文本上的算法 v1.1

路彦雄 (@yanxionglu)

yanxionglu@gmail.com

尽请期待...

序：《文本上的算法》主要分两大部分：第一部分是理论篇，主要介绍机器学习的基础和一些具体算法；第二部分应用篇，主要是一些 NLP 的应用，比如：搜索引擎原理是什么？它为什么要建索引？有什么理论基础吗？之所以抽时间整理出这个文档，是由于以下方面的考虑：1、这些都是我个人曾经学习时的笔记和理解，有手写的，有电子版的，零零散散，所以想整理成一个稍微正式一点的文档，方便查阅；2、每个人有每个人的学习习惯，我自己学习一个新知识的时候，就会参考各种文献，因为每个作者表达的侧重点不同，参考的越多，理解的越深刻，所以这个文档也可以当作读者的一种参考。3、除了必不可少的公式外，尽量以更口语化的方式表达出来，抛弃掉繁琐的证明，提取出算法的核心。由于本人水平有限，难免会有一些错误，希望大家不吝指出。

理论篇

第一章、你必须知道的一些基本知识

要想明白机器学习，一些概率论和信息论的基本知识一定要知道，本章就简单的回顾下这些知识（本章可跳过阅读）。本文如不特殊声明，无下标的变量（例如 x）均为向量，有下标的（例如x_i）均为标量。

1.1 概率论

概率就是描述一个事件发生的可能性。我们生活中绝大多数事件都是不确定的，每一件事情的发生都有一定的概率（确定的事件就是

100%的概率），天气预报说明天有雨，那么它也只是说明天下雨的概率很大。再比如：掷骰子，我把一个骰子掷出去，问某一个面朝上的概率是多少？在骰子没有做任何手脚的情况下，直觉告诉你任何一个面朝上的概率都是 1/6，如果你只掷几次很难得出这个结论，但是如果你掷上 1 万次或更多，那么必然可以得出任何一个面朝上的概率都是 1/6 的结论，这就是大数定理：当试验次数（样本）足够多的时候，事件出现的频率无限接近于该事件真实发生的概率。

假如我们用概率函数$p(x)$来表示随机变量$x \in X$的概率，那么就要满足如下两个特性：

$$0 \leq p(x) \leq 1$$

$$\sum_{x \in X} p(x) = 1$$

联合概率$p(x,y)$表示两个事件共同发生的概率。假如这两个事件相互独立，那么就有联合概率$p(x,y) = p(x)p(y)$。

条件概率$p(y|x)$是指在已知事件x发生的情况下，事件y发生的概率，且有：$p(y|x) = p(x,y)/p(x)$。如果这两个事件相互独立，那么$p(y|x)$与$p(y)$相等。

联合概率和条件概率分别对应两个模型：生成模型和判别模型，这两个模型将在下一章中解释。

概率分布的均值称为**期望**，定义如下：

$$E[X] = \sum_{x \in X} x\, p(x)$$

期望就是对每个可能的取值x，与其对应的概率值p(x)，进行相乘求和。假如一个随机变量的概率分布是均匀分布，那么它的期望就等于均值，因为它的概率分布$p(x) = 1/N$。

概率分布的**方差**定义如下：

$$\mathrm{Var}[X] = \sum_{x \in X} (x - E[x])^2 \, p(x) = E[(X - E[X])^2]$$

可以看出，方差是表示随机变量偏离期望的大小，所以它是衡量数据的波动性，方差越小表示数据越稳定，反之方差越大表示数据的波动性越大。

另外，你还需要知道的几个常用的概率分布：均匀分布、正态分布、二项分布、泊松分布、指数分布等等，你还可以了解下矩阵的东西，因为所有公式都可以表示成矩阵形式。

1.2 信息论

假如一个朋友告诉你外面下雨了，你也许觉得不怎么新奇，因为下雨是很平常的一件事情，但是如果他告诉你他见到外星人了，那么你就会觉得很好奇，真的吗？外星人长什么样？同样两条信息，一条信息量很少，一条信息量很大，很有价值，那么这个价值怎么量化呢？就需要信息熵，一个随机变量X的**信息熵**定义如下：

$$H(X) = -\sum_{x \in X} p(x) \log p(x)$$

信息越少，事件（变量）的不确定性越大，它的信息熵也就越大，需要搞明白该事件所需要的额外信息就越大，也就是说搞清楚小概率事件所需要额外的信息就越大，比如说，为什么大多数人愿意相信专家

的话，因为专家在他专注的领域了解的知识（信息量）多，所以他对某事件的看法越透彻，不确定性就越小，那么他所传达出来的信息量就越大，听众搞明白该事件所需要的额外信息就越少。不过，在这个利益至上的社会，有时对专家说的话也只能呵呵了。总之，记住一句话：**信息熵表示的是不确定性的度量。**信息熵越大，不确定性越大。

联合熵的定义为：

$$H(X,Y) = -\sum_{x\in X, y\in Y} p(x,y)\ \log p(x,y)$$

联合熵描述的是一对随机变量X和Y的不确定性。

条件熵的定义为：

$$H(Y|X) = -\sum_{x\in X, y\in Y} p(x,y)\ \log p(y|x)$$

条件熵衡量的是在一个随机变量X已知的情况下，另一个随机变量Y的不确定性。

两个随机变量X和Y，它们的**互信息**定义为：

$$I(X;Y) = \sum_{x\in X, y\in Y} p(x,y)\ \log \frac{p(x,y)}{p(x)p(y)}$$

互信息是衡量两个随机变量（事件）的相关程度，当X和Y完全相关时，它们的互信息就是 1；反之，当X和Y完全无关时，它们的互信息就是 0。

互信息和熵有如下关系：

$$I(X;Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

相对熵（又叫交叉熵或者 KL 距离）的定义如下：

$$D(P||Q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

相对熵是衡量相同事件空间里两个概率分布（函数）的差异程度（不同于前面的熵和互信息，它们衡量的是随机变量的关系），当两个概率分布完全相同时，他们的相对熵就为 0，当他们的差异增加时，相对熵就会增加。相对熵又叫 KL 距离，但是它不满足距离定义的三个条件中的两个：1、非负性（满足）；2、对称性（不满足）；3、三角不等式 （不满足）。

好了，介绍了这么多概念公式，那么我们来个实际的例子，在文本处理中，有个很重要的数据就是词的互信息，上面说了，互信息是衡量两个随机变量（事件）的相关程度，那么词的互信息，就是衡量两个词的相关程度，比如，“计算机”和“硬件”的互信息就比“计算机”和“杯子”的互信息要大，因为它们更相关。那么如何在大量的语料下统计出词与词的互信息呢？公式中可以看到需要计算三个值：$p(x)$、$p(y)$和$p(x,y)$，它们分别表示x独立出现的概率，y独立出现的概率，x 和 y同时出现的概率。前两个很容易计算，直接统计下词频然后除以总词数就知道了，最后一个也很容易，统计一下x 和 y同时出现（有的会限定一个窗口）的频率除以所有无序对的个数就可以了。这样，词的互信息就计算出来了，这种统计最适合使用 Map-Reduce 来计算。

1.3 贝叶斯法则

贝叶斯法则是概率论的一部分，之所以单独拿出来写，是因为它真的很重要。它是托马斯•贝叶斯生前在《机遇理论中一个问题的解》中提出的一个当时叫“逆概率”问题，贝叶斯逝世后，由他的一个朋友替他发表了该论文，后来在这一理论基础上，逐渐形成了贝叶斯学派。

贝叶斯法则的定义如下：

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

p(x|y)称为后验概率，p(y|x)称为似然概率，p(x)称为先验概率，p(y)一般称为标准化常量。也就是说，后验概率可以用似然概率和先验概率来表示。就这个公式，非常非常有用，很多模型的基础就是它，比如：贝叶斯模型估计、机器翻译、Query 纠错、搜索引擎等等，在以后的章节中，大家经常会看到这个公式。

好了，这个公式看着这么简单，到底能有多大作用呢？那我们先拿中文分词来说说这个公式如何应用的。

中文分词在中文自然语言处理中可以算是最低层，最基本的一个技术了，因为几乎所有的文本处理任务都要首先经过分词这步操作，那么到底要怎么对一句话分词呢？最简单的方法就是查字典，如果这个词在词典中出现了，那么就是一个词，当然，查字典要有一些策略，最常用的就是最大匹配法，最大匹配法是怎么回事呢？举个例子来说，比如要对“中国地图”来分词，先拿“中”去查字典，发现“中”在字典里（单个词肯定在字典里），这时肯定不能返回，要接着查，“中国”也在字典里，然后再查“中国地”，发现没在字典里，那么“中

国”就是一个词了；然后同样的处理剩下的句子。所以，最大匹配法就是匹配最长在字典中出现的词。查字典法有两种：正向最大匹配法和反向最大匹配法，一个是从前向后匹配，一个是从后向前匹配。但是查字典法会遇到一个自然语言处理中很棘手的问题：歧义问题，如何解决歧义问题呢？

我们就以“学历史知识”为例来说明，使用正向最大匹配法，我们把“学历史知识”从头到尾扫描匹配一遍，就被分成了“学历\史\知识”，很显然，这种分词不是我们想要的；但是如果我们使用反向最大匹配法从尾到头扫描匹配一遍，那就会分成“学/历史/知识”，这才是我们想要的分词结果。可以看出查字典法可以用来分词，就会存在二义性，一种解决办法就是分别从头到尾和从尾到头匹配，在这个例子中，我们分别从头到尾和从尾到头匹配后将得到“学历\史\知识”和“学/历史/知识”，很显然，这两个分词都有“知识”，那么说明“知识”是正确的分词，然后就看“学历\史”和“学/历史”哪个是正确的，从我们人的角度看，很自然想到“学/历史”是正确的，为什么呢？因为 1、在“学历\史”中“史”这个词单独出现的几率很小，在现实中我们几乎不会单独使用这个词；2、“学历”和“史”同时出现的概率也要小于“学”和“历史”同时出现的概率，所以“学/历史”这种分词将会胜出。这只是我们大脑的猜测，有什么数学方法证明呢？有，那就是基于统计概率模型。

我们的数学模型表示如下：假设用户输入的句子用S表示，把S分词后可能的结果表示为A：A_1，A_2，…，A_k（A_i表示词），那么我们就

是求条件概率$p(A|S)$达到最大值的那个分词结果，这个概率不好求出，这时贝叶斯法则就用上派场了，根据贝叶斯公式改写为:

$$p(A|S) = \frac{p(S|A)p(A)}{p(S)}$$

显然， $p(S)$是一个常数，那么公式相当于改写成：

$$p(A|S) \propto p(S|A)p(A)$$

其中，$p(S|A)$表示(A)这种分词生成句子S的可能性；$p(A)$表示(A)这种分词本身的可能性。

下面的事情就很简单了，对于每种分词计算一下$p(S|A)p(A)$这个值，然后取最大的，得到的就是最靠谱的分词结果。比如“学历史知识”（用S表示）可以分为如下两种（当然，实际情况就不止两种情况了）：“学历\史\知识”（用A表示，A_1=学历，A_2=史，A_3=知识）和“学/历史/知识”（用B表示，B_1=学，B_2=历史，B_3=知识），那么我们分别计算一下$p(S|A)p(A)$和$p(S|B)p(B)$，哪个大，说明哪个就是好的分词结果。

