模型选择过程中的在**偏差（bias）** / **方差（variance）**之间的权衡给出了定量方式。例如，我们有原始的假设类 H ，然后考虑切换到某个更大规模的假设类 $H' \supseteq H$ 。如果我们切换到了 H' ，那么第一项的 $\min \xi(h)$ 降低或不变，因此，使用一个更大规模的假设类来进行学习，我们的学习算法的**偏差**只会降低；然而 k 的增大，会导致第二项变大，那么算法的**方差**就会增大。

通过固定 γ, δ ，我们像上面一样求解 m ，还可以得到下面的样本复杂度约束：

$$\begin{aligned} m &\geq \frac{1}{2\gamma^2} \log \frac{2k}{\delta} \\ &= O\left(\frac{1}{\gamma^2} \log \frac{k}{\delta}\right) \end{aligned}$$

4. 无限个假设的情况（The case of infinite H ）

我们已经针对有限个假设类的情况证明了一些有用的定理，然而有很多假设类都包含有无限个函数，那么针对这种无限个假设的情况，我们能给出相似的定理吗？我们先从一些不太准确的论证内容开始，虽然有更好的通用的论证，但这样做有助于锻炼我们在此领域内的直觉（intuitions about the domain）。

若我们有一个假设类 H ，使用 d 个参数来进行参数化。由于我们使用计算机表达实数，而 *double* 型数据使用 64 bit 表示，这就意味着在学习算法中如果使用 *double* 型数据表示参数的话，那么我们的算法就有 64d bit 表示，这样我们的假设类个数 k 最多为 $k = 2^{64d}$ 。结合上一节最好的推论，我们可以发现：在概率至少为 $1 - \delta$ 的条件下，要保证 $\xi(\hat{h}) \leq \xi(h^*) + 2\gamma$ ，则训练样本规模 m 需满足 $m \geq O\left(\frac{1}{\gamma^2} \log \frac{2^{64d}}{\delta}\right) = O\left(\frac{d}{\gamma^2} \log \frac{1}{\delta}\right) = O_{\gamma, \delta}(d)$ 。因此，所需的训练样本的规模与模型中的参数个数最多也是线性的（linear）。

由于上述我们的推论依赖 64 bit 浮点数，所以上面的论证还不能令人完全满意，但是这个结论大致上是正确的：如果我们试图最小化**训练误差（training error）**，为了让具有 d 个参数的假设类的学习效果“较好”，那么我们需要与 d 线性相关的个数的训练样本。（表述不一定正确，意思大概是这样）

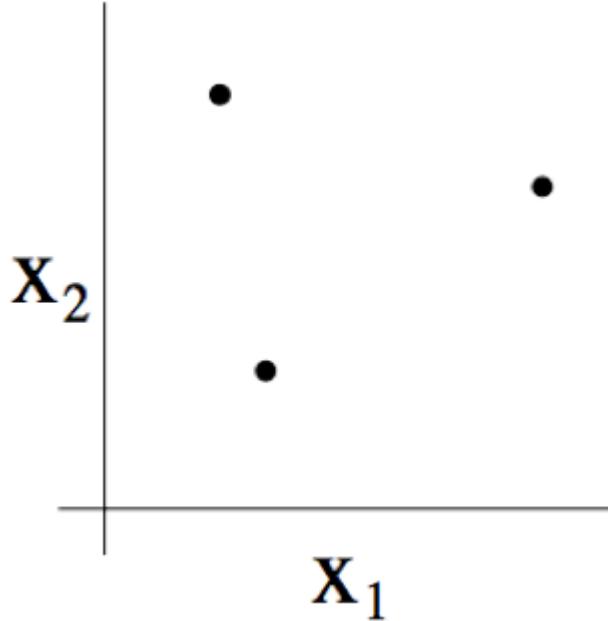
事实上，对于使用经验风险最小化（empirical risk minimization, ERM）的学习算法，上面这些结论已经被证明适用；因此，样本复杂度（sample complexity）对 d 的线性依赖性适用于大多数分类识别学习算法，但训练误差或者训练误差近似值的最小化，就未必适用了。

前面的论证还有另外一部分让分不太满意，就是依赖于对 H 的参数化。直观上看，这个参数化似乎应该不会有太大影响：例如我们定义的线性分类器，写成 $h_\theta(x) = \mathbf{1}\{\theta_0 + \theta_1 x_1 + \dots + \theta_n x_n \geq 0\}$ 有 $n+1$ 个参数；而写成 $h_{u,v}(x) = \mathbf{1}\{(u_0 - v_0) + (u_1 - v_1)x_1 + \dots + (u_n - v_n)x_n \geq 0\}$ 有 $2n+2$ 个参数。然而这两种形式都定义了同样一个 H ：一个 $n+1$ 维的线性分类器集合。为了推导出更准确的结果，我们接下来额外定义一些概念。

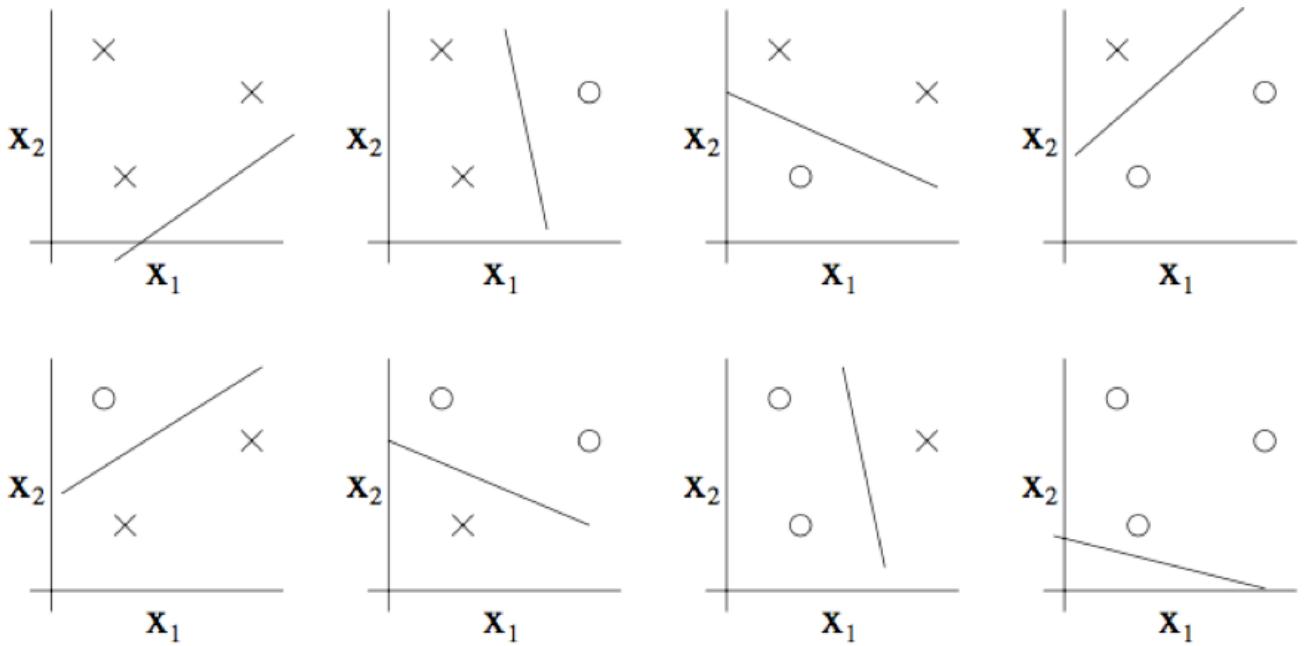
给定一个点的集合 $S = \{x^{(i)}, \dots, x^{(d)}\}$ （与训练集无关），其中 $x^{(i)} \in X$ ，如果 H 能够实现集合 S 的任意标签分布（无论 S 中的 $x^{(i)}$ 的如何选取标签， H 中总存在一个假设 h 生成的分隔超平面能够将 $x^{(i)}$ 按标签分隔开来，即选择的假设 h 满足 $h(x^{(i)}) = y^{(i)}$ 总是正确的（标签只分正、负两种）），则称 H 打散（shatter）了 S 。

给定一个假设类 H ，我们定义它的**VC 维度**（Vapnik-Chervonenkis dimension）为 H 能打散的最大的集合的大小（the size of largest set that shattered by H ），记作 $VC(H)$ 。（如果 H 能打散任意大的集合，则 $VC(H) = \infty$ ）

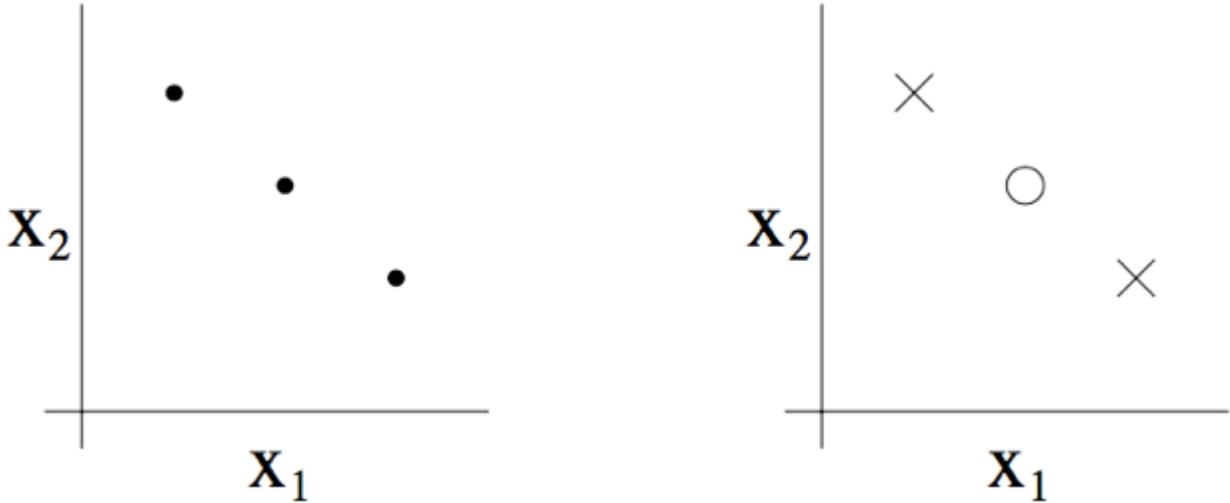
例如，若一个集合由下图所示的三个点组成：



那么二位线性分类器 ($h(x) = 1\{\theta_0 + \theta_1 x_1 + \theta_2 x_2 \geq 0\}$) 的集合 H 能否讲上图所示的这个集合打散呢？答案是能。具体结果如下图所示，对于一下八种情况中的任意一个，我们都能找到一种零训练误差的线性分类器：



这也表明，这个假设类 H 不能打散 4 个点构成的集合。因此，该假设类的 VC 维为 $VC(H) = 3$ 。这里要注意， H 的 VC 维为 3，不代表所有的 3 个点的集合都能被打散。例如如果三个点都在一条直线上，那就没办法能够用线性分类器来对这三个点的类别进行划分了。



通过上面的例子我们可以知道，在 VC 维的定义下，如果 $VC(H)$ 至少为 d ，只需要证明至少有一个规模为 d 的集合能够被 H 打散即可。这样，我们就能给出如下定理 (theorem)：

定理 (Theorem)：给定 H ，设 $d = VC(H)$ ，然后在概率至少为 $1 - \delta$ 的条件下，所有的 $h \in H$ ，存在

$$|\xi(h) - \hat{\xi}(h)| \leq O\left(\sqrt{\frac{d}{m} \log \frac{m}{d}} + \frac{1}{m} \log \frac{1}{\delta}\right)$$

此外，有至少 $1 - \delta$ 的概率，有

$$\xi(h) \leq \xi(h^*) + O\left(\sqrt{\frac{d}{m} \log \frac{m}{d}} + \frac{1}{m} \log \frac{1}{\delta}\right)$$

换句话说，如果一个假设类 H 的 VC 维有限，即 $VC(H) = d < \infty$ ，那么只要训练规模 m 足够大，就能保证联合收敛成立 (uniform convergence occurs)，即第一个等式。和之前一样，这也能够让我们用 $\xi(h^*)$ 来给出 $\xi(h)$ 的界限，即第二个等式表达的意思。

我们还有如下推论 (corollary)：对于所有的 $h \in H$ ，有 $|\xi(h) - \hat{\xi}(h)| \leq \gamma$ 。

换个方式来说，要保证使用假设类 H 的机器学习算法学习效果 well，那么训练集样本规模 m 需要和 H 的 VC 维度线性相关。这也表明，对于 most 假设类 H 来说，VC 维度也大概和参数的个数线性相关。这两条综合一起，我们就能得出这也一个结论：对于一个试图讲训练误差最小化的学习算法来说，训练样本个数 通常都大概与 假设类 H 的参数个数 线性相关。

1. 交叉验证 (Cross Validation)

- 1.1 保留交叉验证 (hold-out cross validation)
- 1.2 k-折叠交叉验证 (k-fold cross validation)
- 1.3 弃一法交叉验证 (leave-one-out cross validation)

2. 特征选择 (Feature Selection)

- 2.1 封装特征选择 (Wrapper feature selection)
- 2.2 过滤特征选择 (Filter feature selection)

3. 贝叶斯统计和正则化 (Bayesian statistics and regularization)

4. 怎样使用机器学习算法

模型选择问题：设想一个机器学习问题，有多种模型可以选择。例如，我们可能是用一个多项式回归模型 $h_\theta(x) = g(\theta_0 + \theta_1 x + \dots + \theta_k x^k)$ ，如何判定这里的多项式次数 k 应该是多少？（即我们怎样才能自动选择一个能够在偏差 / 方差之间进行权衡的模型）或者在另一类参数选择模型中，我们希望能够自动选择一个带宽参数 τ 用于局部加权回归 (locally weighted regression, LWR)；还有就是自动选择一个参数 C 用于 $L1$ 正则化的 SVM 算法中。

形式化定义：假设可选的模型集合是 $M = \{M_1, M_2, \dots, M_d\}$ ，例如在分类问题中，模型可以是逻辑回归算法、神经网络、支持向量机等模型， M 都包含了这些模型；那么模型选择问题就是在这些模型中选择一个最好的。那么首先我们需要使用一种方法定义“最好”，即交叉验证；然后进行选择，选择方式有很多，可以根据交叉验证的结果选择，也可以通过特征选择去筛选模型。

1. 交叉验证 (Cross Validation)

假如我们有一个训练集 S ，如果我们想要通过经验风险最小化 (ERM) 来进行模型选择，那么选择算法如下：

1. 在训练集 S 上，对模型集合中的每个模型 M_i 都进行训练，得到对应的假设 h_i ；
2. 从这些假设中选择训练误差最小的假设 (hypothesis)。