但是$p(S|A_1, A_2, \dots A_k)p(A_1, A_2, \dots A_k)$这个公式并不是很好计算，$p(S|A_1, A_2, \dots A_k)$可以认为就是 1，因为由$A_1, A_2, \dots A_k$必然能生成S, 那么剩下就是如何计算$p(A_1, A_2, \dots A_k)$？

在数学中，要想简化数学模型，那就是假设。我们假设句子中一个词的出现概率只依赖于它前面的那个词(当然可以依赖于它前面的m个词），这样，根据全概率公式：

$$p(A_1, A_2, \dots A_k) =$$

$$P(A_1)P(A_2|A_1)P(A_3|A_2, A_1) \dots P(A_k|A_1, A_1, \dots, A_{k-1})$$

就可以改写为：

$$p(A_1, A_2, \dots A_k) = P(A_1)P(A_2|A_1)P(A_3|A_2) \dots P(A_k|A_{k-1})$$

接下来的问题就是如何估计$P(A_i|A_{i-1})$。然而，$P(A_i|A_{i-1})=P(A_{i-1}, A_i)/P(A_{i-1})$，这个问题变得很简单，只要数一数这对词（$A_{i-1}, A_i$）在统计的文本中前后相邻出现了多少次，以及$A_{i-1}$本身在同样的文本中出现了多少次，然后用两个数一除就可以了。

上面计算$p(A_1, A_2, \dots A_k)$的过程其实就是**统计语言模型**，然而真正在计算语言模型的时候要对公式进行平滑操作。Zipf 定律指出：一个单词出现的频率与它在频率表里的排名（按频率从大到小）成反比。这说明对于语言中的大多数词，它们在语料中的出现是稀疏的，数据稀疏会导致所估计的分布不可靠，更严重的是会出现零概率问题，因为$P(A_{i-1}, A_i)$的值有可能为 0，这样整个公式的得分就为 0，而这种情况是很不公平的，所以平滑就是解决这种零概率问题。具体的平滑算法，读者可以参考下论文：An empirical study of smoothing techniques for language modeling。

然而在实际系统中，由于性能等因素，很少使用语言模型来分词消歧，而是使用标注、共现等的方法以及一些策略来消歧。

第二章、我们生活在一个寻求最优的世界里

金庸小说里一般一个人的内功越高他的武功就越高，练了易筋经之后，随便练点什么功夫都能成为高手。同样的，最优化模型就是机器学习的内功，几乎每个机器学习模型背后都是一个最优化模型。这章讲解最优化模型。

2.1 最优化问题

通常人们会认为商人最精明，因为他们总是希望付出最小的成本来获得最大的收益。其实，我们每个人都生活在一个寻求最优的世界里，因为人心是贪婪的，人们永远想得到他们认为最好的东西。我们买东西的时候，是不希望花尽可能少的钱买到质量更好的物品？我们从一个地方去到另一个地方，是不希望尽可能走最捷径的路线？

科学抽象于生活，科学服务于生活。所以，每个机器学习背后都是一个最优化问题。一般的最优化形式表示如下：

$$\min \quad f(x)$$

$$\text{s.t.} \quad h(x) = 0$$

$$g(x) \leq 0$$

f(x)是目标函数，h(x)和g(x)分别是约束条件，有的问题可以没有约束条件（只有f(x)，称为无约束优化；只有f(x)和h(x)称为有等式约束优化；f(x)和h(x)、g(x)都有称为有不等式约束优化）。

那么目标函数到底是什么呢？

来了一个机器学习问题以后，肯定会有一个真实模型可以解决它，但是我们并不知道这个真实模型是什么（如果能知道的话，那就不用学习了，直接用就可以了），那么就要设计一个模型来代替真实模型（假设$h = f(x)$为你设计的模型，$Y = R(x)$为真实模型，$x = \{x_1, .., x_N\}$为整个模型的输入），那么怎么才能说你设计的这个模型很好呢？很简单，只要你设计的模型和真实模型的误差越小，那就说明你的模型越好，误差通常使用损失函数来表示，常用的有以下几种：

平方损失：$L\big(y, f(x)\big) = (y - f(x))^2$

绝对损失：$L(y, f(x)) = |y - f(x)|$

似然损失：$L(y, f(x)) = -\log P(y|x)$

似然损失的最小化，就是求$\log P(y|x)$的最大化，这就是后面专门要说的最大似然估计。

而损失函数（误差）的期望，称为期望风险，学习的目标就是使期望风险最小，即（M 为样本数）：

$$\min \frac{1}{M} \sum_{i=1}^{M} L\left(y^i, f(x^i, \theta)\right)$$

上面不是说真实模型是不知道的么？那么它们的期望风险自然也没法计算了，怎么最小化这个期望风险啊？期望风险是指你设计的模型和真实模型的期望误差，不知道真实模型这个自然求不出来了；虽然我们不知道真实模型$y = R(x)$是什么，但是如果我们知道所有的输入x和它对应的输出y的话，那么我们也没必要知道这个模型R是什么了，因为你给任何一个输入x我都可以给你计算一个最优的y（显然也不可能得到，能得到的话，也没必要求模型了）。那好，那我们找一些输入x'（它肯定是x的子集），然后用人工的笨办法把x'的所有最优y'算出来（$D = \{x', y'\}$称为样本对），这样，我们在计算期望风险的时候，就可以用计算好的y'直接替代真实模型$y = R(x)$就可以了，用这种方法计算出来的风险就是经验风险，根据大数定理，当样本对趋于无穷大时，经验风险也就越接近期望风险。所以，我们就可以用**经验风险最小化**来估计期望风险。

但是，我们的样本对有限，就导致经验风险估计期望风险并不理想，会产生过拟合现象。过拟合现象就是你把样本数据拟合的太完美，

也可以说是模型复杂度很高，然后到未知数据中却拟合的很差（这种对未知数据的预测能力叫做泛化能力），相反，欠拟合现象就是在样本数据上拟合的不好，在未知数据上也不好。所以，为了尽可能避免过拟合现象的出现，就要对模型的复杂度进行惩罚，这就是正则化，一般正则化，就是对模型的参数进行惩罚。这样，就相当于目标函数变成了：

$$\min \frac{1}{M}\sum_{i=1}^{M} L\left(y^i, f\left(x^i, \theta\right)\right) + \gamma J(f)$$

这也叫**结构风险最小化**。正则化公式可以有很多种，比如，L_0范数、L_1范数、L_2范数等，例如下面的正则化公式：

$$J(f) = \frac{1}{2}\,||\theta||^2 = \frac{1}{2}\sum_{i=1}^{N} {\theta_i}^2$$

现在还有个问题，对于一个优化问题，我的模型可以有好多种选择，最简单的比如$f(x,\theta)$中θ选的不同，那么最终结果就不同，如何确定一个好的模型呢？那就需要交叉验证。

交叉验证就是随机的把样本数据分成：训练集、验证集。首先在训练集中训练出各种模型$f_1(x,\theta), f_2(x,\theta), \ldots$，然后在验证集上评价各个模型的误差，选出一个误差最小的模型就是好的模型。在这儿，就要解释两个概念：偏差和方差。偏差是衡量单个模型的误差，比如：$f_1(x,\theta)$这个模型的偏差就可以用$L_1 = (y - f_1(x,\theta))^2$来表示，$f_2(x,\theta)$这个模型的偏差可以用$L_2 = (y - f_2(x,\theta))^2$来表示，所以偏差是衡量单个模型自身的好坏，它并不管别的模型怎么样；而方差是用来多个模型间比较，他并不管自己这个模型和真实模型的误差多大，而是从别的模型来衡

量自己的好坏，也就是它认为所有模型的平均值，就可以代表真实模型（这也有个潜在假设：大多数情况是正常无噪声的，否则平均值也代表不了真实模型），那么它和这个平均值比较就可以了，比如：$f_1(x,\theta)$这个模型的方差就可以用$(f_1(x,\theta)-(L_1+L_2)/2)^2$来表示。从这儿，就可以得出一些结论，一个模型越复杂，偏差就越小，方差就越大；相反，一个模型越简单，偏差就越大，方差就越小，这两个概念就是一个博弈的过程，最好的模型就是偏差和方差之和最优的模型。

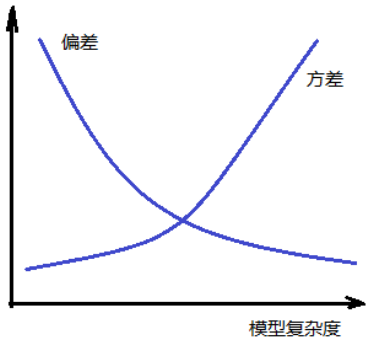

图 2.1

这就是最优化模型，你需要根据实际问题设计一个模型，设计出它的目标函数，然后可以根据交叉验证选个最好的模型（如果你的数据较好，这步有时可以省略）。

2.2 最大似然估计 /最大后验估计

上面的讲解中可以看出，对一个最优化问题，我们首先要选定模型$h=f(x,\theta)$，但是这个模型h会有无穷多个选择，到底要选哪个呢？如果你现在要急用钱，那你首先想到的是找家人或者朋友去借钱，总不能找奥巴马去借吧；同样，选模型我们也是优先选择我们熟悉的模型，比如：线性模型，高斯模型等，因为这些模型我们研究的已经很

透彻了，只需根据样本数据代入公式就可以求解出它们的参数θ，这就是参数估计；如果对一无所知的模型估计参数，那就是非参数估计。

参数估计也有很多种方法，最常用的就是最小二乘法、最大似然估计和贝叶斯估计等。

最大似然估计和贝叶斯估计分别代表了频率派和贝叶斯派的观点。频率派认为，参数是客观存在的，只是未知而已，因此，频率派最关心最大似然函数，只要参数求出来了，给定自变量 x，y 也就知道了。假设我们观察的变量是 x，观察的变量取值（样本）为$x = \{x_1, .., x_N\}$，要估计的模型参数是θ，x 的分布函数是$p(x|\theta)$。那么最大似然函数就是θ的一个估计值，它使得事件发生的可能性最大，即：

$$\theta_{MLE} = \text{argmax}_\theta p(x|\theta)$$

通常，我们认为 x 是独立同分布的，即有：

$$p(x|\theta) = \prod_{i=1}^{N} p(x_i|\theta)$$

由于连乘会可能造成浮点下溢，所以通常就最大化对数形式，也就是：

$$\theta_{MLE} = \text{argmax}_\theta \{\sum_{i=1}^{N} \log p(x_i|\theta)\}$$

所以最大似然估计的一般求解流程就是：

（1）写出似然函数：$\mathcal{L}(\theta) = p(x|\theta) = \prod_{i=1}^{N} p(x_i|\theta)$。

（2）对似然函数取 log：$L(\theta) = \log\mathcal{L}(\theta) = \sum_{i=1}^{N} \log p(x_i|\theta)$。

（3）求$\text{argmax}_\theta L(\theta)$：对$L(\theta)$求导，令其为零，解出θ。

最大似然估计中θ是固定的一个值，只要这个θ能很好的拟合样本x就是好的，前面说了，它拟合样本数据很好，不一定拟合未知数据就很好（过拟合现象）。所以用频率派的理论可以得出很多扭曲事实的

结论：只要我没看到过飞机相撞，那么飞机永远就不可能相撞。这时，贝叶斯学派就开始说了，参数θ也应该是随机变量（$p(\theta)$），和一般随机变量没有本质区别，它也有概率（θ取不同值的概率），也就是尽管我没看到飞机相撞，但是飞机还是有一定概率可能相撞，正是因为参数不能固定，当给定一个输入 x 后，我们不能用一个确定的 y 表示输出结果，必须用一个概率的方式表达出来。所以，我们希望知道所有θ在获得观察数据x后的分布情况，也就是后验概率$p(\theta|x)$，根据贝叶斯公式我们有：

$$p(\theta|x) = \frac{p(x|\theta)p(\theta)}{p(x)} = \frac{p(x|\theta)p(\theta)}{\int p(x|\theta)p(\theta)d\theta}$$

可惜的是，上面的后验概率通常是很难计算的，因为要对所有的参数进行积分，而且，这个积分其实就是所有θ的后验概率的汇总，其实它是与最优θ是无关的，而我们只关心最优θ。在这种情况下，我们采用了一种近似的方法求后验概率，这就是最大后验估计：

$$\theta_{MAP} = \text{argmax}_\theta p(\theta|x) = \text{argmax}_\theta p(x|\theta)p(\theta)$$

最大后验估计相比最大似然估计，只是多了一项先验概率，它正好体现了贝叶斯认为参数也是随机变量的观点，在实际运算中通常通过超参数给出先验分布。**最大似然估计其实是经验风险最小化的一个例子，而最大后验估计是结构风险最小化的一个例子。**如果样本数据足够大，最大后验概率和最大似然估计趋向于一致，如果样本数据为 0，最大后验就仅由先验概率决定，就像推荐系统中对于一个毫无历史数据的用户，只能给他推荐热门（先验概率高）的内容了。尽管最大后验估计看着要比最大似然估计完善，但是由于最大似然估计简单，很

多方法还是使用最大似然估计，也就是频率派和贝叶斯派谁也没把谁取代掉。

最大似然估计和最大后验估计已经懂了，那么顺带说下最大二乘估计。对于最小二乘法估计，当从模型总体随机抽取 M 组样本观测值后，最合理的参数估计值应该使得模型能最好地拟合样本数据，也就是估计值和观测值之差的平方和最小。而对于最大似然估计，当从模型总体随机抽取 M 组样本观测值后，最合理的参数估计值应该使得从模型中抽取该 M 组样本观测值的概率最大。显然，这是从不同原理出发的两种参数估计方法。而且，最小二乘估计有个假设：模型服从高斯分布。

现在我们已经知道如何构造最优化问题的目标函数了，以及进行参数估计的一些方法，剩下的问题就是如何具体的求解参数了？

2.3 梯度下降法

大多数最优化问题（凸规划）是有全局最优解的（下面左图），而有的最优化问题只有局部最优解，当然局部最优解有可能就是全局最优解，但是不容易使得得到的局部最优解就正好是全局最优解（下面右图）。求解最优化问题，本质上就是怎么向最优解移动，达到某个条件（收敛）就停止。

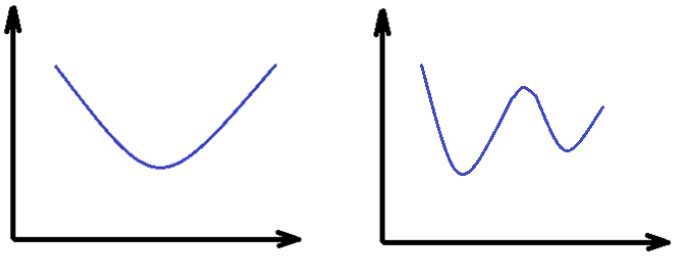

图 2.2

有的最优化问题有解析解，可以直接求解，那么怎么求解呢？直接对其求导数，然后令导数为零，就可以解出候选最优解了。

有的最优化问题没有解析解，只能通过一些启发式算法（遗传算法等）或者数值计算的方法来求解。对于无约束优化问题常用的算法大概有：梯度下降法，牛顿法/拟牛顿法，共轭梯度法，坐标下降法等（这些方法使用导数，还有些算法是不使用导数的）。而对于有约束优化问题大多是通过拉格朗日乘子法转换成无约束问题来求解。

本节就以线性回归模型来看下梯度下降法到底是怎么回事？假设有如下表示的线性函数：

$$f(x,\theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots = \sum_{i=0}^{N} \theta_i x_i = \theta^T x$$

如果它的损失函数使用平方损失函数，则有（没有正则化）：

$$L(\theta) = \frac{1}{2}\sum_{j=1}^{M}\left(f\left(x^j,\theta\right) - y^j\right)^2$$ （M 为样本数）

梯度下降法指出，**函数f在某点x沿着梯度相反的方向下降最快**，也就是说我从某点x出发，沿着梯度相反的方向移动，就可以走到最优解了。梯度是什么呢？在这儿就是导数。所以它的每一步的迭代公式就是：

$$\theta_i = \theta_i + \alpha\left(-\frac{\partial}{\partial\theta_i}L(\theta)\right)$$

$$= \theta_i - \alpha\frac{\partial}{\partial\theta_i}L(\theta)$$

其中α是个定值，就是学习速度，控制每步移动的幅度。

又有：

$$\frac{\partial}{\partial\theta_i}L(\theta)=\sum_{j=1}^{M}\left(y^j-f(x^j,\theta)\right)x_i{}^j$$

所以最终的迭代公式就是：

$$\theta_i=\theta_i+\alpha\sum_{j=1}^{M}\left(y^j-f(x^j,\theta)\right)x_i{}^j$$

梯度下降法的流程就是：

while (直到收敛)：

$$\theta_i=\theta_i+\alpha\sum_{j=1}^{M}\left(y^j-f(x^j,\theta)\right)x_i{}^j\ (i=0..N)$$

这就是梯度下降法的迭代流程，首先任意选定一个θ，然后使用公式迭代，一直到收敛（θ的变化小于一个阈值）停止，就解出了参数θ。

可以看出，公式中有个求和，也就是每次迭代都需要计算全部样本，所以当样本M很大时，计算代价很大，那么就需要考虑个好的办法减少计算，这就是随机梯度下降法。

随机梯度下降法并不计算梯度的精确值，而是计算一个估计值，也就是每次迭代都是基于一个样本，迭代流程就成为：

for j=1 to M：

$$\theta_i=\theta_i+\alpha\left(y^j-f(x^j,\theta)\right)x_i{}^j\ (i=0..N)$$

这个公式有什么好处呢？它可以并行计算，因为样本间是无关的。所以当样本特别大时，可以使用随机梯度下降法，它的缺点是通常找到的最小值是最优解的近似值。当样本数较小时，一般也不用梯度下降法，因为它收敛太慢了（尤其接近最优解的时候），而是使用 LBFGS 或者 CG 等来训练。

在这儿要记住一个观点：1、要想得到最优解就要付出更大的代价。就像生活中，吊儿郎当的人活得会比较轻松，因为他做事不需要最优；谨慎细致的人就活得比较累，每件事情都要做到最好，不同性格的人有不同的生活态度，没有对错。2、好多问题没有最优解或者最优解无法求出，那么就要用近似的方法来求解出近似最优解。生活中好多事情压根没办法做到十全十美，所以我们只能尽自己最大努力达到尽可能好。

梯度下降法是最简单且很好理解的一个算法，理解了这个算法，其他求解算法（牛顿法/拟牛顿法，共轭梯度法，坐标下降法等）就较容易了。

对于无约束优化问题我们已经知道一些求解算法了，那么还有两类问题：等式约束优化问题和不等式优化问题。

对于等式约束优化问题：

$$\min \quad f(x)$$

$$s.t. \;\; h(x) = 0$$

写出它的拉格朗日乘子法：

$$L(x, \alpha, \beta) = f(x) + \alpha h(x)$$

之后的求解方法就和求解析解一样了，用$L(x, \alpha, \beta)$分别对x, α, β求导数，然后令它们的导数为零，就可以分别解出x, α, β的候选最优解了。

对于不等式约束优化问题：

$$\min \quad f(x)$$

$$s.t. \;\; h(x) = 0$$

$$g(x) \leq 0$$

写出它的拉格朗日乘子法：

$$L(x,\alpha,\beta) = f(x) + \alpha h(x) + \beta g(x) \quad (\beta \geq 0)$$

但是上式要想有和原不等式约束优化问题一样的最优解，必须满足 KKT 条件：（1）$L(x,\alpha,\beta)$分别对x求导为零；（2）$\beta g(x) = 0$；（3）$g(x) \leq 0$；（4）$\beta \geq 0$，$\alpha \neq 0$；（5）$h(x) = 0$。

当原问题不太好解决的时候可以利用拉格朗日乘数法得到其对偶问题，满足强对偶性条件时它们的解是一致的。那么什么是对偶问题呢？还是以上述不等式约束问题为例说明，写出它的拉格朗日函数：

$$L(x,\alpha,\beta) = f(x) + \alpha h(x) + \beta g(x) \quad (\beta \geq 0)$$

然后定义一个函数：$\theta_p(x) = \max_{\alpha,\beta,\beta\geq 0} L(x,\alpha,\beta)$

这个函数是α,β的函数，如果x违反原始问题的约束条件，即$h(x) \neq 0$或者$g(x) > 0$，那么我们总是可以调整α和$\beta \geq 0$来使得$\theta_p(x)$有最大值为正无穷，而只有$h(x)$和$g(x)$都满足约束时，$\theta_p(x)$为$f(x)$，也就是说$\theta_p(x)$的取值是：$f(x)$或者 ∞。所以$\min f(x)$就转为求$\min_x \theta_p(x)$了，即：

$\min_x \theta_p(x) = \min_x \max_{\alpha,\beta,\beta\geq 0} L(x,\alpha,\beta)$ （它的最优解记为p^*）

再定义一个函数：$\theta_d(\alpha,\beta) = \min_x L(x,\alpha,\beta)$，则有：

$\max_{\alpha,\beta,\beta\geq 0} \theta_d(\alpha,\beta) = \max_{\alpha,\beta,\beta\geq 0} \min_x L(x,\alpha,\beta)$ （它的最优解记为d^*）

$\theta_p(x)$和$\theta_d(x)$互为对偶问题。可以证明：$d^* \leq p^*$，如果满足强对偶性：目标函数和所有不等式约束函数是凸函数，等式约束函数是仿射函数（形如$y = w^t x + b$），且所有不等式约束都是严格的约束，那么$d^* = p^*$，就是说原问题和对偶问题的解一致。