事实上，上面这个算法不可行。比如考虑要选择多项式的阶的问题，多项式的阶越高，对训练集的拟合程度越好，训练误差自然也最小；然而，这种方法选出来的都是偏差小、方差大的高阶多项式，出现过拟合现象。这种方法选出来的通常都是很差的选择。

1.1 保留交叉验证 (hold-out cross validation)

下面这个算法就好一些，这个方法叫保留交叉验证 (hold-out cross validation)，也叫简单交叉验证 (simple cross validation)，步骤如下：

1. 将训练集 S 随机拆分成 S_{train} (用于训练，例如选择整体数据的 70%) 和 S_{cv} (用于验证，剩余的 30%)。这里的 S_{cv} 就叫做保留交叉验证集 (hold-out cross validation set)；
2. 只在集合 S_{train} 上，对每一个模型 M_i 进行训练，然后得到假设 h_i ；
3. 在集合 S_{cv} 上测试每一个 h_i ，得到相应的训练误差 $\hat{\xi}_{S_{cv}}(h_i)$ ；
4. 选择具有最小训练误差 $\hat{\xi}_{S_{cv}}(h_i)$ 的 h_i 作为最佳模型。

这样通过在一部分未进行训练的样本集合 S_{cv} 上进行测试，我们可以认为这里的训练错误 $\hat{\xi}_{S_{cv}}(h_i)$ 接近于泛化误差 (generalization error)。通常可以选择 $1/4 - 1/3$ 的样本用来作为保留交叉验证集，30% 是一个典型选择。

这个方法还有一种备选方法，就是在选择出最佳假设 h_i 后，我们用 h_i 对应的模型 M_i 对整个训练集 S 再次进行训练得到“更好”的假设。（这个思路通常不错，但有一种情况例外，就是学习算法对初始条件和数据的扰动非常敏感的情况；在这样的算法中，适用 S_{train} 的模型未必同样适用于 S_{cv} ，这样最好放弃再训练的步骤。）

不过保留交叉验证方法的一个弊端就是浪费了训练集的一部分，甚至我们使用备用的重新进行训练的步骤也不行。因为我们的步骤（2）相当于试图在 70% 的样本上寻找一个最好的模型，而不是使用全部的样本。在样本充足的情况下，可以使用这种方法。如果样本较少，那最好用其他方法进行模型选择。

1.2 k-折叠交叉验证（k-fold cross validation）

下面我们介绍一种使用较少的验证集的方法：

1. 随机将训练集 S 平均分成 k 的不相交的子集， S_1, \dots, S_k ；

2. 对每个模型 M_i ，我们都安装下面的步骤进行评估：

对 $j = 1, \dots, k$

- 在除 S_j 的其他子集上，对模型 M_i 训练得到假设 h_{ij} ；
- 在 S_j 上，用假设 h_{ij} 进行测试，得到训练误差 $\hat{\xi}_{S_j}(h_{ij})$ 。

对 $\hat{\xi}_{S_j}(h_{ij})$ 取平均值，计算得到的值就当是模型 M_i 的估计泛化误差；

3. 选择具有最小估计泛化误差的模型 M_i ，然后在整个训练集 S 上重新训练该模型，得到的假设就可以作为最优模型。

通常这里进行折叠的次数 k 一般是 10，这样每次用于保留用于验证的训练样本就小多了。不过这样会消耗更多的计算成本，因为对每个模型都要进行 k 次训练。

1.3 弃一法交叉验证（leave-one-out cross validation）

当然我们也可以走一点极端，设 $k = m$ ，这样是为了每次能够尽可能多地利用数据，尽可能少地排除数据。然后其他步骤不变，这样每个模型都要进行 m 次训练。

总的来说，我们介绍的不同版本的交叉验证，不仅可以用来作为模型选择的方法，实际上也可以淡村的对一个具体的模型或算法进行评估。

2. 特征选择（Feature Selection）

模型选择的一个非常重要且特殊的情况就是特征选择（Feature Selection）。假设一个监督学习问题，其中特征值 n 特别大（甚至比训练样本规模还大）；然而你怀疑可能只有一小部分的特征（feature）与学习任务相关。即便使用的是一个简单的 n 维线性分类器，假设类 H 的 VC 维依然也能达到 $O(n)$ ，因此有过拟合的潜在风险，除非样本规模很大。

在这样的背景下，就可以使用特征选择算法，来降低特征值的数量。假设有 n 个特征，那么就有 2^n 个特征子集，如果采用普通的枚举算法对比 2^n 种模型，成本就太高了，所以通常的做法就是使用某些启发式的搜索过程（heuristic search procedure）。

2.1 封装特征选择（Wrapper feature selection）

前向搜索（forward search）：

1. 初始化一个集合为空集 $F = \emptyset$ ；

2. 重复一下过程：

- a. 对 $i = 1, \dots, n$, 如果 $i \notin F$, 则令 $F_i = F \cup \{i\}$, 然后使用某种交叉验证方法来评估特征模型 F_i ;
- b. 在若干个 F_i 中选择最好的一个作为 F 代入 a 中;

3. 整个搜索过程中筛选出来了最佳特征子集，将其输出。

算法的外层循环即步骤 (2) 的停止条件可以是 $F = \{1, \dots, n\}$ 达到全部规模，也可以是 $|F|$ 超过某个预先设置的阈值。

这个算法描述的是对模型特征进行封装（**封装特征选择 (Wrapper feature selection)**）的一个实例，算法本身就是一个将学习算法进行“打包 (wraps)”的过程，然后重复调用这个算法来评估学习算法对不同的特征子集的处理效果。除了前向搜索外，还可以使用其他的搜索特征，例如后向搜索 (backward search)，从 $F = \{1, \dots, n\}$ k开始，然后重复删减特征，只要满足停止条件。

这种封装特征选择算法通常效果不错，不过对算力开销也很大，时间复杂度约为 $O(n^2)$ 。

2.2 过滤特征选择 (Filter feature selection)

过滤特征选择 (Filter feature selection) 方法的原理是针对每一个特征 $x_i, i = 1, \dots, n$, 计算 x_i 相对于类别标签 y 的信息量 $S(i)$, 得到 n 个结果，然后按照从大到小排名，输出前 k 的特征。这样在算力上的开销也更低，大概为 $O(n)$ 。

那么问题的关键是如何定义信息量 $S(i)$, 实际中，通常 (x 为连续值时可以对其进行离散化) 选择 x_i 和 y 的互信息 (mutual information) $MI(x_i, y)$ 作为 $S(i)$ 。

$$MI(x_i, y) = \sum_{x_i \in \{0,1\}} \sum_{y \in \{0,1\}} p(x_i, y) \log \frac{p(x_i, y)}{p(x_i)p(y)}$$

(当 $x_i \in \{0, 1\}$ 的时候，这个公式如上；更一般的情况是变量 x_i 的范围) 这里的概率 $p(x_i, y), p(x_i), p(y)$ 都可以根据训练集上的经验分布来估计。

要对信息量的值有一个更直观的印象，也可以将互信息表达成 **KL 散度** (Kullback-Leibler divergence) :

$$MI(x_i, y) = KL(p(x_i, y) || p(x_i)p(y))$$

这里通俗的表述为，这个概念对 $p(x_i, y), p(x_i)p(y)$ 的概率分布的差异程度给出了一个衡量。如果两个随机变量相互独立，那么必有 $p(x_i, y) = p(x_i)p(y)$ ，那么两个分布之间的 KL 散度就应该是 0。这样符合下面这种很自然的认知：如果 x_i 和 y 相互独立，那么 x_i 很明显对 y 是“无信息量(non-informative)”，因此对应的信息量 $S(i)$ 就应该很小。与之相反，如果 x_i 对 y 有“很大的信息量”，那么这两者的互信息 $MI(x_i, y)$ 就应该很大。

最好一个是：我们如何选择特征个数 k 。一个标准办法就是使用交叉验证来从可能的不同 k 值中进行筛选。

3. 贝叶斯统计和正则化 (Bayesian statistics and regularization)

在本节中，我们要讲另一种工具，用来对抗过拟合 (over fitting)。

在本章的开头，我们谈到了使用最大似然 (maximum likelihood, ML) 来进行参数拟合，然后根据下面的式子来选择合适的参数：

$$\theta_{ML} = \arg \max_{\theta} p(y^{(i)} | x^{(i)}; \theta)$$

在上式中，我们都把 θ 看做一个未知参数（unknown parameter）。频率统计（frequentist statistics）中也把 θ 当成一个未知常量。在频率论世界观中， θ 只是一个未知的、不随机的常量，我们的任务就是提出某种统计程序（例如，最大似然）来对参数进行估计。

另一种解决参数估计问题的方法是使用贝叶斯世界观，把 θ 当成未知的随机变量。这个方法中我们要先指定一个在 θ 上的先验分布（prior distribution） $p(\theta)$ ，这个分布表达了我们关于参数的“预先判断”。给定一个训练集 $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^m$ ，当我们对一个新的 x 的值进行预测的时候，我们可以先计算参数的后验分布（posterior distribution）：

$$\begin{aligned} p(\theta|S) &= \frac{p(S|\theta)p(\theta)}{p(S)} \\ &= \frac{\left[\prod_{i=1}^m p(y^{(i)} | x^{(i)}, \theta) \right] p(\theta)}{\int_{\theta} \left[\prod_{i=1}^m p(y^{(i)} | x^{(i)}, \theta) \right] d\theta} \end{aligned}$$

在上面的等式中， $p(y^{(i)} | x^{(i)}, \theta)$ 来自学习问题中所用的模型。例如，在贝叶斯逻辑回归问题中（Bayesian logistic regression）： $p(y^{(i)} | x^{(i)}, \theta) = h_{\theta}(x^{(i)})^{y^{(i)}} (1 - h_{\theta}(x^{(i)}))^{(1-y^{(i)})}$ ，其中， $h_{\theta}(x^{(i)}) = 1/(1 + \exp(-\theta^T x^{(i)}))$ 。

由于我们这里把 θ 看成一个随机变量，那么就可以使用条件判断，即用 $p(y^{(i)} | x^{(i)}, \theta)$ 替代 $p(y^{(i)} | x^{(i)}; \theta)$ 。

若有一个新的样本 x ，我们可以使用 θ 上的后验分布来计算标签的后验分布：

$$p(y|x, S) = \int_{\theta} p(y|x, \theta)p(\theta|S)d\theta$$

如果目标是根据输入 x 来预测输出 y 的均值，那么可以通过下式计算：

$$E[y|x, S] = \int_y y p(y|x, S) dy$$

上面就是我们简述的过程，可以认为是一种“完全贝叶斯（fully Bayesian）”预测，其中我们的预测是通过计算相对于 θ 上的后验概率 $p(\theta|S)$ 的平均值得出的。实际上，这个后验分布的计算通常比较困难，例如分母上的积分，而 θ 通常是高维度的，这通常不能以闭合形式（closed-form）来实现。

因此实际应用中，我们都是用 θ 的后验分布的近似替代。一个常用的近似是把对 θ 的后验分布替换为一个单点估计（signal point estimate）。 θ 的最大一个后验（maximum a posteriori, MAP）估计为：

$$\theta_{MAP} = \arg \max_{\theta} \prod_{i=1}^m p(y^{(i)} | x^{(i)}) p(\theta)$$

与最大似然 θ_{ML} 相比， θ_{MAP} 在末尾多了一个先验概率分布 $p(\theta)$ 。

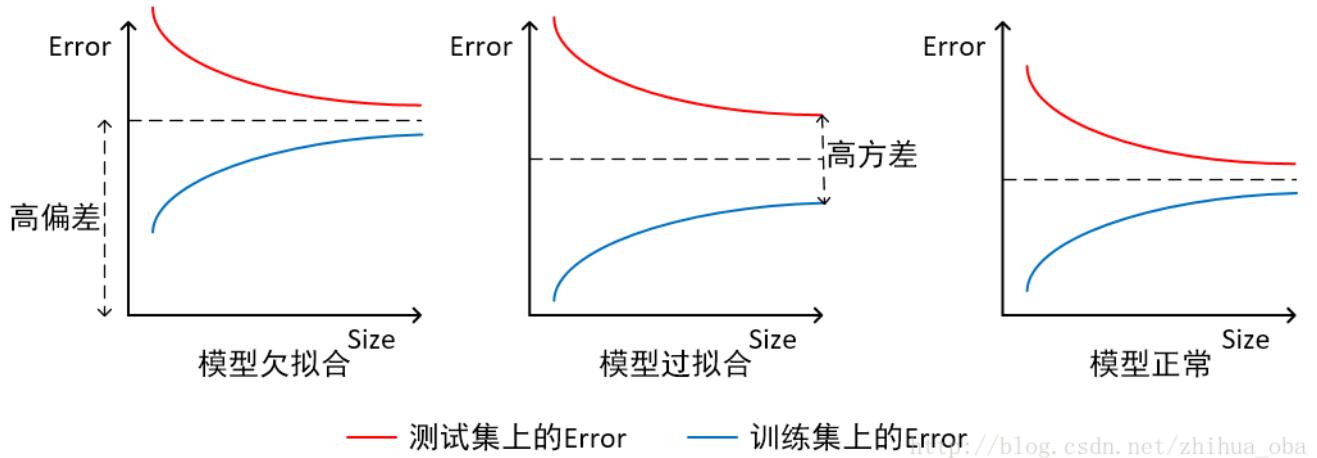
实际应用中，对先验概率分布 $p(\theta)$ 的常见选择是假设 $\theta \sim N(0, \tau^2 I)$ 。使用这样一个先验概率分布，拟合出来的参数 θ_{MAP} 将比最大似然得到的参数 θ_{ML} 有更小的范数（smaller norm）。是实践中，贝叶斯最大后验估计比参数的最大似然估计更容易避免过拟合。

4. 怎样使用机器学习算法

这个是 CS229-11 课上的 PPT 总结，主要介绍的是怎样使用机器学习算法处理问题，主要分为三个部分：

- 学习算法的调试诊断法;
- 误差分析和销蚀分析;
- 如何处理一个机器学习问题
 - 过早优化问题。

针对第一部分的学习算法的调试诊断法，我们可以先判断算法是否欠拟合或过拟合，通过分析学习曲线，提出相应的解决方法：



我们可以发现：模型欠拟合的时候，会出现高偏差问题，可以通过增加特征解决；当模型过拟合的时候，会出现高方差问题，可以通过增大训练集、减少特征解决。

http://blog.csdn.net/zhihua_oba

1. k 均值聚类算法 (k-means clustering algorithm)

2. 高斯混合模型 (Mixtures of Gaussians)

1. k 均值聚类算法 (k-means clustering algorithm)