对偶原理告诉我们，如果$\min f(x) = \min_x \max_{\alpha,\beta,\beta\geq 0} L(x,\alpha,\beta)$不好求解，那么就求解$\max_{\alpha,\beta,\beta\geq 0} \min_x L(x,\alpha,\beta)$，后面我们可以看到它的应用。

总结一下：最优化问题其实很简单，首先需要一个模型：目标函数和约束函数（函数中的变量通常就是特征，下章解释），不同的问题会对应不同的模型，需要自己设计；然后对该模型的参数进行求解。

第三章、让机器像人一样可以学习

再讲机器学习之前，先要灌输三个名词：训练数据、特征、模型。它们就是机器学习的核心，模型就是上一章讲解过的。

3.1 何为机器学习

我们首先看看人是如何学习的。

小明（万能主角出场）在校园里看到两排东西（一排图 3.1，一排图 3.2），但他不认识它们是什么，于是他就去问同学，同学瞥了一眼就说：左边的是汽车；右边的是摩托车。

图 3.1

图 3.2

小明仔细观察了下，左边这个叫汽车的比较大，而且有四个轮子；右边叫摩托车的比较小，而且只有两个轮子。于是小明的大脑中就形成了这么一个概念（模型一）：

$$\text{if}(\text{大，而且有四个轮子})\ \text{then}\ \text{它是汽车}$$

$$\text{if}(\text{小，而且有两个轮子})\ \text{then}\ \text{它是摩托车}$$

太好了，小明学会了如何区分汽车和摩托车，然后高高兴兴的回家了，但是在路上，他见到了图 3.3 这么个东西，根据他之前学到的知识，它应该就是摩托车，于是他就说到：这是个摩托车。路人甲听到了，说：小朋友，这不是摩托车，这是自行车。

图 3.3

小明很沮丧，但是他没有放弃，他回家仔细想了想：如果我有更多可参考的汽车和摩托车的样例，然后提取出更多他们的特点，那么我一定能区分的很好。没错！于是小明总结了更多的不同点，最后他大脑中就形成了这么另一个概念（模型二）：

$$\text{if}\begin{pmatrix}\text{大，}\\ \text{有四个轮子，}\\ \text{轮子大小一样，}\\ \text{好四个门，}\\ \text{轮胎较粗，}\\ \text{座位大，}\\ \text{零部件个数} > N\text{，}\\ \text{等等}\end{pmatrix}\ \text{then}\ \text{它是汽车}$$

$$\text{if}\left(\begin{array}{c}\text{小，}\\ \text{有两个轮子，}\\ \text{轮子大小一样，}\\ \text{没有门，}\\ \text{轮胎较细，}\\ \text{座位小，}\\ \text{零部件个数} < M\text{，}\\ \text{等等}\end{array}\right)\ \text{then 它是摩托车}$$

小明想通了以后，觉得这下他会区分汽车和摩托车了吧，结果有一天他见到了图 3.4，这货前后轮胎大小不一样？前面还有个豹子头？不满足摩托车条件啊？小明迷茫了！

图 3.4

于是小明跑去问他爸爸，他爸爸说，你上面那些条件限制的太死板了，有些条件是可有可无的（说白了这些条件需要惩罚），我告诉你一个较好的区分方法吧（模型三）：

$$\text{if}\left(\begin{array}{c}\text{大，}\\ \text{有至少四个轮子，}\\ \text{使用汽油，}\\ \text{其他不重要}\end{array}\right)\ \text{then 它是汽车}$$

$$\text{if}\left(\begin{array}{c}\text{小，}\\ \text{有两个轮子，}\\ \text{使用汽油，}\\ \text{其他不重要}\end{array}\right)\ \text{then 它是摩托车}$$

小明想了想，这个比他前两个都好（当然不是最好的），以后就这样区分汽车和摩托车了。小明学会了如何区分汽车和摩托车。

前面说过一句话：“科学抽象于生活，科学服务于生活。”。所以，机器学习的过程其实就是模拟人学习的过程（当然，人比机器更智能，机器智能还远达不到人的程度）。从这个例子中可以看出，“小”，“有两个轮子”，“使用汽油”等就叫**特征**，那个 if，then 就叫**模型**，小明的模型很简单，各个特征都满足才可以。图 3.1 和图 3.2 就是**训练数据**（图 3.2 下面的自行车可以理解为是个噪声数据，一般来说，噪声数据难免会有），小明的模型一很简单，即没有把训练数据分的很好，也没有把新的测试数据分的很好，这就是欠拟合现象；模型二很复杂，虽然把训练数据区分的很好，但是对新的测试数据（图 3.4）不能很好地区分，这就是过拟合现象，模型三是较理想的一个模型。

这就是机器学习，它的核心就是特征，模型，训练数据（标注数据或未标注数据），首先建立一个模型，然后抽取一些特征，最后在训练数据中把模型参数学习出来就可以了，这个是监督学习，它需要标注训练数据（还有一大类叫非监督学习，不需要标注训练数据，比如：聚类问题），前面说了，训练数据趋于无穷多时，模型训练的越好，但是现实中拿到更多的训练数据代价太大，再加上特征表示和模型本身都不会是最优的，所以机器学习一般得到的都是近似解，就像小明利用模型三也会有区分不出来的，也就是说机器学习只能解决大部分情况，而总会有些个例会解决不了。

可以看出，不同的机器学习任务就需要不同的特征和模型，有的问题模型是可以通用的（比如分类问题），但是特征却不能通用，需要

人根据不同的问题来选取，如果特征也是由机器学习出来的那该有多好，所以深度学习的一个目标就是自动学习特征，后面会讲到深度学习。

当训练出模型后，对一个输入，同样提取的特征，然后使用模型来进行预测，而这个模型$y = f(x)$一般就是条件概率分布：$p(y|x)$。

监督学习通常分为两个模型：生成模型和判别模型。

判别模型（它的概率图是无向图）是求解条件概率的$p(y|x)$，然后直接进行预测，比如：逻辑回归，SVM，神经网络，CRF 都是判别模型，所以判别模型求解的是条件概率。

生成模型（它的概率图是有向图）是首先求解两个概率$p(x|y)$和$p(y)$，然后根据贝叶斯法则求出后验概率$p(y|x)$，再进行预测，比如：朴素贝叶斯，HMM，贝叶斯网络，LDA 都是生成模型，又因为$p(x, y) = p(x|y) * p(y)$，所以生成模型求解的是联合概率。

举例来说，比如观察到一只狮子，要判断是美洲狮还是非洲狮？按照判别模型的思路，我们首先需要有一定的资料，机器学习上称为训练集，比如过去观察到的狮子的特征可以得到一个预测函数，之后把我们当前观察的狮子的一些特征提取出来，输入到预测函数中，得到一个值，就知道它是什么狮子了。而对于生成模型，我们先从所有美洲狮的特征中学习得到美洲狮的模型，同样得到非洲狮的模型，然后提取当前观察的狮子的模型，放到两个模型中，看哪个概率更大，就是什么狮子，这就是生成模型。可以看出，判别模型只需要关注类的边界就可以了，并不需要知道每一类到底是什么分布，这样它只需

要有限样本就可以搞定；而生成模型要得到每类的具体分布，然后根据每个分布去判断类别，它的训练集自然需要无限样本，学习复杂度也高。造成两个模型样本空间不同的原因在于条件概率我们“已经知道”了一部分信息，这部分已经知道的信息缩小了可能取值的范围，即缩小了它的样本空间。

由生成模型可以得到判别模型，但由判别模型得不到生成模型。

接下来就看些具体的机器学习算法。在这儿要提醒一下，几乎每个机器学习模型都有假设，所以具体的应用场景或者数据应该尽可能接近所使用机器学习模型的假设。

3.2 逻辑回归

现在我有个分类任务要做：判断一篇邮件是否为垃圾邮件。什么是垃圾邮件呢？总得有个定义吧，那我们姑且先把这类邮件归为垃圾邮件：里面有广告推广、有诈骗、有非法活动等的邮件。

我们先选用 2.3 节中的线性回归模型来完成这个任务，即 $y = f(x) = \theta^T x$，x为特征向量（比如：x_1表示出现广告词的次数，x_2表示是否含有电话号码，x_3表示是否含有网址链接，等等）；y为分类结果，y=1 表示为垃圾邮件，y=0 表示为非垃圾邮件。那么对于这个二分类问题就可以设置个阈值来判断：

$$y = \begin{cases} 1, & \text{if } \theta^T x \geq 0.5 \\ 0, & \text{if } \theta^T x < 0.5 \end{cases}$$

但是线性回归的输出 y的取值范围随着特征向量x的变化或者增多可以是任何数值，如果要使我们上面设定阈值的分类方法有效，必须

对线性回归的输出值映射到一个固定的范围，这就需要请出逻辑回归（Logistic Regression）。

逻辑回归在线性回归的输出y上引入了一个函数g(z)：

$$g(z) = \frac{1}{1 + e^{-z}}$$

该函数（称为 sigmoid 函数）的作用就是可以把某个值映射到 0,1 区间，它的曲线图大致如下所示：

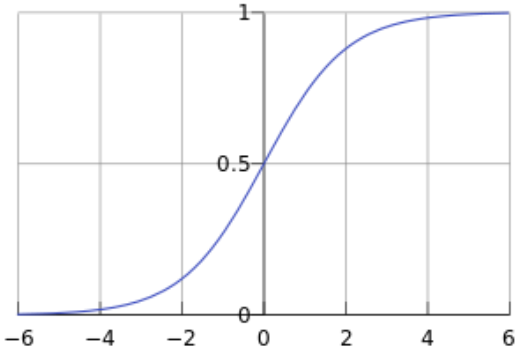

图 3.5

这样，整个逻辑回归公式就为：

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

好了，现在**特征向量**x有了，**模型**$h_\theta(x)$也有了，**数据**也很容易得到，请人标注一批数据即可，那剩下的问题就是如何求解模型的参数θ？

如果我们选用和线性回归模型一样的平方损失作为目标函数（没有正则化），即：

$$L(\theta) = \frac{1}{2}\sum_{j=1}^{M}\left(h_\theta(x^j) - y^j\right)^2$$ （M 为样本数）

那么就会有个问题，由于$h_\theta(x)$是 sigmoid 函数，导致目标函数不是凸函数（如下图所示），那么就没有最优解，所以我们就需要做些工作使得目标函数是凸函数。

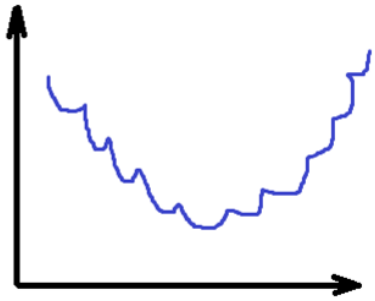

图 3.6

对于二分类问题，假设：

$$p(y=1|x,\theta)=h_\theta(x)$$

那么：

$$p(y=0|x,\theta)=1-h_\theta(x)$$

根据这两个概率，可以写出概率分布（二项分布）为：

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

这时就可以写出似然函数了：

$$L(\theta) = p(Y|X,\theta)$$

$$=\prod_{j=1}^{M} p\left(y^j|x^j,\theta\right)$$

$$=\prod_{j=1}^{M}(h_\theta(x^j))^{y^j}(1-h_\theta(x^j))^{1-y^j}$$

然后就可以使用最大似然估计来求解了。

（1）似然函数取 log：

$$L(\theta)=\log\mathcal{L}(\theta)=\sum_{j=1}^{M} y^j\log\left(h_\theta(x^j)\right)+(1-y^j)\log(1-h_\theta(x^j))$$

（2）对$L(\theta)$求导：

$$\frac{\partial}{\partial\theta_i}L(\theta)=\sum_{j=1}^{M}(y^j\frac{1}{g(\theta^Tx^j)}-(1-y^j)\frac{1}{1-g(\theta^Tx^j)})\frac{\partial}{\partial\theta_i}g(\theta^Tx^j)=$$

$$\sum_{j=1}^{M}\left(y^j\frac{1}{g(\theta^Tx^j)}-(1-y^j)\frac{1}{1-g(\theta^Tx^j)}\right)g(\theta^Tx^j)(1-g(\theta^Tx^j))\frac{\partial}{\partial\theta_i}\theta^Tx^j$$

$$=\sum_{j=1}^{M}(y^j(1-g(\theta^Tx^j))-(1-y^j)g(\theta^Tx^j))x_i{}^j$$

$$=\sum_{j=1}^{M}(y^j-h_\theta(x^j))x_i{}^j$$

这里用了一个推导公式：$g'(z)=g(z)(1-g(z))$。

（3）最后的迭代公式是：

$$\theta_i=\theta_i+\alpha\sum_{j=1}^{M}(y^j-h_\theta(x^j))x_i{}^j$$

看到了吗？它的迭代公式形式是不和前面线性回归模型的梯度下降法的迭代公式一样呢（除了$h_\theta(x)$不同）？所以逻辑回归的求解速度也是很快的。

那么逻辑回归到底有什么用呢？目前谷歌、百度等各大公司都是使用逻辑回归来对广告点击率（Click-through Rate）进行预估。

搜索广告相比传统广告效果更好，就是因为它利用了用户主动的搜索意图。任何一种广告的目的一是为了赚钱，二是为了尽可能多的赚钱，简单来说，就是最大化：流量×每千次展示收益（CPM），而每千次展示收益 = 展示之后被点击的概率(CTR) × 一次点击的收入，那么提高 CTR 就自然会提高收入，也就是把最可能被用户点击的广告

展示出来。而这个 CTR 就是使用逻辑回归计算出来的，一般每天产生的训练数据（线上用户真实点击与否的数据）将有上亿个，而且特征通常也会有近亿个，通常有三大类特征：流量特征（query 分词、主题、地域、覆盖率、页面特征等等）、广告特征（切词、专名、url、飘红、主题等等）和用户特征（用户兴趣、cookie、主题等等）。所以 CTR 的逻辑回归模型有亿级别特征和百亿级的训练数据，必须依靠分布式并行来搞定。

3.3 最大熵模型

最大熵模型是我个人比较喜欢的模型之一，它背后的原理其实非常简单：当我们对一个随机变量的分布预测时，对已知条件一定要满足外，对未知数据一无所知时，不要做任何主观假设，要同等对待。这时，它们的概率分布最均匀，风险就越小，概率分布最均匀就意味着信息熵最大，所以就叫最大熵模型。

如果不好理解的话，我们就举个例子来说。现在有一场很激烈的篮球比赛，比分是 81:82，而且比赛时间只剩最后 3 秒，球权在落后的一方，庆幸的是，你就是领先这一队的教练，现在你的任务就是暂停之后防守住对方的最后一次进攻，不让他投进，否则就输球了，而且已知对方球队的 5 个球员（A,B,C,D,E）中，A 是超级球星，所以他进行最后一投的概率非常大；E 是个防守悍将，进攻端很差，所以他最后一投的概率非常小，他的主要目的应该是掩护使得 A 能顺利投球；其他三个都是能力相当的普通球员。那么你就有很多战术来完成这次防守，其中就包括下面两种方案：（1）让你们队中防守最好的球员去防

守 A,然后让本来防守 E 的球员主要去协防 A(这样 E 几乎没人防守了),其他三个人一对一防守对方;(2)同样,让防守最好的球员去防守 A,防守 E 的球员去协防 A,剩下三个人(B,C,D)的防守同上面方案不同,你让一个球员防守 B,然后让防守 C 的球员花很大精力也去协防 B,D 是一对一防守。那么你觉得上面两个方案哪个好呢?很显然是方案(1)啊,因为方案(2)中为什么要对球员 C 减轻防守,他和球员 B、D 的能力相当啊,那么如果不是 A 最后一投的话,C 在轻防守下命中率就会增加,那么你输球的概率就自然增加了。所以对完全未知的情况不要做任何主观假设,平等对待,风险才能最小。

好,那我们开始看看那最大熵模型是怎么回事?事先我们需要介绍几个概念,假设样本集为$D = \{X, Y\}$,X 为输入,Y 为输出,比如对于文本分类问题,X 就为输入文本,Y 就是类别号;对于词性标注问题,X 就为词,Y 就是词性,等等。可以看出,在不同的问题中,输入 X 和输出 Y 比较多样化,为了模型表示方便,我们需要将输入 X 和输出 Y 表示为一些特征。对于某个(x_0, y_0),定义特征函数:

$$f(x,y) = \begin{cases} 1: y = y_0 \text{ and } x = x_0 \\ 0: \text{other} \end{cases}$$

那么特征函数的**样本期望**就可以表示为:

$$\bar{p}(f) = \sum_{x,y} \bar{p}(x,y) f(x,y)$$

而特征函数的**模型期望**表示为:

$$p(f) = \sum_{x,y} p(x,y) f(x,y) = \sum_{x,y} \bar{p}(x) p(y|x) f(x,y)$$

样本期望和模型期望到底是什么东西呢？还记得之前提到的经验风险和期望风险的区别吗？它们的区别也一样，样本期望是从样本数据中计算的，模型期望是从我们希望要求解的最优模型中计算的，而且，模型期望最终简化为条件概率$p(y|x)$，它就是我们要求解的模型，因为$\bar{p}(x)$是从样本中统计来的，根据最大似然法，数一数 x 出现的次数除以总次数就得到了，同样样本期望中的$\bar{p}(x,y)$也是数一数(x,y)同时出现的次数除以总次数就可以得到了。

机器学习就是从样本中学习到真实模型，也就是说模型期望就应该尽可能的等于从数据中观察到的样本期望，这样就出现了一个约束条件：$\bar{p}(f) = p(f)$。那么目标函数是什么呢？开头说了：要使熵最大。那么目标函数就是：$\text{argmax}_p H(p) = -\sum_{x,y} \bar{p}(x)p(y|x)\log p(y|x)$

所以最大熵模型就是（$p(y|x)$是变量）：

$$\text{argmax}_p H(p) = -\sum_{x,y} \bar{p}(x)p(y|x)\log p(y|x)$$

$$\text{s.t.} \begin{cases} \forall f_i (i = 1..k) \sum_{x,y} \bar{p}(x,y) f_i(x,y) = \sum_{x,y} \bar{p}(x)p(y|x)f_i(x,y) \\ \forall x \sum_{y} p(y|x) = 1 \end{cases}$$

可以把最大化转换为最小化问题，即：

$$\text{argmin}_p -H(p) = \sum_{x,y} \bar{p}(x)p(y|x)\log p(y|x)$$

$$\text{s.t.} \begin{cases} \forall f_i (i = 1..k) \sum_{x,y} \bar{p}(x,y) f_i(x,y) = \sum_{x,y} \bar{p}(x)p(y|x)f_i(x,y) \\ \forall x \sum_{y} p(y|x) = 1 \end{cases}$$

OK，模型构建好了，那剩下的就是如何求解模型了，这是个等式约束优化，就可以用拉格朗日乘子法来求解，写出拉格朗日函数：

$$L(p,\alpha) = -H(p) + \sum_{i=1..k} \alpha_i \left(\sum_{x,y} \bar{p}(x,y) f_i(x,y) - \sum_{x,y} \bar{p}(x) p(y|x) f_i(x,y) \right) + \alpha_0 \left(\sum_{y} p(y|x) - 1 \right)$$

不幸的是，直接对参数求导使它们等于零，没法求解出各个参数，因为参数互相耦合到一起了，所以就要尝试其他方法求解了。

还记得前面讲的对偶原理吗？原问题是：

$$\min_p \max_\alpha L(p,\alpha)$$

对偶问题是：

$$\max_\alpha \min_p L(p,\alpha)$$

由于$L(p,\alpha)$是凸函数，所以原始问题和对偶问题是等价的。这样我们就求解对偶问题，首先求解$\min_p L(p,\alpha)$。

对 p 求导，求解$\frac{\partial}{\partial p} L(p,\alpha) = 0$，即：

$$\frac{\partial}{\partial p} L(p,\alpha) = \sum_{x,y} \bar{p}(x)(\log p(y|x) + 1) - \sum_{x,y} \sum_{i=1..k} \alpha_i \bar{p}(x) f_i(x,y) + \sum_{y} \alpha_0$$

$$= \sum_{x,y} \bar{p}(x)[\log p(y|x) + 1 - \sum_{i=1..k} \alpha_i f_i(x,y) + \alpha_0]$$

解得：

$$p^*(y|x) = e^{\sum_i \alpha_i f_i(x,y) - \alpha_0 - 1} = Z(x) e^{\sum_i \alpha_i f_i(x,y)}$$

又：$\forall x \sum_y p(y|x) = 1$，那么

$$\sum_y p^*(y|x) = \sum_y Z(x) e^{\sum_i \alpha_i f_i(x,y)} = 1$$

得到：$Z(x) = \frac{1}{\sum_y e^{\sum_i \alpha_i f_i(x,y)}}$

总结一下，最大熵模型的条件概率分布为：

$$p(y|x) = Z(x)e^{\sum_i \alpha_i f_i(x,y)}$$

$$Z(x) = \frac{1}{\sum_y e^{\sum_i \alpha_i f_i(x,y)}}$$

这样$\min_p L(p,\alpha)$的最优解$p^*(y|x)$就求解出来了，剩下就是求：

$$\max_\alpha \min_p L(p,\alpha) = \max_\alpha p^*(y|x)$$

它是α的函数，只要α求解出来了，最大熵模型$p(y|x)$也就求解出来了，但是很显然，α的解析解无法直接求出（有指数函数），那么就需要数值方法来求解了，比如：GIS，IIS，LBFGS 等方法。