在聚类问题 (clustering problem) 中，我们得到一组训练集 $\{x^{(1)}, \dots, x^{(m)}\}$ ，然后想要把这些样本划分到若干个集群 (clusters) 中。其中， $x^{(i)} \in R^n$ ，而并未给出分类标签 $y^{(i)}$ ，这就是一个无监督学习问题。

k 均值聚类算法 (看meansclustering algorithm) 如下所示：

1. 随机初始化 k 个聚类重心 $\mu_1, \mu_2, \dots, \mu_k \in R^n$ ；
2. 重复一下过程直至收敛：

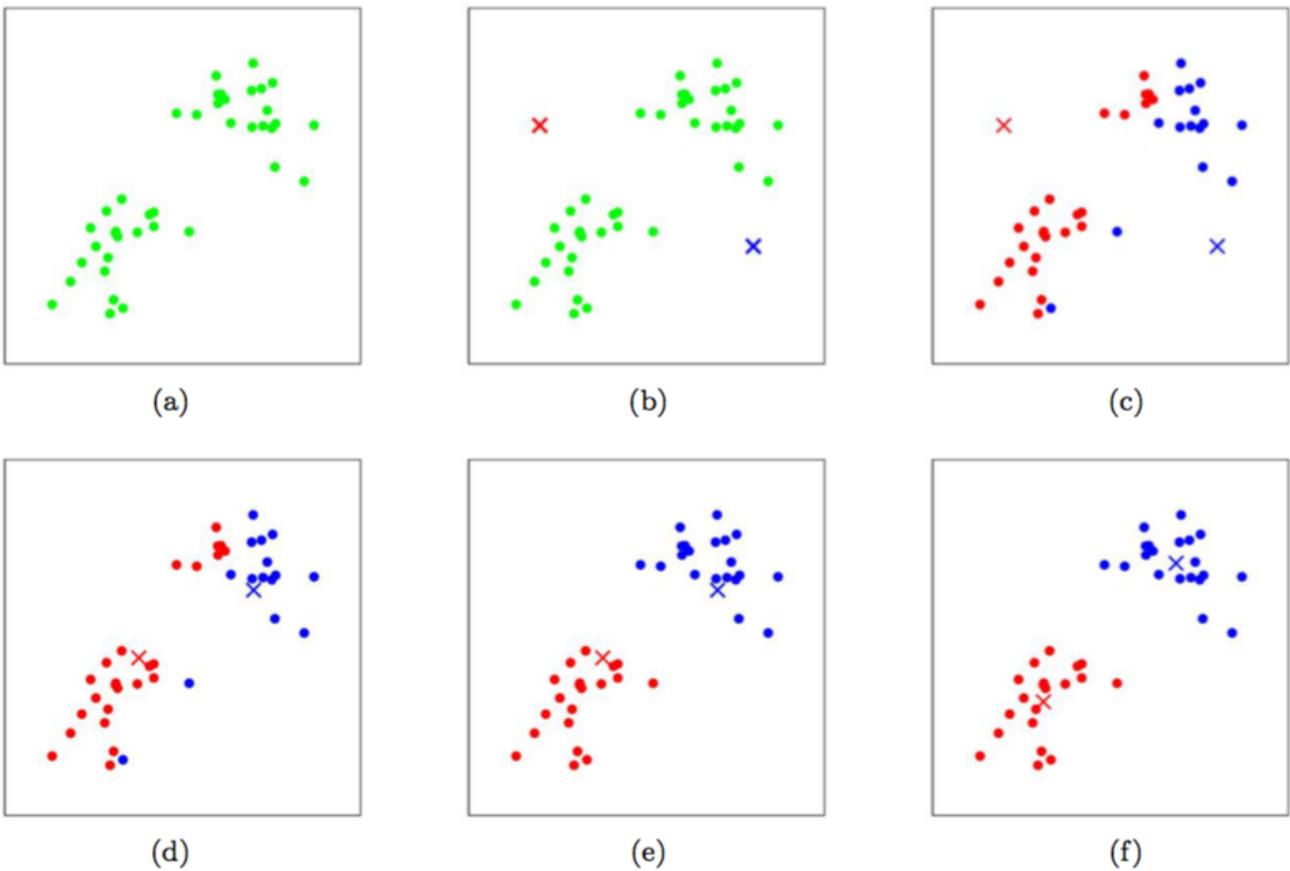
- 对每个 i ，设

$$c^{(i)} = \arg \min_j \|x^{(i)} - \mu_j\|^2$$

- 对每个 j ，设

$$\mu_j = \frac{\sum_{i=1}^m \mathbf{1}\{c^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m \mathbf{1}\{c^{(i)} = j\}}$$

在上面的算法中， k 是我们这个算法的一个参数，也是我们要分出来的群组个数 (number of clusters)；而聚类重心 μ_j 表示的是我们对各个聚类的中心位置的当前猜测。算法第一步中，需要初始化聚类重心，可以这样实现：随机选择 k 个训练样本，然后设置聚类重心等于这 k 个样本各自的值（也有其他的初始化方法）。第二步中，循环体内重复执行两个步骤：1) 将每个训练样本 $x^{(i)}$ “分配” 给距离最近的聚类重心 μ_j ，并用 $c^{(i)}$ 表示其归属的聚类重心；2) 把每个聚类重心 μ_j 移动到所分配的样本点的均值位置。下图就展示了运行 k 均值聚类算法的过程：



k 均值聚类算法在一定意义上是可以保证收敛的。我们可以定义失真函数 (distortion function) 如下所示：

$$J(c, \mu) = \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

这样就可以用 J 来衡量每个样本 $x^{(i)}$ 和对应的聚类重心 $\mu_{c^{(i)}}$ 之间距离的平方和。很明显可以看出 k 均值聚类算法正好就是对 J 的坐标下降过程。尤其是内部的循环体中， k 均值聚类算法重复对 J 进行最小化，当 μ 固定的时候用 c 来最小化 J ；当 c 固定的时候则用 μ 来最小化 J 。这就保证了 J 是单调降低的，它的值也就必然收敛。

失真函数 J 是一个非凸函数 (non-convex function)，所以对 J 进行坐标下降并不一定能够收敛到全局最小值，即可能得到局部最优的结果。除了这个问题外， k 均值聚类效果都不错，能给出很好的聚类。如果担心陷入比较差的局部最小值，可以通过多次运行 k 均值聚类算法（尝试不同的初始值），从中选择最好的。

2. 高斯混合模型 (Mixtures of Gaussians)

在本节中，我们要介绍的是使用期望最大化 (Expectation-Maximization, EM) 来进行密度估计 (density estimation)。和之前一样，假设我们得到了某个训练样本集 $\{x^{(i)}, \dots, x^{(m)}\}$ 。由于这是非监督学习 (unsupervised learning)，所以这些样本就没有标签。

我们希望通过联合分布 $p(x^{(i)}, z^{(i)}) = p(x^{(i)}|z^{(i)})p(z^{(i)})$ 对数据进行建模。其中， $z^{(i)} \sim \text{Multinomial}(\phi)$ (即 $z^{(i)}$ 是一个以 ϕ 多项式分布，其中 $\phi_j = p(z^{(i)} = j)$, $\phi_j \geq 0$, $\sum_{j=1}^k \phi_j = 1$ ，这里的 $z^{(i)} = j$ 表示的是 $x^{(i)}$ 属于集群 j 的概率)；另外， $x^{(i)}|z^{(i)} = j \sim N(\mu_j, \Sigma_j)$ (高斯分布)。我们设 k 来表示 $z^{(i)}$ 能取的值的个数。因此，我们的模型可以简单描述为对于每个样例 $x^{(i)}$ ，我们先从 k 个类别中按多项式分布抽取一个 $z^{(i)}$ ，然后根据 $z^{(i)}$ 所对应的 k 个多值高斯分布中的一个生成 $x^{(i)}$ 。这就叫做高斯混合模型 (Mixtures of Gaussians model)。这里要注意的是 $z^{(i)}$ 是一个隐藏的随机变量。, 我们要对模型中的参数有 ϕ, μ, Σ 进行估计，对似然函数进行对数化如下所示：

$$\begin{aligned}
l(\phi, \mu, \Sigma) &= \sum_{i=1}^m \log p(x^{(i)}; \phi, \mu, \Sigma) \\
&= \sum_{i=1}^m \log \sum_{z^{(i)}=1}^k p(x^{(i)} | z^{(i)}; \mu, \Sigma) p(z^{(i)}; \phi)
\end{aligned}$$

这个式子的最大值是不能通过前面使用的求导数为 0 的方法解决的，因为求的结果不是闭合形式（close form）。假设我们知道每个样本 $x^{(i)}$ 的 $z^{(i)}$ ，那么这个最大似然估计问题可以简化成：

\$\$ l(\phi, \mu, \Sigma) = \sum_{i=1}^m [\log p(x^{(i)} | z^{(i)}; \mu, \Sigma) + \log p(z^{(i)}; \phi)]

对上面的函数最大化，就能得到对应的参数的表达式：

$$\begin{aligned}
\phi_j &= \frac{1}{m} \sum_{i=1}^m 1\{z^{(i)} = j\} \\
\mu_j &= \frac{\sum_{i=1}^m 1\{z^{(i)} = j\} x^{(i)}}{\sum_{i=1}^m 1\{z^{(i)} = j\}} \\
\Sigma_j &= \frac{\sum_{i=1}^m 1\{z^{(i)} = j\} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m 1\{z^{(i)} = j\}}
\end{aligned}$$

事实上，我们已经看到了，如果 $z^{(i)}$ 是已知的（相当于高斯判别分析中提供的标签），那么这个最大似然估计就几乎等同于之前高斯判别分析模型中对参数的估计，即 $z^{(i)}$ 作为高斯判别分析模型中的标签。

然而，在密度估计问题中， $z^{(i)}$ 是不知道的。为了解决这个问题，我们使用期望最大化算法（EM algorithm）。**期望最大化算法（EM algorithm）**是一个迭代算法，有两个主要步骤，在步骤 E 中，算法试图猜测 $z^{(i)}$ 的值；然后在步骤 M 中，这一步我们假设了上一部是对的，然后更新参数，以获得最大似然估计。算法如下：

重复以下过程直到收敛：

- 步骤 E：对每个 i, j ，设

$$w_j^{(i)} := p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

- 步骤 M：更新参数

$$\begin{aligned}
\phi_j &:= \frac{1}{m} \sum_{i=1}^m w_j^{(i)} \\
\mu_j &:= \frac{\sum_{i=1}^m w_j^{(i)} x^{(i)}}{\sum_{i=1}^m w_j^{(i)}} \\
\Sigma_j &:= \frac{\sum_{i=1}^m w_j^{(i)} (x^{(i)} - \mu_j)(x^{(i)} - \mu_j)^T}{\sum_{i=1}^m w_j^{(i)}}
\end{aligned}$$

在步骤 E 中，在给定 $x^{(i)}$ 以及使用当前参数设置的情况下，我们计算出了参数 $z^{(i)}$ 的后验概率。使用贝叶斯规则 (Bayes rule)，我们可以得到：

$$p(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma) = \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)p(z^{(i)} = j; \phi)}{\sum_{l=1}^k p(x^{(i)} | z^{(i)} = l; \mu, \Sigma)p(z^{(i)} = l; \phi)}$$

上面的式子中， $p(x^{(i)} | z^{(i)} = j; \mu, \Sigma)$ 是通过评估一个高斯分布的密度得到的，这个高斯分布的均值为 μ_j ，协方差为 Σ_j ； $p(z^{(i)} = j; \phi)$ 是通过 ϕ_j 得到的。在步骤 E 中计算出来的 $w_j^{(i)}$ 表示我们对 $z^{(i)}$ 这个值的弱估计 (soft guesses)。

另外，我们可以将步骤 M 中进行的更新与 $z^{(i)}$ 已知的形式进行对比。二者的形式是相同的，不同的是之前使用的是指示函数 $1\{z^{(i)} = j\}$ ，表示样本具体服从的高斯分布；而现在用概率 $w_j^{(i)}$ 展示，表示样本服从某高斯分布的概率。

EM 算法也和 k 均值聚类算法相似，只不过 k 均值聚类算法中对样本所属集群 $c^{(i)}$ 进行了强赋值 (hard)；而在 EM 算法中，对 $w_j^{(i)}$ 进行弱赋值 (soft)。与 k 均值聚类算法类似，EM 算法也容易导致局部最优，所以一个好方法就是使用不同的初始参数重新进行初始化。很明显，EM 算法中对 $z^{(i)}$ 进行重复猜测是一个很自然的思路，但是这个算法如何产生，并且是否收敛等性质需要我们进行探讨。我们会在下一章中介绍这些内容。

1. Jensen 不等式 (Jensen's inequality)
2. EM 算法 (The EM algorithm)
3. 高斯混合模型 (Mixture of Gaussians Model)
4. 朴素贝叶斯混合模型 (Mixture of Naive Bayes Model)

在前面高斯混合模型中，我们已经讲过了期望最大化算法对其进行拟合。在本章中，我们要给出期望最大化算法 (EM algorithm) 的更广泛应用，并且演示如何将其应用到一个大系列的具有潜在变量 (latent variables) 的估计问题 (estimation problems)。下面主要介绍 EM 的整个推导过程。

1. Jensen 不等式 (Jensen's inequality)

设 f 为一个函数，其定义域为整个实数域。如果函数 f 的二阶导数 $f''(x) \geq 0 (x \in R)$ ，则函数 f 为一个凸函数 (convex function)。如果输入为向量，那么这个函数就泛化了，这个时候该函数的海森矩阵 (hessian) H 就是一个半正定矩阵。如果对于所有的 x ，都有二阶导数 $f''(x) > 0$ ，我们称这个函数 f 是严格凸函数 (对于输入 x 是向量的情况，对应的条件就是海森矩阵正定)。这样就可以用如下形式来表述 **Jensen 不等式**：

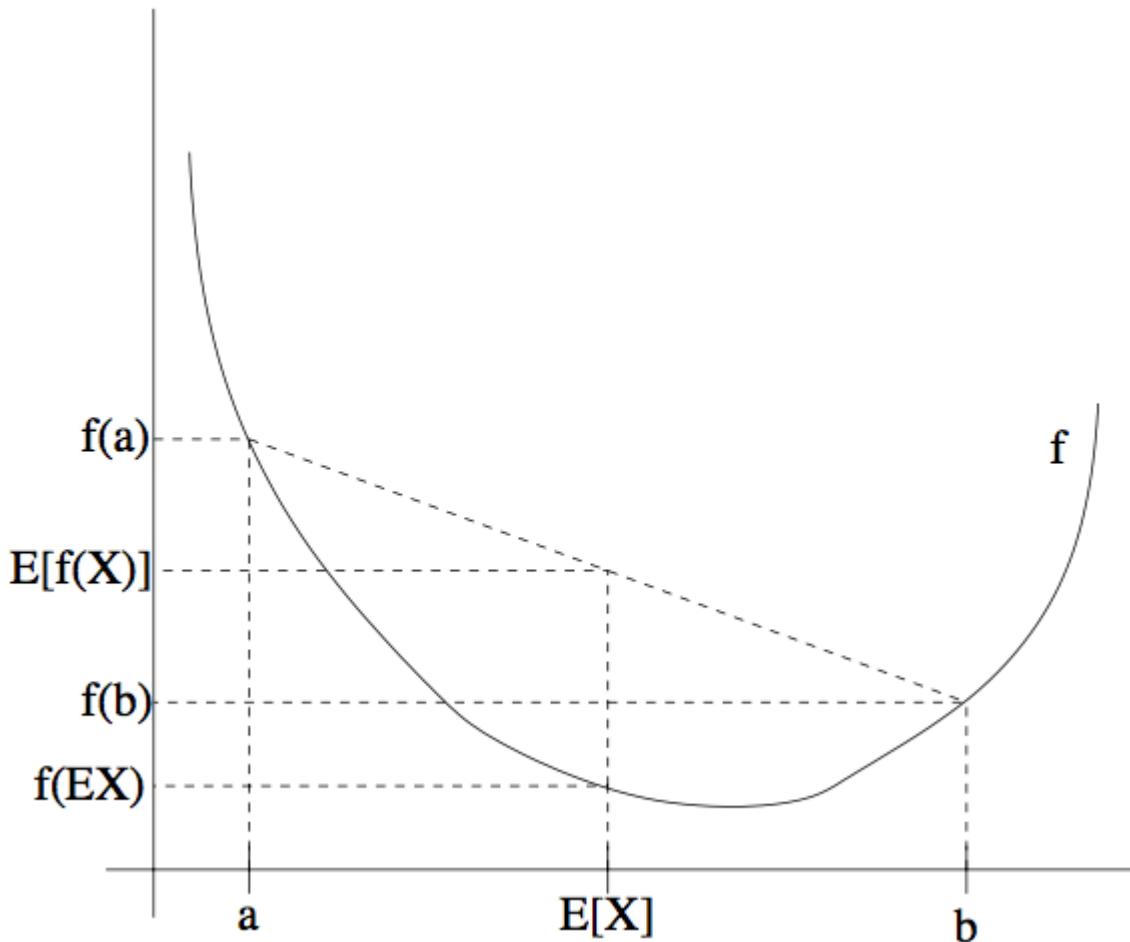
定理 (Theorem)：设 f 是一个凸函数，且设 X 是一个随机变量。然后则有：

$$E[f(X)] \geq f(E[X])$$

(函数值的期望大于等于期望的函数值)

此外，如果函数 f 是严格凸函数，那么 $E[f(X)] = f(E[X])$ 当且仅当 $X = EX$ 的概率为 1，即 X 是常量。

为了方便理解这个定理，可以参考下面的图：



上图中， f 是一个凸函数，在图中用实线表示。另外 X 是一个随机变量，有 0.5 的概率取值为 a ，另外有 0.5 的概率取值为 b 。这样， X 的期望 $E[X]$ 就在图中所示的 a 和 b 的中点位置。图中在 y 轴上也标出了 $f(a)$ 、 $f(b)$ 和 $f(E[X])$ 。函数值的期望 $E[f(X)]$ 在 y 轴上就处于 $f(a)$ 和 $f(b)$ 之间的中点位置。由于 f 是凸函数，很明显 $E[f(X)] \geq f(E[X])$ 。

当且仅当 $-f$ 是严格凸函数的时候， f 是严格凹函数（strictly concave function）。Jensen 不等式也适用于凹函数（concave function），不等式的方向要反过来，即 $E[f(X)] \leq f(E[X])$ 。

2. EM 算法 (The EM algorithm)

假设我们有一个估计问题，其中训练集 $\{x^{(1)}, \dots, x^{(m)}\}$ 包含了 m 个独立样本。我们用模型 $p(x, z)$ 对数据进行建模，拟合其参数，其似然函数 (likelihood) 如下所示：

$$\begin{aligned} l(\theta) &= \sum_{i=1}^m \log p(x_i; \theta) \\ &= \sum_{i=1}^m \log \sum_z p(x_i, z; \theta) \end{aligned}$$

然而，确切地找到对参数 θ 的最大似然估计可能会很难，此处的 $z^{(i)}$ 是一个潜在的随机变量。通常情况下，如果 $z^{(i)}$ 事先得到了，然后在进行最大似然估计，就容易多了。期望最大化算法 (EM algorithm) 是一种解决存在隐含变量优化问题的有效方法。既然不能直接最大化 $l(\theta)$ ，我们可以不断地构建 l 的下界 (步骤 E)；然后对这个下界进行优化 (步骤 M)。

对于每个 i , 设 Q_i 表示该样本隐藏变量 $z^{(i)}$ 的某种分布, 即 $z^{(i)} \sim Q_i, \sum_{z^{(i)}} Q_i(z^{(i)}) = 1, Q_i(z^{(i)}) \geq 0$ 。 (根据上面的高斯混合模型, 可以将 $z^{(i)}$ 可能的取值假设为 $1, 2, \dots, k$ 等 k 个值) 则有下列各式:

$$\sum_i \log p(x^{(i)}; \theta) = \sum_i \log \sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta) \quad (1)$$

$$= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (2)$$

$$\geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (3)$$

上面的等式 (3) 使用了 Jensen 不等式。其中的 $f(x) = \log(x)$ 是一个凹函数, 而且式子

$$\sum_{z^{(i)}} Q_i(z^{(i)}) \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

表示的是变量 $\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$ 基于 $z^{(i)}$ 的期望, 其中 $z^{(i)}$ 是根据 Q_i 给定的分布确定。然后利用 Jensen 不等式可以得到:

$$f\left(E_{z^{(i)} \sim Q_i}\left[\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}\right]\right) \geq E_{z^{(i)} \sim Q_i}\left[f\left(\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}\right)\right]$$

接下来, 对于任意的一个分布 Q_i , 等式 (3) 就给出了似然函数 $l(\theta)$ 的下界。 Q_i 有很多种可能选择, 我们应该选择哪一个 (我们选择合适的 Q_i 的目的是让下界尽可能大, 等号尽可能成立, 这样保证似然函数最大, 然后得到参数 θ) ? 假设我们对参数 θ 有某种估计 (即 θ 假设已知), 很自然的做法就是让下界紧逼当前 θ 值下的似然函数。也就是说, 针对当前的 θ 值, 我们让等号成立 (即先构建下界, 然后对下界优化)。

根据 Jensen 不等式, 要让等式成立, 需要让随机变量变成常数值, 也就是需要:

$$\frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} = c$$

其中常数 c 不依赖 $z^{(i)}$ 。要实现这一条件, 只需满足:

$$Q_i(z^{(i)}) \propto p(x^{(i)}, z^{(i)}; \theta)$$

实际上, 由于我们已知 $\sum_{z^{(i)}} Q_i(z^{(i)}) = 1$, 即 $\sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta) = c$, 这就进一步表明:

$$\begin{aligned} Q_i(z^{(i)}) &= \frac{Q_i(z^{(i)})}{\sum_{z^{(i)}} Q_i(z^{(i)})} \\ &= \frac{p(x^{(i)}, z^{(i)}; \theta)}{\sum_{z^{(i)}} p(x^{(i)}, z^{(i)}; \theta)} \\ &= \frac{p(x^{(i)}, z^{(i)}; \theta)}{p(x^{(i)}; \theta)} \\ &= p(z^{(i)} | x^{(i)}; \theta) \end{aligned}$$

因此, 在给定 $x^{(i)}$ 和参数 θ 的条件下, 我们可以简单的选择 Q_i 为 $z^{(i)}$ 的后验分布 (posterior distribution)。

至此，等式（3）给出了似然函数的一个下界，我们的目的就是最大化似然函数，目的转化为对 Q_i 的选择；然后在参数 θ 假定的情况下， Q_i 的计算公式就是 $z^{(i)}$ 的后验分布；这就是 EM 算法的步骤 E。接下来的步骤 M 中，就是在给定 Q_i 的条件下，最大化等式（3）中的方程，得到新的参数 θ 。重复这两个步骤，就是完整的 EM 算法，总结如下：

重复一下过程直到收敛：{

- 步骤 E：对每个 i ，设：

$$Q_i(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta) \quad (4)$$

- 步骤 M：设

$$\theta := \arg \max_{\theta} \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})} \quad (5)$$

}

我们怎么知道这个算法是否收敛？事实上，假设 $\theta^{(t)}$ 和 $\theta^{(t+1)}$ 是上面 EM 算法迭代过程中的某两个相邻参数，如果存在 $l(\theta^{(t)}) \leq l(\theta^{(t+1)})$ ，这就表明 EM 算法迭代过程总是让似然函数单调递增的。根据上述介绍的 EM 算法，我们的参数起点不妨设为 $\theta^{(t)}$ ，这样我们选择的 Q_i 为：

$$Q_i^{(t)}(z^{(i)}) := p(z^{(i)} | x^{(i)}; \theta^{(t)})$$

在等式（3）的推导过程中，我们选择的 Q_i 可以让不等式的等号成立，因此：

$$l(\theta^{(t)}) = \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{Q_i^{(t)}(z^{(i)})} \quad (6)$$

参数 $\theta^{(t+1)}$ 是通过让上式中等号右侧的部分最大化而得到的，即有：

$$l(\theta^{(t+1)}) \geq \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t+1)})}{Q_i^{(t)}(z^{(i)})} \quad (7)$$

$$\geq \sum_i \sum_{z^{(i)}} Q_i^{(t)}(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta^{(t)})}{Q_i^{(t)}(z^{(i)})} \quad (8)$$

$$\geq l(\theta^{(t)}) \quad (9)$$

不等式（7）成立的条件可以参见不等式（3），不等式（3）对于任意值的 Q_i, θ 都成立。不等式（8）的条件来自于 $\theta^{(t+1)}$ 的选择性，根据不等式（5）， $\theta^{(t+1)}$ 取的是使得似然函数下界最大的参数值，故也成立。不等式（9）的条件来自于等式（6）。因此，EM 算法得到的参数能够保证似然函数的单调收敛。

如果我们定义

$$J(Q, \theta) = \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \theta)}{Q_i(z^{(i)})}$$

通过我们之前的推导，可以知道 $l(\theta) \geq J(Q, \theta)$ 。这样 EM 算法也可看作是在 J 上的坐标上升法，其中步骤 E 在 Q 上对 J 进行了最大化，然后步骤 M 则在 θ 上对 J 进行最大化。

3. 高斯混合模型 (Mixture of Gaussians Model)

有了对 EM 算法的广义定义之后，我们再来看看之前的高斯混合 (MoG) 模型问题。其中要拟合的参数有 ϕ, μ, Σ 。在高斯混合模型中，我们知道一下概率分布：

$$p(z^{(i)} = j) = \phi_j$$
$$p(x^{(i)} | z^{(i)} = j) = \frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)\right)$$

步骤 E 很简单，还是按照上面的算法推导过程，只需要计算：

$$w_j^{(i)} = Q_i(z^{(i)} = j) = P(z^{(i)} = j | x^{(i)}; \phi, \mu, \Sigma)$$

这里面的 $Q_i(z^{(i)} = j)$ 表示的是在分布 Q_i 上 $z^{(i)}$ 取值为 j 的概率。

接下来在步骤 M 中，就是要最大化关于参数 ϕ, μ, Σ 的值：

$$\begin{aligned} & \sum_{i=1}^m \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \phi, \mu, \Sigma)}{Q_i(z^{(i)})} \\ &= \sum_{i=1}^m \sum_{j=1}^k Q_i(z^{(i)} = j) \log \frac{p(x^{(i)} | z^{(i)} = j; \mu, \Sigma) p(z^{(i)} = j; \phi)}{Q_i(z^{(i)} = j)} \\ &= \sum_{i=1}^m \sum_{j=1}^k w_j^{(i)} \log \frac{\frac{1}{(2\pi)^{n/2} |\Sigma_j|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu_j)^T \Sigma_j^{-1} (x^{(i)} - \mu_j)\right) \phi_j}{w_j^{(i)}} \end{aligned}$$

1) 先关于 μ_j 来进行最大化，即先求关于 μ_j 的偏导数，得到：

$$\nabla_{\mu_j} = \sum_{i=1}^m w_j^{(i)} (\Sigma_j^{-1} x^{(i)} - \Sigma_j^{-1} \mu_j)$$

令上式为零，然后解出 μ_j 就得到了更新规则：

$$\mu_j := \frac{\sum_{i=1}^m w_j^{(i)} x^{(i)}}{\sum_{i=1}^m w_j^{(i)}}$$

2) 然后推导 M 步骤中参数 ϕ_j 的更新规则。把仅关于参数 ϕ_j 的表达式结合起来，就能发现只需要最大化下面的表达式：

$$\sum_{i=1}^m \sum_{j=1}^k w_j^{(i)} \log \phi_j$$

然而，此处还有一个附加的约束条件，即 ϕ_j 的和为 1，因为其表示的是概率。为了保证约束条件成立，我们构建一个拉格朗日函数如下：

$$L(\phi) = \sum_{i=1}^m \sum_{j=1}^k w_j^{(i)} \log \phi_j + \beta \left(\sum_{j=1}^k \phi_j - 1 \right)$$

这里我们不用在意约束条件 $\phi_j \geq 0$, 因为很快就能发现, 这里推导得到的解会自然满足这个条件的。

其中的 β 是拉格朗日乘数。求导, 然后得到:

$$\nabla_{\phi_j} L(\phi) = \sum_{i=1}^m \frac{w_j^{(i)}}{\phi_j} + \beta$$

设导数为零, 然后解方程, 就得到了:

$$\begin{aligned}\phi_j &= \frac{\sum_{i=1}^m w_j^{(i)}}{-\beta} \\ \sum_{j=1}^k \phi_j &= \frac{\sum_{j=1}^k \sum_{i=1}^m w_j^{(i)}}{-\beta} \\ &= \frac{\sum_{i=1}^m \sum_{j=1}^k w_j^{(i)}}{-\beta} \\ &= \frac{\sum_{i=1}^m 1}{-\beta} \\ &= \frac{m}{-\beta} = 1\end{aligned}$$

由于 $\sum_{j=1}^k \phi_j = 1$, 可得 $-\beta = m$, 由此可以得到参数 ϕ_j 的更新规则:

$$\phi_j = \frac{1}{m} \sum_{i=1}^m w_j^{(i)}$$

接下来对 M 步骤中对 Σ_j 的更新规则的推导就很容易了。

4. 朴素贝叶斯混合模型 (Mixture of Naive Bayes Model)