在这简单介绍下 IIS 求解的流程（里面有很多证明，请参考论文 The Improved Iterative Scaling Algorithm: A Gentle Introduction），我们现在已经知道了条件概率$p(y|x)$的表达式了，那么就可以用最大似然估计（最大熵和最大似然估计训练结果一致，只是考虑的方式不同：最大熵是以样本数据的熵值最大化为目标；最大似然估计是以样本数据的概率值最大化作为目标，既然训练结果一致，那就找个简单的求解），写出它的 log 似然函数：

$$L'(\alpha) = \sum_{x,y} \bar{p}(x,y)\log p(y|x)$$

$p(y|x)$我们已经求解出来了，是α的函数，那么$L'(\alpha)$自然也是α的函数了，但是如果用$L'(\alpha)$对 α求导的话，也是无法解出α的，那也就需要迭代法：$\alpha = \alpha + \delta$，满足$L'(\alpha+\delta) \geq L'(\alpha)$，也就是每步迭代都要向最优解移动，要想移动的快，也就是要使$L'(\alpha+\delta) - L'(\alpha)$尽可能大，也就是最大化$L'(\alpha+\delta) - L'(\alpha)$即可（它是$\delta_i$的函数），如果

$L'(\alpha+\delta)-L'(\alpha)$无法表示出来，那就找个下限$L'(\alpha+\delta)-L'(\alpha)\geq\varphi(\delta)$，然后就是每次迭代求解$\varphi(\delta)$的最大值了。

IIS 的算法流程如下：

（1）设$\alpha_i=0$，$i=1..k$。

（2）for i=1 to k:

2.1 $\frac{\partial}{\partial\delta}\varphi(\delta_i)=\sum_{x,y}\bar{p}(x,y)f_i(x,y)-\sum_x\bar{p}(x)\sum_y p(y|x)f_i(x,y)e^{\delta_i f^{\#}(x,y)}$

2.2 令$\frac{\partial}{\partial\delta}\varphi(\delta_i)=0$，解出$\delta_i$。（其中$f^{\#}(x,y)=\sum_{i=1..k}f_i(x,y)$）

2.3 $\alpha_i=\alpha_i+\delta_i$

至此，最大熵模型介绍完了，然后简单看下条件随机场（CRF）是什么东西。大家已经知道最大熵模型的条件概率分布为：

$$p(y|x)=Z(x)\exp(\sum_i\alpha_i f_i(x,y))$$

$$Z(x)=\frac{1}{\sum_y\exp(\sum_i\alpha_i f_i(x,y))}$$

而且，它的概率图，如下：

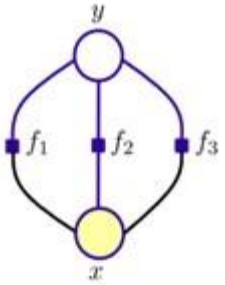

图 3.7

而 CRF 的条件概率分布是：

$$p(y|x)=Z(x)\exp(\sum_i\alpha_i\sum_{t=1..N}f_i(x_t,y_t,y_{t-1}))$$

$$Z(x)=\frac{1}{\sum_y\exp(\sum_i\alpha_i\sum_{t=1..N}f_i(x_t,y_t,y_{t-1}))}$$

而且，它的概率图如下：

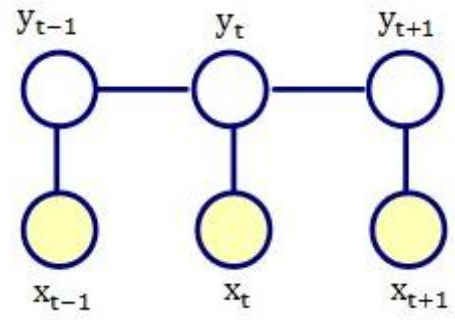

图 3.8

从概率图大致可以看出区别了，CRF 利用了上下文信息，而且最后的条件概率也是全局最优（无标记偏置问题），所以它对标注问题（分词、词性标注、实体识别等）效果更好一点，至于 CRF 的细节大家可以参考一下论文（Conditional Random Fields: An Introduction），它的参数求解过程和最大熵很类似：最大似然估计，然后使用 IIS，LBFGS 等算法求解参数。

3.4 主题模型(Topic Model)

主题模型（Topic Model）可是自然语言处理（NLP）中非常有影响力的模型之一，因为它的初衷就是解决 NLP 中的语义问题，尽管离真正意义上的语义有一定距离。那么什么是“主题”呢？一段话的主题就是小学时学到的中心思想，但是这个中心思想太泛了，计算机要怎么表示呢？那就是用一些最能反映中心思想的词来表示（这就产生了一个假设：bag of words，这个假设是指一个文档被表示为一堆单词的无序组合，不考虑语法、词序等，现在 bag of words 假设其实也是 NLP 任务的一个基本假设），比如下面一段新闻内容：“*站在互联网整个行业的角度，我认为微信是一个极具生命力和想象空间的移动产*

品，它的布局和设计都有可能颠覆移动互联网的明天。”我们可以看出，它的主题应该就是 “微信”、“移动互联网”等。所以，对于一篇文档，我们希望能得到它的主题（一些词），以及这些词属于哪些主题的概率，这样我们就可以进一步分析文档了。

主题模型有不少算法，最经典的两个是：PLSA（Probabilistic Latent Semantic Analysis）和 LDA（Latent Dirichlet Allocation）。

首先来看下 PLSA 模型。$d \in D$表示文档，$w \in V$表示词语，z 表示隐含的主题。$P(d_i)$表示单词在文档d_i中出现概率，$P(z_k|d_i)$表示主题z_k在给定文档d_i下出现的概率，$P(w_j|z_k)$表示单词w_j在给定主题z_k下出现的概率。PLSA 是个典型的生成模型，根据下方的图模型可以写出文档中每个词的生成概率：

$$P(d_i, w_j) = P(d_i)P(w_j|d_i) = P(d_i) \sum_{k=1..K} P(w_j|z_k)p(z_k|d_i)$$

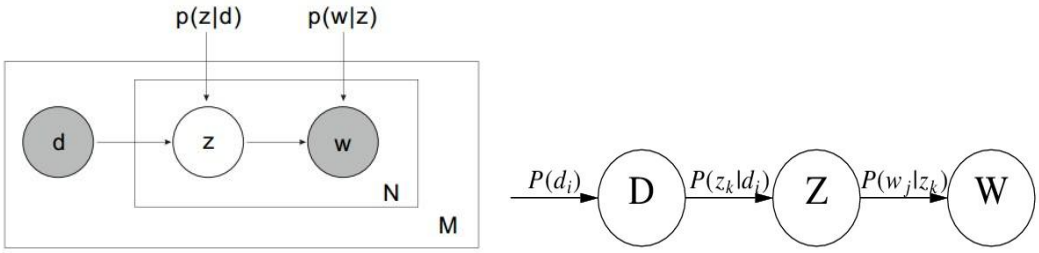

图 3.9（两种不同的表示方法）

由于词和词之间是互相独立的，文档和文档间也是相互独立的，那么整个样本集的分布为：

$$P(D, V) = \prod_{i=1..N} \prod_{j=1..M} P(d_i, w_j)^{n(d_i, w_j)}$$

首先，我们看看，PLSA 模型待估计的参数是什么？就是 $\theta = \{P(w_j|z_k), P(z_k|d_i)\}$，那么样本集的 log 似然函数就为：

$$L(\theta) = \log P(V, D|\theta) = \sum_{i=1..M} \sum_{j=1..N} n(w_j, d_i) \log P(d_i, w_j)$$

其中，$n(w, d)$表示单词 w 在文档 d 中出现的次数，$n(d)$表示文档 d 中词的个数。

好，现在有了 log 似然函数，那么就可以对其求导解出参数了，我们知道我们的参数θ共有$(N \times K + M \times K)$个，而且自变量是包含在对数和中，这就意味着这个方程组的求解很困难，那就要考虑其他方法求解了，而且我们这个问题还包含隐藏变量，就要使用 EM 算法。

EM 算法就是根据已经观察到的变量对隐藏变量进行学习的方法。既然没办法最大化$L(\theta)$，那么$L(\theta)$总该有个下限吧，我们就优化这个下限，不断迭代提高这个下限，那么就可以得到近似最大解了，这个下限其实就是似然函数的期望。通常，EM 算法得到的是局部近似解。

首先对参数$P(w_j|z_k)$和 $P(z_k|d_i)$赋值随机值。

EM 算法第一步 E-step：求隐藏变量的后验概率：

$$P(z_k|d_i, w_j) = \frac{P(w_j|z_k)P(z_k|d_i)}{\sum_{k=1..K} P(w_j|z_k)P(z_k|d_i)}$$

EM 算法第二步 M-step：最大化似然函数的下限，解出新的参数。

那么，现在的问题就是找到$L(\theta)$的下限，可以推导出：

$$L(\theta) = \sum_{i=1..M} \sum_{j=1..N} n(w_j, d_i) \log \sum_{k=1..K} P(z_k|d_i)\, P(w_j|z_k)$$

$$\geq \sum_{i=1..M} \sum_{j=1..N} n(w_j, d_i) \sum_{k=1..K} P(z_k|d_i, w_j) \log(P(z_k|d_i)P(w_j|z_k)) = Q(\theta)$$

$Q(\theta)$就是下限，那么最大化它就可以了，但是还有约束条件：

$$\sum_{j=1..N} P(w_j|z_k) = 1$$

$$\sum_{k=1..K} P(z_k|d_i) = 1$$

那么这时写出拉格朗日函数：

$$H = \sum_{i=1..M} \sum_{j=1..N} n(w_j, d_i) \sum_{k=1..K} P(z_k|d_i, w_j) \log\left(P(z_k|d_i)P(w_j|z_k)\right) + \sum_{k=1..K} \tau_k\left(\sum_{j=1..N} P(w_j|z_k) - 1\right) + \sum_{i=1..M} \rho_i\left(\sum_{k=1..K} P(z_k|d_i) - 1\right)$$

然后分别对$P(w_j|z_k)$和$P(z_k|d_i)$求导（注意：$P(z_k|d_i, w_j)$是已知的，在 E-step 已经计算好了），然后联合约束条件，解得：

$$P(w_j|z_k) = \frac{\sum_{i=1..M} n(w_j, d_i)P(z_k|d_i, w_j)}{\sum_{j=1..N}\sum_{i=1..M} n(w_j, d_i)P(z_k|d_i, w_j)}$$

$$P(z_k|d_i) = \frac{\sum_{j=1..N} n(w_j, d_i)P(z_k|d_i, w_j)}{n(d_i)}$$

然后把$P(w_j|z_k)$和$P(z_k|d_i)$代入到 E-step 接着迭代。

在这儿，也总结下这个经典的 EM 算法，假设 log 似然函数为 $L(\theta) = \log\sum_Z P(X, Z|\theta)$，那么 EM 算法步骤为：

（1）随机初始化θ_i。

（2）EM 步骤：

while ($|\theta_{i+1} - \theta_i| < \delta$):

E-step：计算后验概率$P(Z|X, \theta_i)$。

M-step：$\theta_{i+1} = \text{argmax}_\theta \sum_Z P(Z|X, \theta_i)\log P(X, Z|\theta)$