根据之前讲的生成式算法朴素贝叶斯法, 在文本分类问题中, 使用朴素贝叶斯混合模型的概率分布如下:

$$\begin{aligned}p(z^{(i)} = 1) &= \phi \\ p(x^{(i)} | z^{(i)}) &= \prod_{j=1}^n p(x_j^{(i)} | z^{(i)}) \\ p(x_j^{(i)} = 1 | z^{(i)} = 1) &= \phi_{j|z=1} \\ p(x_j^{(i)} = 1 | z^{(i)} = 0) &= \phi_{j|z=0}\end{aligned}$$

然后, 我们可以写出 EM 算法的各步骤计算过程:

步骤 E:

$$w^{(i)} = p(z^{(i)} = 1 | x^{(i)}; \phi, \phi_{j|z})$$

步骤 M:

$$\begin{aligned}\phi &= \frac{1}{m} \sum_{i=1}^m w^{(i)} \\ \phi_{j|z=1} &= \frac{\sum_{i=1}^m w^{(i)} \mathbf{1}\{x_j^{(i)} = 1\}}{\sum_{i=1}^m w^{(i)}} \\ \phi_{j|z=0} &= \frac{\sum_{i=1}^m (1 - w^{(i)}) \mathbf{1}\{x_j^{(i)} = 1\}}{\sum_{i=1}^m (1 - w^{(i)})}\end{aligned}$$

其实与生成式算法相似，只不过把 y 换成了 z ，并重复多次直至收敛。

1. Σ 的约束条件**2. 多元高斯模型的边界和条件**

2.1 边界分布

2.2 条件分布

3. 因子分析模型**4. 因子分析模型的期望最大化算法**

当我们有一个来自多个高斯模型的混合的数据集 (a mixture of several Gaussians) $x^{(i)} \in R^n$, 那么就可以用期望最大化算法 (EM algorithm) 来对这个混合模型进行拟合。这种情况下, 对于有充足数据的问题, 我们通常假设可以从数据中识别出多个高斯模型结果。例如, 如果我们的训练样本集合规模 m 远远大于数据的维度 n , 就符合这种情况。然后来考虑一下反过来的情况, 即 $n >> m$ 。在这样的问题中, 可能用单独的一个高斯模型对数据建模都很困难, 更不用说多个高斯模型的混合了。由于 m 个数据点张成 (span) 的空间只是一个 n 维空间的低维子空间。如果用高斯模型进行建模, 然后还是用常规的最大似然估计来计算均值和方差, 得到的则是:

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)(x^{(i)} - \mu)^T$$

我们会发现这里的 Σ 是一个奇异矩阵, 这也就意味着其逆矩阵 Σ^{-1} 不存在。但是在计算多元高斯分布的常规密度是需要这些变量。还有另一种方法来讲述清楚这个难题, 即对参数的最大似然估计会产生一个高斯分布, 其概率分布在由样本数据所张成的仿射空间中, 对应着一个奇异的协方差矩阵。

通常情况下, 除非 m 比 n 大很多, 否则最大似然估计得到的均值和方差都会很差。尽管如此, 我们还是希望能用已有的数据, 拟合出一个合理的高斯模型, 而且还希望能够识别出数据中的某些有意义的协方差结构 (covariance)。在接下来的内容中, 我们首先回顾一个对 Σ 的两个可能的约束, 这两个约束条件能让我们使用小规模数据来拟合 Σ , 但都不能就我们的问题给出让人满意的解。接下来我们要讨论一下高斯模型的一些特点, 这些后面会用得上, 具体来说就是如何找到高斯模型的边界和条件分布。最好, 我们会讲一下因子分析模型 (factor analysis model), 以及对应的期望最大化算法 (EM algorithm)。

1. Σ 的约束条件

如果我们没有充足的数据来拟合一个完整的协方差矩阵, 可以对矩阵空间 Σ 给出某些约束条件。例如, 我们可以选择去拟合一个对角的协方差矩阵 Σ 。这样, 我们可以得到这样一个协方差矩阵的最大似然估计可以由满足如下条件的对角矩阵 Σ 给出:

$$\Sigma = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

因此， Σ_{jj} 就是对数据中第 j 个坐标位置的方差值的经验估计。由于高斯模型的概率密度的轮廓是椭圆形的，故对角线矩阵 Σ 对于的就是椭圆长轴与坐标轴平行的高斯模型。有时候，我们还要对这个协方差矩阵给出进一步的约束，不仅设为对角的，还要求所有对角元素都相等。这时候，就有 $\Sigma = \sigma^2 I$ ，其中 σ^2 是我们控制的参数。对这个 σ^2 的最大似然估计则为：

$$\sigma^2 = \frac{1}{mn} \sum_{j=1}^n \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

这种模型对应的是密度函数为圆形轮廓的高斯模型（在二维空间中是圆形，在更高维度当中就是球（spheres）或者超球体（hyper-spheres））。

如果我们要拟合一个完整的，不受约束的协方差矩阵 Σ ，就必须满足 $m \geq n + 1$ ，这样才使得对 Σ 的最大似然估计不是奇异矩阵。在上面提到的两个约束条件只下，只要 $m \geq 2$ ，我们就能获得非奇异的 Σ 。

然而，将 Σ 限定为对角矩阵，也就意味着对数据中不同坐标的 x_i, x_j 建模都将是不相关的、而且相互独立。通常，还是从样本数据里面获得某些有趣的相关信息结构比较好；如果使用上面对 Σ 的约束，就可能没办法获取这些信息了。在本章讲义中，我们会提到因子分析模型（factor analysis model），这个模型使用的参数比对角矩阵 Σ 更多，而且能从数据中获得某些相关性信息，但也不能对完整的协方差矩阵进行拟合。

2. 多元高斯模型的边界和条件

在讲解因子分析之前，我们先讨论下一个联合多元高斯分布（a joint multivariate Gaussian distribution）下的随机变量的条件分布（conditional distribution）和边界分布（marginal distributions）。假如我们有一个值为向量的随机变量

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

其中， $x_1 \in R^r, x_2 \in R^s$ ，因此 $x \in R^{r+s}$ 。设 $x \sim N(\mu, \Sigma)$ ，即以 μ, Σ 为参数的正态分布，则这两个参数为：

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix},$$

其中， $\mu_1 \in R^r, \mu_2 \in R^s, \Sigma_{11} \in R^{r \times r}, \Sigma_{12} \in R^{r \times s}$ ，以此类推。由于协方差矩阵对称，所以有 $\Sigma_{21} = \Sigma_{12}^T$ 。

2.1 边界分布

基于我们的假设， x_1, x_2 是联合多元高斯分布。那么 x_1 的边界分布是什么？不难看出 x_1 的期望 $E[x_1] = \mu_1$ ，而协方差 $Cov(x_1) = E[(x_1 - \mu_1)(x_1 - \mu_1)^T] = \Sigma_{11}$ 。为了证明上述等式最右边的等号成立，即等于 Σ_{11} ，可以利用一下等式：

$$\begin{aligned}
Cov(x) &= \Sigma \\
&= \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \\
&= E[(x - \mu)(x - \mu)^T] \\
&= E \left[\begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix} \begin{pmatrix} x_1 - \mu_1 \\ x_2 - \mu_2 \end{pmatrix}^T \right] \\
&= E \left[\begin{pmatrix} (x_1 - \mu_1)(x_1 - \mu_1)^T & (x_1 - \mu_1)(x_2 - \mu_2)^T \\ (x_2 - \mu_2)(x_1 - \mu_1)^T & (x_2 - \mu_2)(x_2 - \mu_2)^T \end{pmatrix} \right]
\end{aligned}$$

通过上式的计算，我们可以证明等号成立。故高斯分布的边界分布本身也是高斯分布，所以我们可以给出正态分布 $x_1 \sim N(\mu_1, \Sigma_{11})$ 来作为 x_1 的**边界分布**。

2.2 条件分布

此外，我们还可以提出另一个问题，给定 x_2 的情况下 x_1 的条件分布是什么呢？通过参考多元高斯分布的定于，就能得到这个条件分布 $x_1|x_2 \sim N(\mu_{1|2}, \Sigma_{1|2})$ 为：

$$\begin{aligned}
\mu_{1|2} &= \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) \\
\Sigma_{1|2} &= \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}
\end{aligned}$$

根据条件概率公式，可以得到条件分布满足一下等式：

$$p(x_1|x_2) = \frac{p(x_1, x_2)}{p(x_2)} = \frac{p(x)}{p(x_2)}$$

其中， $x \sim N(\mu, \Sigma), x_2 \sim N(\mu_2, \Sigma_{22})$ 。将高斯分布概率公式代入可得条件分布的参数，计算比较复杂。

在下一节对因子分析模型的讲解中，上面这些公式就很有用了，可以帮助寻找高斯分布的条件和边界分布。

3. 因子分析模型

在因子分析模型中，我们制定一个在 (x, z) 上的联合分布，如下所示，其中 $z \in R^k$ 是一个潜在随机变量：

$$\begin{aligned}
z &\sim N(0, I) \\
x|z &\sim N(\mu + \Lambda z, \Psi)
\end{aligned}$$

上面的式子中，我们这个模型的参数是向量 $\mu \in R^n$ ，矩阵 $\Lambda \in R^{n \times k}$ ，以及一个对角矩阵 $\Psi \in R^{n \times n}$ 。 k 的值通常都选择比 n 小一点的。这样，我们就设想每个数据点 $x^{(i)}$ 都是通过在一个 k 维度的多元高斯分布 $z^{(i)}$ 中采样获得的。然后，通过计算 $\mu + \Lambda z^{(i)}$ ，就可以映射到实数域 R^n 中的一个 k 维仿射空间。最后，在 $\mu + \Lambda z^{(i)}$ 上加上协方差 Ψ 作为噪声，就得到了 $x^{(i)}$ 。

反过来，我们也可以使用下面的设定来定义因子分析模型：

$$\begin{aligned}
z &\sim N(0, I) \\
\epsilon &\sim N(0, \Psi) \\
x &= \mu + \Lambda z + \epsilon
\end{aligned}$$

其中的 z 和 ϵ 是相互独立的。然后我们来看这个模型定义的分布，其中，随机变量 z 和 x 有一个联合高斯分布：

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim N(\mu_{zx}, \Sigma)$$

然后我们要找到 μ_{zx}, Σ 。

已知 $E[z] = 0$, 此外我们可以计算得:

$$\begin{aligned} E[x] &= E[\mu + \Lambda z + \epsilon] \\ &= \mu + \Lambda E[z] + E[\epsilon] \\ &= \mu \end{aligned}$$

综合以上这些条件, 就得到了:

$$\mu_{zx} = \begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}$$

下一步就是要找出 Σ , 根据分块矩阵可以得到:

$$\begin{aligned} \Sigma &= \begin{bmatrix} E[(z - Ez)(z - Ez)^T] & E[(z - Ez)(x - Ex)^T] \\ E[(x - Ex)(z - Ez)^T] & E[(x - Ex)(x - Ex)^T] \end{bmatrix} \\ &= \begin{bmatrix} \Sigma_{zz} & \Sigma_{zx} \\ \Sigma_{xz} & \Sigma_{xx} \end{bmatrix} \end{aligned}$$

由于 z 是一个正态分布, 很容易知道 $\Sigma_{zz} = Cov(z) = I$ 。另外:

$$\begin{aligned} \Sigma_{zx}^T &= \Sigma_{xz} = E[(x - Ex)(z - Ez)^T] \\ &= E[(\mu + \Lambda z + \epsilon - \mu)(z - 0)^T] \\ &= E[zz^T]\Lambda + E[\epsilon z^T] \\ &= \Lambda \end{aligned}$$

同样的方法, 我们可以找到 Σ_{xx} :

$$\begin{aligned} \Sigma_{xx} &= E[(x - Ex)(x - Ex)^T] \\ &= E[(\Lambda z + \epsilon)(\Lambda z + \epsilon)^T] \\ &= E[\Lambda zz^T\Lambda^T + \epsilon z^T\Lambda^T + \Lambda z\epsilon^T + \epsilon\epsilon^T] \\ &= \Lambda\Lambda^T + \Psi \end{aligned}$$

故,

$$\begin{bmatrix} z \\ x \end{bmatrix} \sim N\left(\begin{bmatrix} \vec{0} \\ \mu \end{bmatrix}, \begin{bmatrix} I & \Lambda^T \\ \Lambda & \Lambda\Lambda^T + \Psi \end{bmatrix}\right)$$

因此, 我们还能发现 x 的边界分布为 $x \sim N(\mu, \Lambda\Lambda^T + \Psi)$ 。所以, 给定一个训练样本集合 $\{x^{(i)}; i = 1, \dots, m\}$, 我们可以写成参数的最大似然估计函数的对数形式:

$$l(\mu, \Lambda, \Psi) = \log \prod_{i=1}^m \frac{1}{(2\pi)^{(n/2)} |\Lambda\Lambda^T + \Psi|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu)^T (\Lambda\Lambda^T + \Psi)^{-1} (x^{(i)} - \mu)\right)$$

为了进行最大似然估计，我们就要最大化上面这个关于参数的函数。但确切地对上面这个方程进行最大化，是很难的，而且我们都知道没有算法能够以封闭形式来实现这个最大化。所以，我们就改用期望最大化算法。

4. 因子分析模型的期望最大化算法

E 步骤的推导很简单。只需要计算出来 $Q_i(z^{(i)}) = p(z^{(i)}|x^{(i)}; \mu, \Lambda, \Psi)$ 。根据多元高斯模型及其条件分布公式可知： $z^{(i)}|x^{(i)}; \mu, \Lambda, \Psi \sim N(\mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}})$ ， 其中：

$$\begin{aligned}\mu_{z^{(i)}|x^{(i)}} &= \Lambda^T (\Lambda \Lambda^T + \Psi)^{-1} (x^{(i)} - \mu) \\ \Sigma_{z^{(i)}|x^{(i)}} &= I - \Lambda^T (\Lambda \Lambda^T + \Psi)^{-1} \Lambda\end{aligned}$$

所以，通过对 $\mu_{z^{(i)}|x^{(i)}}, \Sigma_{z^{(i)}|x^{(i)}}$ 进行这样的定义，就能得到：

$$Q_i(z^{(i)}) = \frac{1}{(2\pi)^{k/2} |\Sigma_{z^{(i)}|x^{(i)}}|^{1/2}} \exp\left(-\frac{1}{2}(z^{(i)} - \mu_{z^{(i)}|x^{(i)}})^T \Sigma_{z^{(i)}|x^{(i)}}^{-1} (z^{(i)} - \mu_{z^{(i)}|x^{(i)}})\right)$$

接下来就是 M 步骤了，这里需要最大化下面这个关于参数 μ, Λ, Ψ 的函数值：

$$\sum_{i=1}^m \int_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)}, z^{(i)}; \mu, \Lambda, \Psi)}{Q_i(z^{(i)})} dz^{(i)} \quad (4)$$

在本文中我们仅对 Λ 进行优化，关于 μ, Ψ 的更新就作为练习留给读者自己进行推导了。把等式 (4) 简化成下面的形式：

$$\sum_{i=1}^m \int_{z^{(i)}} Q_i(z^{(i)}) [\log p(x^{(i)}|z^{(i)}; \mu, \Lambda, \Psi) + \log p(z^{(i)}) - \log Q_i(z^{(i)})] dz^{(i)} \quad (5)$$

$$\sum_{i=1}^m E_{z^{(i)} \sim Q_i} [\log p(x^{(i)}|z^{(i)}; \mu, \Lambda, \Psi) + \log p(z^{(i)}) - \log Q_i(z^{(i)})] \quad (6)$$

上面的等式中，下标 $z^{(i)} \sim Q_i$ 表示的意思是这个期望是关于从 Q_i 中取得的 $z^{(i)}$ 的。在后续的推导过程中，如果没有歧义的情况下，我们就会把这个下标省略掉。观察等式 (6)，事实上我们只需要最大化第一项，因为第二项 $p(z^{(i)})$ 、第三项 $Q_i(z^{(i)})$ 不依赖参数；第二项不依赖参数很好理解， $z \sim N(0, I)$ ；第三项看似含有参数，但参数在 E 步骤中已经固定，不影响结果。故省略部分项目后，我们只需要最大化下式：

$$\begin{aligned}&\sum_{i=1}^m E[\log p(x^{(i)}|z^{(i)}; \mu, \Lambda, \Psi)] \\ &= \sum_{i=1}^m E[\log \frac{1}{(2\pi)^{(n/2)} |\Psi|^{1/2}} \exp\left(-\frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)})\right)] \\ &= \sum_{i=1}^m E[-\frac{1}{2} \log |\Psi| - \frac{1}{2} \log(2\pi) - \frac{1}{2} (x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)})]\end{aligned}$$

我们先对上面的函数进行关于 Λ 的最大化。可见只有最后的一项依赖 Λ ，同时利用下面几个结论：

$$\begin{aligned}
tra &= a & a \in R \\
trAB &= trBA \\
\nabla_A trABA^TC &= CAB + C^T AB
\end{aligned}$$

可以得到：

$$\begin{aligned}
\nabla_{\Lambda} \sum_{i=1}^m E[-\frac{1}{2}(x^{(i)} - \mu - \Lambda z^{(i)})^T \Psi^{-1} (x^{(i)} - \mu - \Lambda z^{(i)})] \\
&= \sum_{i=1}^m \nabla_{\Lambda} E[-tr \frac{1}{2} (z^{(i)})^T \Lambda^T \Psi^{-1} \Lambda z^{(i)} + tr (z^{(i)})^T \Lambda^T \Psi^{-1} (x^{(i)} - \mu)] \\
&= \sum_{i=1}^m \nabla_{\Lambda} E[-tr \frac{1}{2} \Lambda^T \Psi^{-1} \Lambda z^{(i)} (z^{(i)})^T + tr \Lambda^T \Psi^{-1} (x^{(i)} - \mu) (z^{(i)})^T] \\
&= \sum_{i=1}^m E[-\Psi^{-1} \Lambda z^{(i)} (z^{(i)})^T + \Psi^{-1} (x^{(i)} - \mu) (z^{(i)})^T]
\end{aligned}$$

令导数为 0，然后简化，就能得到：

$$\sum_{i=1}^m \Lambda E_{z^{(i)} \sim Q_i} [z^{(i)} z^{(i)T}] = \sum_{i=1}^m (x^{(i)} - \mu) E_{z^{(i)} \sim Q_i} [z^{(i)T}]$$

接下来，求解 Λ ，就能得到：

$$\Lambda = \left(\sum_{i=1}^m (x^{(i)} - \mu) E_{z^{(i)} \sim Q_i} [z^{(i)T}] \right) \left(\sum_{i=1}^m E_{z^{(i)} \sim Q_i} [z^{(i)} z^{(i)T}] \right)^{-1} \quad (7)$$

有一个很有意思的地方需要注意，上面这个等式和用最小二乘线性回归推出的正则方程有密切关系：

$$\theta^T = (y^T X)(X X^T)^{-1}$$

与之类似，这里的 x 是一个关于 z （以及噪声）的线性方程。考虑在 E 步骤中对 z 已经给出了猜测，接下来就可以尝试来对 x, z 之间的未知线性量 Λ 进行估计。接下来不出意料，我们就会得到某种类似正则方程的结果。然而，这个还是和利用对 z 的“最佳猜测”进行最小二乘算法有一个很大的区别。这一点我们很快就会看到了。

为了完成 M 步骤的更新，接下来我们要解出等式 (7) 当中的期望值。根据 Q_i 的均值和协方差，以及公式 $Cov(Y) = E[YY^T] - E[Y]E[Y]^T$ ，得到：

$$\begin{aligned}
E_{z^{(i)} \sim Q_i} [z^{(i)T}] &= \mu_{z^{(i)}|x^{(i)}}^T \\
E_{z^{(i)} \sim Q_i} [z^{(i)} z^{(i)T}] &= \mu_{z^{(i)}|x^{(i)}} \mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}}
\end{aligned}$$

把这个公式代入等式 (7)，就得到了 M 步骤中 Λ 的更新规则：

$$\Lambda = \left(\sum_{i=1}^m (x^{(i)} - \mu) \mu_{z^{(i)}|x^{(i)}}^T \right) \left(\sum_{i=1}^m \mu_{z^{(i)}|x^{(i)}} \mu_{z^{(i)}|x^{(i)}}^T + \Sigma_{z^{(i)}|x^{(i)}} \right)^{-1} \quad (8)$$

上面这个等式中，要特别注意等号右边这一侧的 $\Sigma_{z^{(i)}|x^{(i)}}$ 。这是一个 $z^{(i)}$ 的后验分布的协方差，在 M 步骤中必须要考虑到这个后验分布中 $z^{(i)}$ 的不确定性。

最后，我们对参数 μ, Ψ 进行优化，不难发现其中的 μ 为：

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

由于这个值不随着参数的变化而改变（也就是说，和 Λ 的更新不同，这里等式右侧不依赖 $Q_i(z^{(i)}) = p(z^{(i)} | x^{(i)}; \mu, \Lambda, \Psi)$ ，其中这个 $Q_i(z^{(i)})$ 是依赖参数的），所以这个只需要计算一次就可以，在算法运行过程中，也不需要进一步更新。类似的，对角矩阵 Ψ 也可以通过计算下面这个式子来获得：

$$\Phi = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} - x^{(i)} \mu_{z^{(i)} | x^{(i)}}^T \Lambda^T - \Lambda \mu_{z^{(i)} | x^{(i)}} x^{(i)T} + \Lambda (\mu_{z^{(i)} | x^{(i)}} \mu_{z^{(i)} | x^{(i)}}^T + \Sigma_{z^{(i)} | x^{(i)}}) \Lambda^T$$

然后只需要设 $\Psi_{ii} = \Phi_{ii}$ （也就是说，设 Ψ 为一个仅仅包含矩阵 Φ 中对角线元素的对角矩阵）。

1. 主成分分析算法

1.1 预处理

1.2 主成分分析

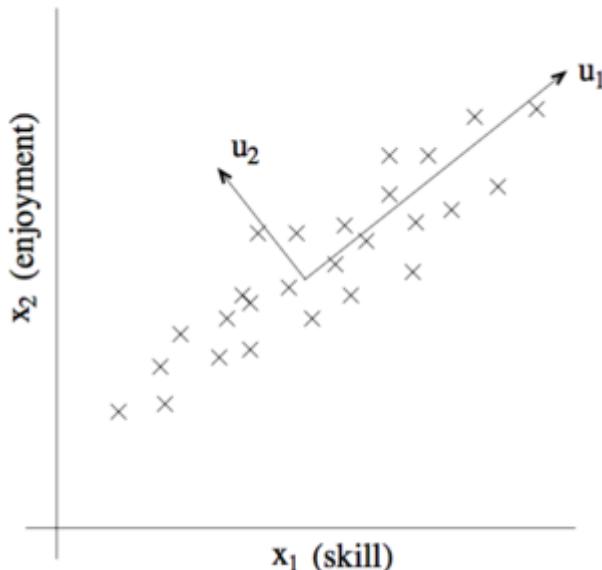
2. 备注

在因子分析 (factor analysis) 模型中，我们在某个 k 维度子空间对 $x \in R^n$ 进行近似建模，其中 $k << n$ 。具体来说，我们设想每个点 $x^{(i)}$ 用如下方法创建：首先在 k 维度仿射空间 (affine space) 中生成某个 $z^{(i)}$ ，然后计算公式 $\mu + \Lambda z$ ，最好增加均值为 0，协方差为 Ψ 的噪声。因子分析是一个基于概率模型，然后使用迭代期望最大化算法进行参数估计 (parameter estimation) 的算法。

在本章中，我们要学习一种新的方法：**主成分分析** (Principal Components Analysis, PCA)，这个方法也是用来对数据近似所处的子空间进行判别 (identity)。然而，主成分分析算法 (PCA) 会更加直接，只需要进行一种特征向量计算 (在 Matlab 里面可以通过 `eig` 函数轻松实现)，不需要再去使用期望最大化 (EM) 算法。

1. 主成分分析算法

设想有一个数据集，其中包含的是对一个无线遥控直升机飞行员协会进行调查得到的数据，其中的 $x_1^{(i)}$ 指的是飞行员 i 的飞行技能， $x_2^{(i)}$ 指的是该飞行员对飞行的喜爱程度。无线遥控直升机是很难操作的，只有那些非常投入并且热爱飞行的学生，才能称为好的飞行员。所以，上面这两个属性 x_1 和 x_2 之间应该**强相关**。故我们可以认为在数据中沿着对角线方向 (即下图中的 u_1 方向) 表征了一个人对飞行投入程度的“源动力” (karma)，只有少量的噪声脱离这个对角线方向。如下图所示，我们怎么自动计算出 u_1 的方向呢？（或者说， x_1, x_2 存在冗余，我们如何自动检测和删除这一冗余。）



接下来我们就要讲主成分分析算法 (PCA algorithm) 了。但在运行 PCA 之前，我们首先要进行一些预处理，正则化数据的均值和方差。

1.1 预处理

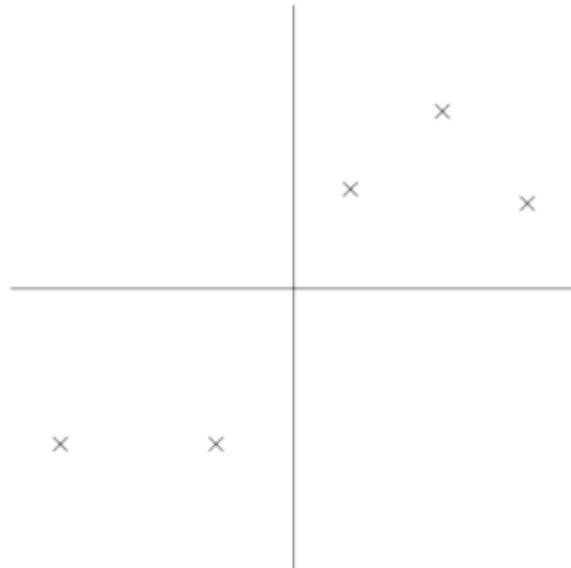
1. 设 $\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$;
2. 将每个 $x^{(i)}$ 替换成 $x^{(i)} - \mu$;
3. 设 $\sigma_j^2 = \frac{1}{m} \sum_i (x_j^{(i)})^2$;
4. 将每个 $x_j^{(i)}$ 替换成 $x_j^{(i)} / \sigma_j$;

步骤 1 和 2 把数据的平均值清零；步骤 3 和 4 将每个坐标缩放，使之具有单位方差，确保不同的属性都在同样的“尺度”上。例如，如果 x_1 是汽车的最大速度（以 mph 为单位，精确到十位）；然后 x_2 是汽车的座位数量（取值一般在 2 - 4）。这样这个重新正则化就把不同的属性进行了缩放，然后这些不同属性就更具有对比性。如果我们事先已经知道不同的属性在同一尺度上，就可以省略第 3、4 步。

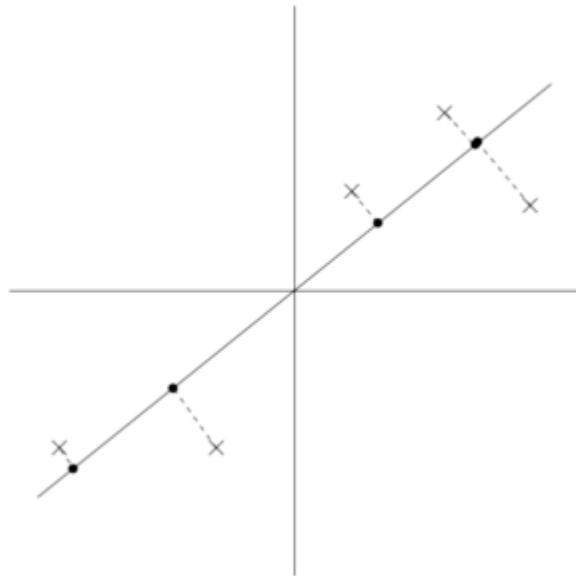
1.2 主成分分析

接下来，进行正则化之后，对数据近似所处的方向，也就是“变化的主轴” \vec{u} ，该如何进行计算？一种方法是找出一个单位向量 \vec{u} ，使得数据在向量 \vec{u} 上投影时，投影的数据的方差最大。直观上看，在这个方向上，数据一开始就有一定规模的 **方差 / 信息量** (variance / information)。我们要选择的是这样一个方向的单位向量 \vec{u} ：数据能近似位于与向量 \vec{u} 一致的 **方向 / 子空间** (direction / subspace) 中，并且尽可能多地保留上面的方差。