至此，PLSA 算法就求解完了，它和其他算法的求解过程其实没多大区别，唯一的是多了隐藏变量，然后使用 EM 算法求解。

PLSA 求出了所有$P(z_k|d_i)$和$P(w_j|z_k)$，也就是文档d_i属于主题z_k的概率和主题z_k下各个单词w_j的概率，但是 PLSA 并没有考虑参数的先验知识，这时候出现了另一个改进的算法：LDA，它对参数增加了先验分布（所以理论上 LDA 比 PLSA 不容易过拟合），也就是说参数也是一个分布，这个分布的参数（参数的参数）叫做超参数。

LDA 是个挺复杂的模型，我们简单看下它的大致思路，下图就是从论文中截取的 LDA 的图模型。

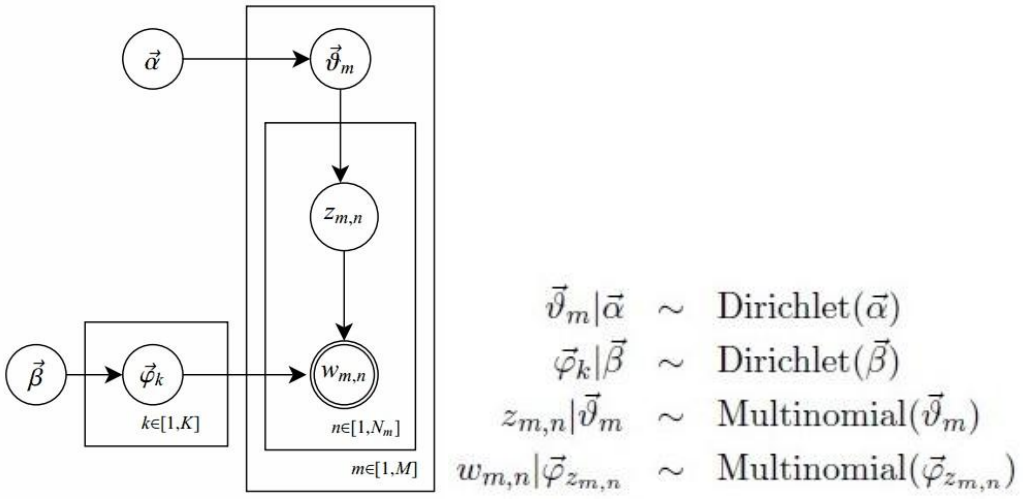

图 3.10

上图中，$w_{m,n}$是观察到的变量（文档 m 中第 n 个词），其他都是参数或者隐藏变量，$\vec{\alpha}$，$\vec{\beta}$就是超参数。$\phi = \{\overrightarrow{\varphi_k}\}$表示主题和词之间的分布，是一个$M \times K$的矩阵，$\Theta = \{\overrightarrow{\vartheta_m}\}$表示文档和主题之间的分布，是一个$K \times V$的矩阵。M 文档数，K 主题数，V 词数，$N_m$ 文档 m 的长度。

LDA 有个假设：$\vec{\alpha}$->$\overrightarrow{\vartheta_m}$服从 Dirichlet 分布，$\overrightarrow{\vartheta_m}$->$z_{m,n}$服从 Multinomial 分布，$\vec{\beta}$->$\overrightarrow{\varphi_k}$服从 Dirichlet 分布，$\overrightarrow{\varphi_k}$->$w_{m,n}$服从 Multinomial 分布，大家知道 Dirichlet 分布和 Multinomial 分布是一对共轭分布，那么为什么 LDA 要假设共轭分布呢？当某个似然概率异常复杂时，后验概率计算会叫人非常头大，使用共轭先验可以简化问题！而且 Dirichlet 分布和

Multinomial 分布都研究透了，它们的概率分布啊，期望啊等都是可以直接求解出来的，在推导最终参数时就可以直接使用了。

假设某个分布（观察变量为X），要估计其中的参数θ。参数θ有个先验分布$p(\theta)$，贝叶斯法则告诉我们$p(\theta|X) \propto p(X|\theta)p(\theta)$, 可以知道，若$p(\theta)$与$p(X|\theta)$有相同的函数形式，那么后验概率$p(\theta|X)$和它们也有相同的函数形式，当然也与先验概率$p(\theta)$有相同的函数形式，这样使得$\theta$的后验概率与先验概率具有相同的表达式，只是参数不同而已！所以选择与似然函数共轭的先验，得到的后验函数只是参数调整后的先验函数。

同样，可以写出 LDA 的似然函数，然后使用最大似然估计求解参数，但是似然函数参数耦合太大，无法求解出来，而且包括隐藏变量，所以就像 PLSA 一样，想到了 EM 算法，但是 LDA 相比 PLSA 还有一个困难：后验概率无法求解出来（EM 算法中 E-step 要求解后验概率），那么这时就又要近似计算了：变分-EM 算法。

变分推理是一种近似计算后验概率的方法，首先寻找一个和原来不能直接求解的后验概率等价或者近似的函数 Q，然后就通过求解最优近似函数 Q 的参数来近似得到原后验概率的参数。

这样变分-EM 算法迭代的流程如下：

（1）设定参数的初始值；

（2）E-step：计算近似函数 Q；

（3）M-step：最大化近似函数 Q，解出参数；

求解 LDA 的参数还有一种方法：Gibbs Sampling。

Gibbs Sampling 算法是 MCMC 的一个特例，如果某个概率 P(X)不易求得，那么可以交替的固定某一维度x_i，然后通过其他维度$x_{\neg i}$（去除x_i的其他所有值）的值来抽样近似求解，也就是说，Gibbs 采样就是用条件分布的采样来替代全概率分布的采样。

回到 LDA，使用 Gibbs Sampling，就是要迭代求解：

$$P(z_{m,n} = k|\vec{w}, z_{\neg(m,n)}, \vec{\alpha}, \vec{\beta})$$

这个公式要利用联合概率$P(\vec{w}, \vec{z}|\vec{\alpha}, \vec{\beta})$，而联合概率的计算其实就是利用 Dirichlet 分布和 Multinomial 分布推导出来的，这两个分布是已知的，所以$P(z_{m,n} = k|\vec{w}, z_{\neg(m,n)}, \vec{\alpha}, \vec{\beta})$自然可以求解出来了。

当 Gibbs Sampling 收敛后，根据图模型就可以求解后验分布了：$P(\overrightarrow{\vartheta_m}|\overrightarrow{z_m}, \vec{\alpha})$和$P(\overrightarrow{\varphi_k}|\vec{z}, \vec{\alpha}, \vec{\beta})$（共轭分布的性质：它们就和先验分布一样，只是参数不同，所以可以求解出来），然后使用它们的期望就可以解出所有$\overrightarrow{\varphi_k}$和$\overrightarrow{\vartheta_m}$了。

所以整个基于 Gibbs Sampling 的 LDA 算法流程为：

（1）初始化参数。

（2）对所有文档 m=1..M

对文档 m 中所有的单词 n=1..N_m

-采样每个单词$w_{m,n}$对应的主题$z_{m,n}$=k；

-对单词$w_{m,n}$的主题 k 增加计数（“文档-主题”计数，“文档-主题”总数，“主题-单词”计数，“主题-单词”总数）；

（3）迭代步骤：

对所有文档 m=1..M

对文档 m 中所有的单词 n=1..N_m

-对单词$w_{m,n}$的主题 k 减少计数；

-根据$P(z_{m,n} = \tilde{k}|\vec{w}, z_{\neg(m,n)}, \vec{\alpha}, \vec{\beta})$采样新主题；

-对单词$w_{m,n}$的新主题$\tilde{k}$增加计数；

收敛后，利用公式计算所有$\overrightarrow{\varphi_k}$和$\overrightarrow{\vartheta_m}$；

至此 LDA 就求解出来了，如果想深入了解 LDA 的每个细节：下面的文章不可不读（当然还有其他文章，不一一列举了）：

Variational Message Passing and its Applications

Latent Dirichlet Allocation

Parameter estimation for text analysis

The Expectation Maximization Algorithm A short tutorial

Gibbs Sampling in the Generative Model of Latent Dirichlet Allocation

乍一看 LDA 很难理解，但是从最终推导的公式来看，代码还是很清晰的，下面就是 GibbsSampling 的核心代码，其实还是很容易理解的：

```
// --- model parameters and variables ---
int M; // dataset size (i.e., number of docs)
int V; // vocabulary size
int K; // number of topics
double alpha, beta; // LDA hyperparameters
int niters; // number of Gibbs sampling iterations
double * p; // temp variable for sampling
int ** z; // topic assignments for words, size M x doc.size()
int ** nw; // cwt[i][j]: number of instances of word/term i assigned to topic j, size V x K
int ** nd; // na[i][j]: number of words in document i assigned to topic j, size M x K
int * nwsum; // nwsum[j]: total number of words assigned to topic j, size K
int * ndsum; // nasum[i]: total number of words in document i, size M
double ** theta; // theta: document-topic distributions, size M x K
double ** phi; // phi: topic-word distributions, size K x V
int model::init_est()
{
    //...
```

```
    srandom(time(0)); // initialize for random number generation
    z = new int*[M];
    for (m = 0; m < ptrndata->M; m++)
    {
        int N = ptrndata->docs[m]->length;
        z[m] = new int[N];

        // initialize for z
        for (n = 0; n < N; n++)
        {
            int topic = (int)(((double)random() / RAND_MAX) * K);
            z[m][n] = topic;

            // number of instances of word i assigned to topic j
            nw[ptrndata->docs[m]->words[n]][topic] += 1;
            // number of words in document i assigned to topic j
            nd[m][topic] += 1;
            // total number of words assigned to topic j
            nwsum[topic] += 1;
        }
        // total number of words in document i
        ndsum[m] = N;
    }
    //...
    return 0;
}
void model::estimate()
{
    printf( "Sampling %d iterations!\n" , niters);

    int last_iter = liter;
    for (liter = last_iter + 1; liter <= niters + last_iter; liter++)
    {
        printf( "Iteration %d ...\n", liter);

        // for all z_i
        for (int m = 0; m < M; m++)
        {
            for (int n = 0; n < ptrndata->docs[m]->length; n++)
            {
                // (z_i = z[m][n])
                // sample from p(z_i|z_-i, w)
                int topic = sampling(m, n);
                z[m][n] = topic;
            }
        }

        if (savestep > 0)
        {
            if (liter % savestep == 0)
            {
                // saving the model
                printf( "Saving the model at iteration %d ...\n" , liter);
                compute_theta();
                compute_phi();
                save_model(utils::generate_model_name(liter));
            }
        }
    }//end for

    printf( "Gibbs sampling completed!\n" );
    printf( "Saving the final model!\n" );
    compute_theta();
    compute_phi();
    liter--;
    save_model(utils::generate_model_name(-1));
}
int model::sampling(int m, int n)
{
    // remove z_i from the count variables
    int topic = z[m][n];
```

```
        int w = ptrndata->docs[m]->words[n];
        nw[w][topic] -= 1;
        nd[m][topic] -= 1;
        nwsum[topic] -= 1;
        ndsum[m] -= 1;

        double Vbeta = V * beta;
        double Kalpha = K * alpha;

        // do multinomial sampling via cumulative method
        for (int k = 0; k < K; k++)
        {
                p[k] = (nw[w][k] + beta) / (nwsum[k] + Vbeta) *
                        (nd[m][k] + alpha) / (ndsum[m] + Kalpha);
        }

        // cumulate multinomial parameters
        for (int k = 1; k < K; k++)
        {
                p[k] += p[k - 1];
        }

        // scaled sample because of unnormalized p[]
        double u = ((double )random() / RAND_MAX) * p[K - 1];

        for (topic = 0; topic < K; topic++)
        {
              if (p[topic] > u)
                    break;
        }

        // add newly estimated z_i to count variables
        nw[w][topic] += 1;
        nd[m][topic] += 1;
        nwsum[topic] += 1;
        ndsum[m] += 1;

        return topic;
}
void model::compute_theta()
{
        for (int m = 0; m < M; m++) {
                for (int k = 0; k < K; k++) {
                    theta[m][k] = (nd[m][k] + alpha) / (ndsum[m] + K * alpha);
                }
        }
}
void model::compute_phi()
{
        for (int k = 0; k < K; k++) {
                for (int w = 0; w < V; w++) {
                    phi[k][w] = (nw[w][k] + beta) / (nwsum[k] + V * beta);
                }
        }
}
// ------    end -------
```