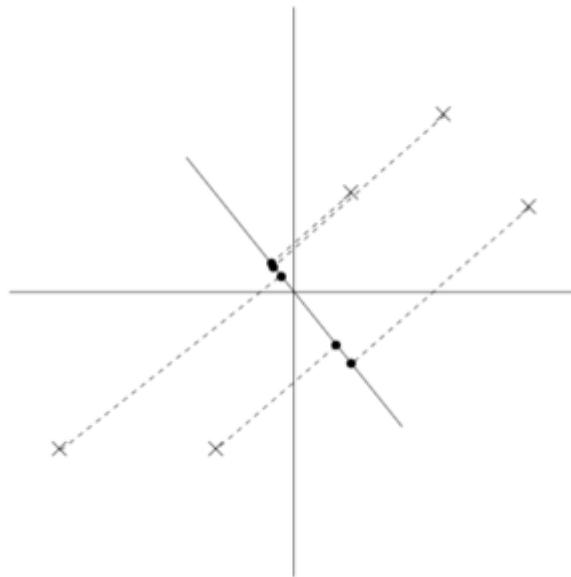
设下面的数据集，我们已经进行了正则化步骤：



现在，加入我们选择的 \vec{u} ，对应于下图中所示的方向。下图中的圆点表示的就是原始数据在这条线上的投影。



可以看到，上面投影得到的数据依然有还算比较大的方差，而且这些投影点距离零点也都比较远。反例则如下图所示，我们选择另一个方向上的向量：



上面这幅图的投影中的方差就明显小了很多，而且投影得到的点位置也距离原点近很多。

我们希望能自动地选择出来如上面两幅通知第一幅那样方向的单位向量 \vec{u} 。要对这个过程进行方程化，要注意到给定一个向量 \vec{u} 和一个点 x ， x 投影到 u 上的投影长度可以通过 $x^T u$ 来得到。也就是说，如果 $x^{(i)}$ 是我们数据集中的一点，那么这个点在 u 上的投影就是从原点到 $x^{(i)T} u$ 的距离。因此，要最大化投影的方差，就要找到一个能够将下面式子最大化的单位长度向量 u ：

$$\begin{aligned} \arg \max_u \frac{1}{m} \sum_{i=1}^m (x^{(i)T} u)^2 &= \frac{1}{m} \sum_{i=1}^m u^T x^{(i)} x^{(i)T} u \\ &= u^T \left(\frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T} \right) u \end{aligned}$$

很容易就能发现，要让上面的式子最大化，需满足如下条件：向量 $\|u\| = 1$ 且为 $\Sigma = \frac{1}{m} \sum_{i=1}^m x^{(i)} x^{(i)T}$ 的**主特征向量** (principal eigenvector)，其中 Σ 正好是数据的经验协方差矩阵 (数据正则化后均值为零)。

总结一下，如果我们要找一个 1 维度子控件来近似数据，就要选择 Σ 的主特征向量（principal eigenvector）作为单位向量 u 。更广义的理解，就是如果要将数据投影到一个 k 维度子空间 ($k < n$)，就应当选择 Σ 的 k 个特征向量来作为单位向量 u_1, \dots, u_k 。这里的 u_i 就形成了数据的一组新的正交基。然后，要使用这组正交基来表示 $x^{(i)}$ ，只需要进行如下计算：

$$y^{(i)} = \begin{bmatrix} u_1^T x^{(i)} \\ u_2^T x^{(i)} \\ \vdots \\ \vdots \\ u_k^T x^{(i)} \end{bmatrix} \in R^k$$

$x^{(i)} \in R^n$ ，向量 $y^{(i)} \in R^k$ 就是对 $x^{(i)}$ 的 k 低维度的近似 / 表示。因此，主成分分析算法（PCA）也被称为是一种维度降低算法（dimensionality reduction algorithm），其中的单位向量 u_1, u_2, \dots, u_k 也称为数据集的前 k 个主成分（principal components）。我们对基向量的选择应当是尽可能得保留原始数据的方差信息。

2. 备注

在习题集 4 中，你会发现主成分分析算法（PCA）也可以有另外一种推导方式：将数据投影到数据所张成的 k 维度子空间中，然后选择一组基向量，使得投影引起的近似误差最小。

主成分分析算法（PCA）有很多用法：我们接下来就给出若干样例来收尾这部分。首先是压缩，即降维：用更低纬度的 $y^{(i)}$ 来表示 $x^{(i)}$ 。如果我们把高维度的数据降到 $k = 2$ or 3 ，那么就可以将 $y^{(i)}$ 进行可视化了。例如，如果我们把汽车数据降到 2 维度，那么就可以把压缩后的数据画出来，来看看哪些车彼此相似、可以聚类成组。

另一个常见的应用是：在 $x^{(i)}$ 作为输入特征进行监督学习算法前，作为降低数据维度的预处理步骤。除了有助于缓解计算压力外，降低数据维度还可以降低假设类（hypothesis class）的复杂度，然后避免过拟合。（例如：低纬度的输入特征上的线性分类器会有更小的 VC 维度）

最后，正如在遥控直升机飞行员那个样例，我们可以把 PCA 用作一种降噪算法。在该例子中，可以使用算法在对飞行技巧和热爱程度的含噪数据中估计直观的“遥控飞行源动力（piloting karma）”。在课程中，我们还看到了把这种思路用在人脸图像上，得到的就是面部特征算法。其中每个点 $x^{(i)} \in R^{100 \times 100}$ 都是一个 10000 维度的向量，每个坐标对应的是人脸图像中的一个像素灰度值。使用主成分分析算法，我们就可以用更低维度的 $y^{(i)}$ 来表示每个图像 $x^{(i)}$ 。在这个过程中，我们希望主成分能够保存面孔间有趣的、系统的变化，而不是由于细微的光线变化、轻微的拍摄状况等差别引入的“噪声”。然后我们通过计算降低维度后的 $\|y^{(i)} - y^{(j)}\|$ 来测量面孔 i, j 之间的距离。这样就能得到一个令人惊艳的面部匹配和检索算法。

下表为目前无监督学习方式下的算法总结：主要根据算法的类型、算法处理的数据集类型进行区分。

		非概率算法	密度估计算法
数据在团块中 clumps	“聚类”问题	k 均值聚类 (k-means)	高斯混合模型 (MoG)
数据在子空间中 subspace	“降维”问题	主成分分析 (PCA)	因子分析 (Factor analysis)

12. 独立成分分析 (ICA)

1. 独立成分分析的不确定性 (ICA ambiguities)

2. 密度函数和线性变换

3. 独立成分分析算法

接下来我们要讲的就是**独立成分分析** (Independent Components Analysis, ICA) 算法。这个方法和主成分分析 (PCA) 类似，也是要找到一组新的基向量 (basis) 来表征 (represent) 样本数据。然而，这两个方法的目的却截然不同。

我们还以“鸡尾酒会问题”为例。在一个聚会中，有 n 个人同时说话，并且放置在房间内的任何话筒仅记录这 n 个人叠加在一起的声音。如果假设我们也有 n 个不同的话筒安装在屋子里，并且这些话筒与每个说话人的距离都各自不同，那么记录的也就是不同组合形式的叠加声音。使用这样布置的 n 个话筒来录音，能不能区分开原始的 n 个说话者每个人的声音信号呢？

把这个问题用方程的形式来表示，我们需先假设时刻 i 有某样本数据 $s^{(i)} = (s_1^{(i)}, s_2^{(i)}, \dots, s_n^{(i)})^T \in R^n$ ，这个数据是由 n 个独立的源 (independent sources) 在时刻 i 生成的，例如 $s_j^{(i)}$ 是第 j 个说话者在时刻 i 发出的声音。总样本数据为 $s = (s^{(1)}, s^{(2)}, \dots, s^{(m)}) \in R^{n \times m}$ ，表示 n 个说话者在 m 个时刻中生成的总数据。我们观察到的则为：

$$\begin{aligned} x^{(i)} &= As^{(i)} \\ x &= As \end{aligned}$$

上面式子中的 A 是一个未知的方形矩阵，叫做**混合矩阵** (mixing matrix)。通过重复的观察，我们就得到了训练样本 $x = \{x^{(i)}; i = 1, \dots, m\} \in R^{n \times m}$ ，其中 $x^{(i)}$ 类似于 $s^{(i)}$ ，例如 $x_j^{(i)}$ 是第 j 个话筒在时刻 i 记录的声音。我们的目的是恢复出这些样本 $x^{(i)} = As^{(i)}$ 的原始声音源 $s^{(i)}$ 。

$$x = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ | & | & & | \end{bmatrix} = \begin{bmatrix} | & & & | \\ As^{(1)} & As^{(2)} & \dots & As^{(m)} \\ | & | & & | \end{bmatrix} = A \begin{bmatrix} s_1^{(1)} & s_1^{(2)} & \dots & s_1^{(m)} \\ s_2^{(1)} & s_2^{(2)} & \dots & s_2^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ s_n^{(1)} & s_n^{(2)} & \dots & s_n^{(m)} \end{bmatrix} = As$$

令 $W = A^{-1}$ ，称之为**还原矩阵** (unmixing matrix)。我们的目标就是找到 W ，这样针对给定的 $x^{(i)}$ ，就可以通过计算 $s^{(i)} = Wx^{(i)}$ 来还原出声音源。为了方便，我们用 w_j^T 来表示 W 的第 j 行，如下：

$$W = \begin{bmatrix} w_1^T \\ \vdots \\ w_n^T \end{bmatrix}$$

其中， $w_j \in R^n$ ，然后就可以通过计算 $s_j^{(i)} = w_j^T x_j^{(i)}$ 恢复出第 j 个声源了。

1. 独立成分分析的不确定性 (ICA ambiguities)

下面是两个独立成分分析的不确定性：

第一个，由于 W 和 s 都不确定，那么在没有先验证知识的情况下，无法同时确定这两个相关参数。比如上面的公式 $s = Wx$ 。当 W 扩大两倍时， s 只需要扩大两倍即可，等式仍然成立，因此无法得到唯一的 s 。

第二个，如果将 s 的顺序打乱，变成另外一个顺序，那么只需要调整 A 的列向量顺序即可，因此也无法单独确定 s 。

另外，还有一种独立成分分析不适用的情况，那就是信号不能是高斯分布的。当源信号是高斯分布的时候，可以由不同的混合矩阵 A 乘以 s ，得到相同的 x ，一样无法确定源信号。

2. 密度函数和线性变换

在继续推导独立成分分析 (ICA) 算法之前，我们先来简要讲一讲线性变换对密度函数的影响 (effect)。

假设我们有一个概率密度函数为 $p_s(s)$ 的随机变量 s 。简单起见，我们把 s 当做一个实数，即 $s \in R$ 。然后，设又有一随机变量 x ， x 定义为 $x = As$ 。那么 x 的概率密度函数 $p_x(x)$ 是多少呢？在计算之前，我们先讨论一下概率分布函数 $F(x) = P(X \leq x)$ 与概率密度函数 $f(x) = p(x)$ 之间的关系吧，事实上，概率分布函数是概率密度函数的积分，概率密度函数是概率分布函数的导数，二者关系如下：

$$F(x) = \int_{-\infty}^x f(x)dx$$

$$f(x) = F'(x)$$

因此，关于 $p_x(x)$ 的推导如下：

$$F_x(x) = P(X \leq x) = P(AS \leq x) = P(S \leq Wx) = F_s(Wx)$$

$$p_x(x) = F'_x(x) = F'_s(Wx) = p_s(Wx)|W|$$

即， $p_x(x) = p_s(Wx)|W|$ ($|W|$ 其实也可以分步求导解释)

3. 独立成分分析算法

现在就可以推导独立成分分析 (ICA) 算法了。我们这里描述的算法来自于 Bell 和 Sejnowski，对算法的解释使用最大似然估计的方法。我们假设每个声源的分布 s_i 的概率密度函数为 p_s ，那么联合分布 s 则为：

$$p(s) = \prod_{i=1}^n p_s(s_i)$$

这里要注意，由于我们假设每个声源都是独立的，故可以在建模中将联合分布直接写成边界分布的乘积。利用上一节推导的结论，这就表明 $x = As = W^{-1}s$ 的密度函数为：

$$p(x) = p_s(Wx)|W| = |W| \prod_{i=1}^n p_s(w_i^T x)$$

剩下的就是为单个源 p_s 指定一个密度。回忆一下，给定一个实数值的随机变量 z ，其累积分布函数（即概率分布函数） F 的定义为： $F(z_0) = P(z \leq z_0) = \int_{-\infty}^{z_0} p_z(z)dz$ 。然后对这个累积分布函数求导，就能得到 z 的密度函数： $p_z(z) = F'(z)$ 。

因此，要确定 s_i 的密度函数，首先要做就是确定其累积分布函数。根据我们之前的讨论，这里不能选用高斯分布的累积分布函数，因为独立成分分析不适用于高斯分布的数据。这里我们选择一个保证从 0 到 1 单调递增函数即可，比如 s 形函数 (sigmoid function) $g(s) = 1/(1 + e^{-s})$ 。这样就有： $p_s(s) = g'(s)$ 。

模型中的参数 W 是一个正方形矩阵。给定一个训练集合 $\{x^{(i)}; i = 1, \dots, m\}$ ，对数似然函数则为：

$$l(W) = \sum_{i=1}^m \left(\sum_{j=1}^n \log g'(w_j^T x^{(i)}) + \log |W| \right)$$

我们要做的就是选择合适的 W 最大化上式。通过求导和前面讲义中的定理： $\nabla_W |W| = |W|(W^{-1})^T$ ，就可以很容易推导出随机梯度上升的学习规则。对一个给定的训练样本 $x^{(i)}$ ，这个更新规则为：

$$W := W + \alpha \begin{pmatrix} \left[\begin{array}{c} 1 - 2g(w_1^T x^{(i)}) \\ 1 - 2g(w_2^T x^{(i)}) \\ \vdots \\ 1 - 2g(w_n^T x^{(i)}) \end{array} \right] x^{(i)T} + (W^T)^{-1} \end{pmatrix}$$

上式中的 α 是学习速率。在算法收敛之后，就能计算出 $s^{(i)} = Wx^{(i)}$ ，这样就能恢复出原始的音源了。

1. 马尔可夫决策过程

1.1 马尔可夫决策过程

1.2 策略、值函数

1.3 最优值函数、最佳策略

2. 值迭代和策略迭代

2.1 值迭代 (value iteration)

2.2 策略迭代 (policy iteration)

3. 马尔可夫决策过程的自学习模型

4. 备注

这一章我们开始学习**强化学习** (reinforcement learning) 和**适应性控制** (adaptive control)。在监督学习中，算法的输出都在模仿训练集给出的标签 y ；这种情况下，对于每个输入特征 x ，都有一个对应的标签作为明确的“正确答案”。与之相反，对于许多数学决策和控制问题，很难提供给学习算法一种**明确的显示监督** (explicit supervision)。例如，假如我们制作了一个“四腿机器人”，然后要编程让它能走路，但我们并不知道怎么去采取“正确”的动作来进行四条腿的行走，所以就不能给它提供一个明确的监督学习算法来进行模仿。在强化学习的框架下，我们将仅为我们的算法提供一个**奖励函数** (reward function)，该函数会告知学习算法何时表现良好，何时表现不佳。在“四腿机器人”中，奖励函数会在机器人进步的时候给出正面回馈，即奖励；在机器人退步或摔倒的时候给出负面回馈，即惩罚。然后，学习算法的工作就是判定如何选择动作已得到最大奖励。

强化学习 (Reinforcement Learning, RL) 已经成功用于多种场景了，例如无人直升机的自主飞行、机器人的腿部运动、手机的网络路由、市场营销的策略筛选、工厂控制、高效率的网页索引等等。我们对强化学习的探索，先从**马尔可夫决策过程** (Markov decision processes, MDP) 开始，这个概念给出了强化学习问题中的常见形式。

1. 马尔可夫决策过程

1.1 马尔可夫决策过程

一个马尔可夫决策过程 (Markov decision processes, MDP) 由一个元组: $(S, A, \{P_{sa}\}, \gamma, R)$ 组成，其中的元素分别为：

- S 是一个**状态集合** (a set of **states**)。（例如，在无人直升机的案例中， S 就是直升机所有可能的位置和方向的集合。）
- A 是一个**动作集合** (a set of **actions**)。（例如，还以无人直升机为例， A 就是遥控器上能够操作的所有动作的集合）
- P_{sa} 为**状态转移概率**。对于每个状态 $s \in S$ 和动作 $a \in A$ ， P_{sa} 是一个在状态空间上的分布。简单地说， P_{sa} 给出的是在状态 s 下进行一个动作 a 之后转移到的状态的分布。
- $\gamma \in [0, 1]$ 叫做**折扣因子** (discount factor)。
- $R : S \times A \mapsto R$ 就是**奖励函数** (reward function)。（奖励函数也可以写成仅对状态 S 的函数，这样就可以写成 $R : S \mapsto R$ ）

马尔可夫决策过程的动态更新如下：在某个起始状态 s_0 启动，然后选择某个动作 $a_0 \in A$ 来执行 MDP。根据所选的动作会有对应的结果，即转移到某个后继状态 s_1 ，其服从分布 $s_1 \sim P_{s_0 a_0}$ 。然后再选择另外一个动作 a_1 ，接下来又有对应这个动作的转移状态 $s_2 \sim P_{s_1 a_1}$ 。接着选择一个动作 a_2 ，就这样进行下去。可以用下面的过程作为表示：

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} s_3 \xrightarrow{a_3} \dots$$

通过序列中的所有状态 s_0, s_1, \dots 和对应的动作 a_0, a_1, \dots , 我们就能得到总奖励值, 即总收益函数 (total payoff) 为:

$$R(s_0, a_0) + \gamma R(s_1, a_1) + \gamma^2 R(s_2, a_2) + \dots$$

如果我们把奖励函数作为仅与状态相关的函数, 那么就可以简化为:

$$R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots$$

多数情况下, 我们都用后面这种仅为“状态”的函数, 虽然扩展到“状态—动作”的函数 $R(s, a)$ 也并不难。

强化学习的目标就是找到一组动作, 使得总收益函数的期望值最大:

$$E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots]$$

注意, 在时间步长 t 上的奖励函数通过参数 γ^t 进行了衰减。因此, 要使得期望最大化, 就需要尽早获得奖励、而推迟惩罚的出现。在经济方面中, 可以将 R 理解为盈利额, γ 理解为利率, 这样可以用一种较为自然解释: 今天的一美元就比明天的一美元更有价值。

1.2 策略、值函数

有一种策略 (policy), 是使用任意函数 $\pi: S \mapsto A$, 从状态到动作进行映射。如果在状态 s , 采取动作 $a = \pi(s)$, 就可以说是正在执行策略 π 。我们还可以针对策略 π 定义一个值函数 (value function) :

$$\begin{aligned} V^\pi(s) &= E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi] \\ &= E\left[\sum_{i=0}^{\infty} \gamma^i R(s_i) | s_0 = s, \pi\right] \end{aligned}$$

$V^\pi(s)$ 就是从状态 s 开始, 根据 π 给出的动作积累的折扣奖励 (discounted rewards) 的期望和。

事实上我们在期望的表示中使用的记号 π , 严格来说不太正确。因为 π 并不是一个随机变量, 不过在很多文献里面都这样表示, 已经成了某种标准。

如果我们用 s' 表示下一步状态, 即 $s' \sim P_{s\pi(s)}$; 由于概率分布归一化条件, 即 $\sum_{s' \in S} P_{s\pi(s)} = 1$, 则 $V^\pi(s)$ 还可以写为:

$$V^\pi(s) = E[R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots | s_0 = s, \pi] \quad (1)$$

$$= E\left[\sum_{i=0}^{\infty} \gamma^i R(s_i) | s_0 = s, \pi\right] \quad (2)$$

$$= E[R(s_0) + \gamma \sum_{i=1}^{\infty} \gamma^{(i-1)} R(s_i) | s_0 = s, \pi] \quad (3)$$

$$= E[R(s_0) + \gamma E\left[\sum_{i=1}^{\infty} \gamma^{(i-1)} R(s_i) | s_0 = s, \pi\right]] | s_0 = s, \pi \quad (4)$$

$$= E[R(s_0) + \gamma V^{(\pi)}(s_1) | s_0 = s, s_1 = s', \pi] \quad (5)$$

$$= \sum_{s' \in S} P_{s\pi(s)} \left(R(s) + \gamma V^{(\pi)}(s') \right) \quad (6)$$

$$= R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{(\pi)}(s') \quad (7)$$

其中步骤 (3) 到步骤 (4) 根据 “奖励的期望等于奖励的期望的期望” 可以推导出来；步骤 (5) 到步骤 (6) 相当于按期望的定义展开。针对等式 (7)，我们有一个专业的说法：给定一个固定的策略 π ，其对应的值函数 $V^\pi(s)$ 满足 **贝尔曼等式** (Bellman equations)，即：

$$V^\pi(s) = R(s) + \gamma \sum_{s' \in S} P_{s\pi(s)}(s') V^{(\pi)}(s')$$

这也就意味着，从状态 s 开始的折扣奖励 (discounted rewards) 的期望和即值函数 $V^\pi(s)$ 由两部分组成：1) 是在状态 s 的时候立即获得的奖励函数 $R(s)$ ，也就是上面式子的第一项；2) 后续的折扣奖励的期望和乘以折扣因子。贝尔曼等式可以有效地解出 V^π ，尤其是一个有限状态的 MDP 过程 ($|S| < \infty$)。我们可以把每个状态 s 对应的 $V^\pi(s)$ 的方程写出来，这样就得到了一系列的 $|S|$ 个线性方程，有 $|S|$ 个变量，这些 $V^\pi(s)$ 都很容易解出来。

1.3 最优值函数、最佳策略

我们还可以定义最优值函数 (optimal value function)：

$$V^*(s) = \max_{\pi} V^\pi(s)$$

换句话说，这个值是在所有可能的策略中能得到的最大的折扣奖励的期望和（每一个策略 $\pi(s)$ ，都会有有个值函数，最优值函数就是所有值函数中的最大的那个）。对于最优值函数，也有一个版本的贝尔曼等式：

$$V^*(s) = R(s) + \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

上面这个等式中的第一项，还是跟之前的一样，还是即时奖励函数 $R(s)$ ；第二项是在所有的可能的动作中，后续的值函数的最大值。

另外还可以定义最佳策略： $\pi^*(s) : S \mapsto A$

$$\pi^*(s) = \arg \max_{a \in A} \sum_{s' \in S} P_{sa}(s') V^*(s')$$

这里的 $\pi^*(s)$ 给出的动作 a 都能使贝尔曼等式的第二项取得最大值（这也是称为最佳策略的原因）。对于每个状态 s 和每个策略 π ，都有：

$$V^*(s) = V^{\pi^*}(s) \geq V^\pi(s)$$

重点：上面的第一个等式表明：最佳策略 π^* 的值函数 $V^{\pi^*}(s)$ 等于它的最优值函数 $V^*(s)$ 。右边的不等式表明：最佳策略 π^* 生成的值函数是所有策略 π 的最大值，即最佳策略。

重点：上面的第一个等式表明：最佳策略 π^* 的值函数 $V^{\pi^*}(s)$ 等于它的最优值函数 $V^*(s)$ 。右边的不等式表明：最佳策略 π^* 生成的值函数是所有策略 π 的最大值，即最佳策略。

注意，这个最佳策略 π^* 有一个有趣的特性，它是所有状态 s 的最佳策略（个人理解：因为最佳策略的递归中有下一个状态 s' ，所以解得的最佳策略不依赖于起始状态。我们不妨把问题简单化，即每个动作 a 之后得到的状态是确定的，而不是一个状态空间上的分布，那么我们总能找到一组最佳的动作，生成这组最佳动作的就是最佳策略 π^* 。）具体来说，就是最佳策略 π^* 不会因为起始状态的不同而变化。这也就意味着无论马尔可夫决策过程的初始状态如何，都可以使用同样的策略函数 π^* 。

2. 值迭代和策略迭代

现在我们开始讲解两种不同的算法，都能有效地解决有限状态的马尔可夫决策问题（finite-state MDPs）。目前为止，我们只考虑有限状态和动作空间的马尔可夫决策过程，即状态和动作的个数都是有限的， $|S| < \infty, |A| < \infty$ 。

2.1 值迭代 (value iteration)

第一种算法，称为**值迭代** (value iteration)，过程如下所述：

1. 对每个状态 s ，初始化 $V(s) := 0$ ；
2. 重复直到收敛 {

对每个状态 s ，更新规则为： $V(s) := R(s) + \max_{a \in A} \gamma \sum_{s'} P_{sa}(s') V(s')$

这个算法可以理解成，利用贝尔曼等式重复估计更新**值函数**。

在上述的算法的内部循环体中，有两种进行更新的方法。第一种，我们首先计算每个状态 s 的 $V(s)$ 的新值，然后统一用所有新值覆盖旧值。这也叫做**同步更新** (synchronous update)。在这种情况下，可以将该算法视为实现了一个“贝尔曼备份”操作符，该操作符接收值函数的当前估计值，并将其映射到新的估计值。第二种方法，就是按照某种次序遍历所有的状态，每计算一个状态，就更新该状态的值函数。这种方法也称为**异步更新** (asynchronous update)。值迭代算法收敛后的策略就是最佳策略。

2.2 策略迭代 (policy iteration)

除了值迭代算法外，还有另外一种标准算法可以用来在马尔可夫决策过程 (MDP) 中寻找一个最佳策略，称为**策略迭代** (policy iteration)，算法过程如下：

1. 随机初始化 π ；
2. 重复直到收敛 {

 - 令 $V := V^\pi$ ；
 - 对每个状态 s ，令 $\pi(s) = \arg \max_{a \in A} \sum_{s'} P_{sa}(s') V(s')$ ；

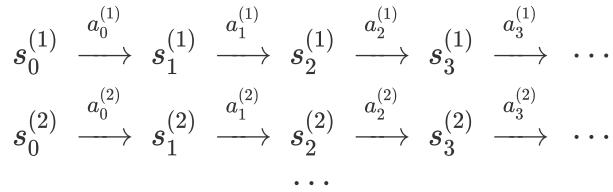
因此，在循环体内就重复计算对于当前策略的值函数；然后使用当前的值函数进行计算，选择值最大的动作来更新策略。在上面的算法迭代了某个最大迭代次数后， V 将会收敛到 V^* ，而 π 将会收敛到 π^* 。

值迭代和策略迭代都是结局马尔可夫决策过程问题的标准算法，但目前对于这两个算法哪个更好，还没有一个统一的意见。对小规模的 MDPs（即状态空间较少）来说，**策略迭代通常很快**，很少的迭代次数就会收敛。然而，对于大规模状态空间的 MDPs，确切求解 V^π 常常涉及到求解一个非常大的线性方程组，可能非常困难。对于这种问题，就更倾向于**选择值迭代**。因此，在实际使用中，值迭代通常比策略迭代更加常用。

3. 马尔可夫决策过程的自学习模型

目前为止，我们讲的 MDPs 以及用于 MDPs 的一些算法，都是基于一个假设，即状态转移概率 P_{sa} 和奖励函数 R 已知。然而在很多显示问题中，却未必知道这两样，而是必须从数据中对其进行估计。（通常 S, A, γ 都是已知的）

例如，对于倒立摆问题，我们在这个 MDP 问题中进行了若干次实验，实验过程如下：



其中， $s_i^{(j)}$ 表示的是第 j 次实验中第 i 次的状态，而 $a_i^{(j)}$ 是该状态下的动作。在实践中，每个实验都会运行到 MDP 过程停止，或者会运行到某个大但有限的步数。

有了在 MDP 中一系列实验得到的 "经验"，就可以对状态转移概率推导出最大似然估计了：

$$P_{(sa)}(s') = \frac{\text{\#times we took action } a \text{ in state } s \text{ and go to } s'}{\text{\#times we took action } a \text{ in state } s}$$

$$= \frac{\text{在状态 } s \text{ 执行动作 } a \text{ 而到达状态 } s' \text{ 的次数}}{\text{在状态 } s \text{ 执行动作 } a \text{ 的总次数}}$$

如果，上面的分式出现了 $0/0$ 的情况，对应的现实就是在状态 s 从没有进行过动作 a ，这样我们就可以将状态转移概率简单估计为 $P_{sa}(s') = \frac{1}{|S|}$ （即把下一状态 s' 的状态空间分布估计为均匀分布）。

值得注意的是，如果在 MDP 过程中我们能获得更多经验信息（更多的观察次数），就能利用新经验来更新估计的状态转移概率，这样很有效率。具体来说，如果我们保存上式的分子和分母的计数，那么观察到更多实验的时候，就可以很简单地累加这些计数值；然后计算比例，就能更新对 P_{sa} 的估计。利用类似的程序，如果奖励函数 R 未知，我们也可以选择在状态 s 下的期望即时奖励函数 $R(s)$ 来当做是在状态 s 观测到的平均奖励函数。

构建了一个马尔可夫决策过程的自学习模型后，我们可以采用值迭代或者策略迭代的方法，利用估计的状态转移概率和奖励函数，来求解这个 MDP 问题。例如，结合模型学习和值迭代，就可以在未知状态转移概率的情况下对 MDP 进行学习，下面就是一种可行的算法：

1. 随机初始化 π ；
2. 重复 {
 - 在 MDP 中执行 π 作为若干次实验。
 - 利用上面在 MDP 积累的经验，更新对状态转移概率 P_{sa} 的估计（如果可以的话，也对奖励函数 R 进行更新）；
 - 利用估计的状态转移概率和奖励函数，应用值迭代，得到一个新的估计的值函数 V ；
 - 更新 π 为与 V 对应的贪婪策略。
}

我们注意到，对于这个特定的算法，有一种简单的优化方法可以让该算法运行得更快。具体来说，在我们应用值迭代的内部循环中，如果不使用 $V = 0$ 初始化迭代，而是使用在我们的算法的前一次迭代中找到的值来初始化它，那么就提供了一个更好的迭代起点，能让算法更快收敛。

4. 备注

事实上，在机器学习中，根据马尔可夫性（即无后效性），有多种马尔可夫子模型，整理如下：

	不考虑动作	考虑动作
状态完全可见	马尔可夫链 (MC)	马尔可夫决策过程 (MDP)
状态部分可见	隐马尔可夫模型 (HMM)	部分可观察马尔可夫决策过程 (POMDP)