3.5 深度学习(Deep Learning)

深度学习（Deep Leaning）是 Hinton 于 2006 年提出来的，然而在 2012 年开始，它变的异常火爆，几乎到处都能听见 deep learning，但是它本质上还是一个机器学习模型，只是解决问题的思路有所不同。

之前我们讲过，传统的机器学习需要提取特征，然后建立模型学习，但是特征是人工提取，如果不需要人工参与，那该多好啊，深度学习就是解决这个问题，所以它也叫无监督特征学习。

前面说过，机器学习的过程其实就是模拟人学习的过程，那么人最聪明的地方是大脑，如果机器能模拟大脑，那势必会更聪明些。20 世纪 80 年代末的神经网络就是模拟人脑的一个学习模型，最经典的就是 BP 神经网络，它有三层网络模型（输入层，隐藏层，输出层），但是它有很多缺点：初始值的选取是随机的，很容易收敛到局部最优解，导致过拟合；如果增加隐藏层的话，就会使传递到前面的梯度越来越稀疏，收敛速度很慢；它没有利用海量的未标注数据。

所以为了克服浅层神经网络的训练缺陷，深度学习是在海量数据中采用贪婪式的逐层学习方法：首先是无监督训练，单独训练一层，然后把该层的输出作为下一层的输入，使用相同的方法一直向上训练；然后到最上层是个有监督的从上到下的微调。所以一般来说使用深度学习有两个框架：

（1）无监督学习+有监督学习：首先从大量未标注数据中无监督逐层学习特征，然后把学习到的特征放到传统的监督学习方法来学习模型。

（2）有监督学习：首先逐层学习，到网络最上层是一个分类器（例如：softmax 分类器），整个是一套深层网络模型。

常用的深度学习模型有：

Auto-Encoders（自动编码器）

Sparse Coding（稀疏编码）

RBMs（限制玻尔兹曼机）

DBNs（深信念网络）

CNN（卷积深度网络）

深度学习的优势在于海量训练数据和好的训练模型（个人认为海量数据起的作用更大），既然有海量训练数据，那么对计算性能就会有很高的要求，一般都是使用 GPU 来训练深度学习模型。

我们看看深度学习在文本上是否有新颖的应用。我们前面提到过，在自然语言处理中有个假设：bag of words，对于一篇文档，它由很多词组成，如果把字典中所有词按照字母序排成一个很长的向量，那么每篇文档就可以使用这个长向量来表示了，某个词出现了，就标记为 1，没有出现的词标记为 0，可以想象，每个文档将会是一个稀疏向量。把文档表示成向量就会有很多缺点，比如：“电脑”和“计算机”有很大的语义相似性，但是表示成向量它们就没有语义关系了，因为它们在词典中的不同位置，不管用什么方法（cos）都没法计算出很好的相似性，也就是说把词表示成一个槽位是不合理的，那么我们可不可以把词表示出更丰富的信息呢？这就是词表示（word embedding）。

词表示就是想办法把词表示成一个向量，既然表示成向量了，那么两个词之间就可以计算相似性了（cos），这样就可以较好的表示语义了。词表示都是通过神经语言模型来训练得到的，目前大致有两种框架：NNLM 和 RNNLM。

NNLM（Neural Net Language Model）最经典的当属 Bengio 于 2001 年发表的通过语言模型得到词表示的方法了，如图 3.11 所示。

这是一个三层语言模型（有的理解为四层，即把输入层分成了输入层和投影层），它根据已知的前 n-1 个词（$w_{t-n+1},..,w_{t-1}$）来预测下一个词w_t，C(w)表示词 w 对应的词向量（C 就是所有词的词向量，是|V|×m 的矩阵），|V|表示词的总个数，m 表示词向量的维数，h 表示隐藏层个数，U（|V|×h 的矩阵）是隐藏层到输出层的参数，W（|V|×(n-1)m 的矩阵）是输入层到输出层的参数。

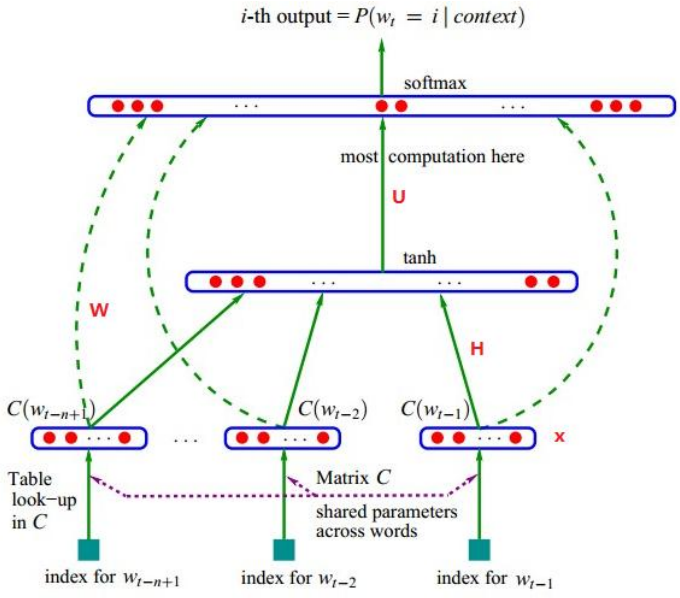

图 3. 11

整个网络的输出计算公式为：

$$y = b + Wx + U\tanh(d + Hx)$$

解释下，网络的输入层就是根据前 n-1 词（$w_{t-n+1},..,w_{t-1}$），找到它们在 C 中对应的向量（$C(w_{t-n+1}),..,C(w_{t-1})$），然后连成一个(n-1)m 维的一维向量 x，即 $x=(C(w_{t-n+1}),..\ C(,w_{t-1}))$；网络的隐藏层和普通神经网络一样做一个线性变换，然后取 tanh；网络的输出层总共有|V|个节点，每个节点表示的就是根据已知的前 n-1 个词，下一个词是该词的概率，这个概率最后使用 softmax 进行归一化，即有（b + Wx表示从输入层到输出层直接有个线性变换）：

$$P(w_t|w_{t-n+1},..,w_{t-1}) = \frac{e^{y_{w_t}}}{\sum_i e^{y_i}}$$

那么，这个模型的参数为$\theta = (b, d, W, U, H, C)$，求解模型就可以使用梯度下降法求解了，即：

$$\theta = \theta + \varepsilon \frac{\partial \log P(w_t|w_{t-n+1},..,w_{t-1})}{\partial \theta}$$

可以看出，这个模型的计算复杂度很高，尤其是隐藏层到输出层的那个大矩阵相乘，所以，为了降低计算复杂度，提出了一些改进的模型，比如下面的 LBL（log-bilinear language model），框架图如图 3.12 所示。

它的网络的计算公式为：

$$h_{w_n} = \sum_{i=1..n-1} H_i \times C(w_i)$$

$$y = C(w_i)^T h_{w_n}$$

$$P(w_n|w_{n-1},..,w_1) = \frac{e^{y_{w_n}}}{\sum_i e^{y_i}}$$

其中$C(w)$就是词 w 对应的词向量。

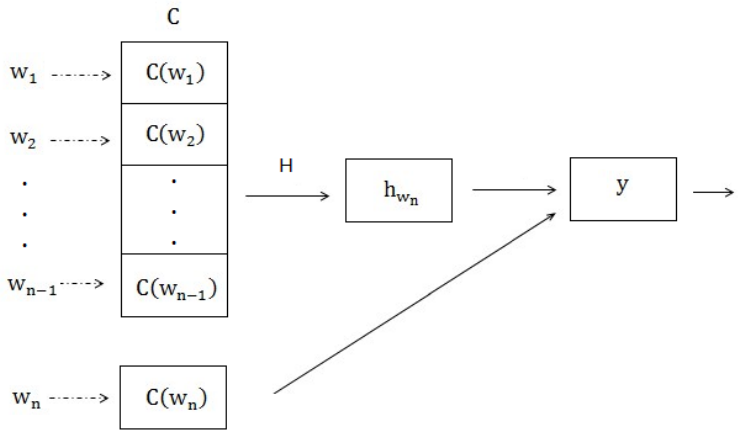

图 3.12

这个模型其实最好理解，h 隐藏层就是前 n-1 个词经过 H 变换之后得来的，也就是前 n-1 个词经过 H 这个变换之后就可以用来预测第

n 个词，然后输出层是什么呢？就是隐藏层和第 n 个词本身做内积，内积是可以表示相似度的，也就是说输出层是隐藏层所预测的第 n 个词和真实的第 n 个词的相似度，最后用 softmax 把概率归一一下。

RNNLM（Recurrent Neural Net Language Model）和上面的方法原理一样，但是思路有些许不同，框架图如图 3.13 所示。

它的网络的计算公式为：

$$s(t) = \text{sigmoid}\big(Uw(t) + Ws(t-1)\big)$$

$$P(w_{t+1}|w_t, s(t-1)) = y(t) = \text{softmax}(Vs(t))$$

其中，w(t)表示当前词w_t的向量，它这个向量的大小就是所有词汇的个数，所以 w(t)是个很长的且只有一个元素为 1 的向量，$s(t-1)$是上次的隐藏层，y(t)是输出层，它和 w(t)具有相同的维数，代表根据当前词w_t和隐藏层对词w_{t+1}的预测概率，由于 w(t)只有一个元素为 1，所以Uw(t)就是该词对应的词向量。整个过程就是来一个词，就和上一个隐藏层联合计算下一个隐藏层，然后反复进行这个操作，所以它对上下文信息利用的非常好。

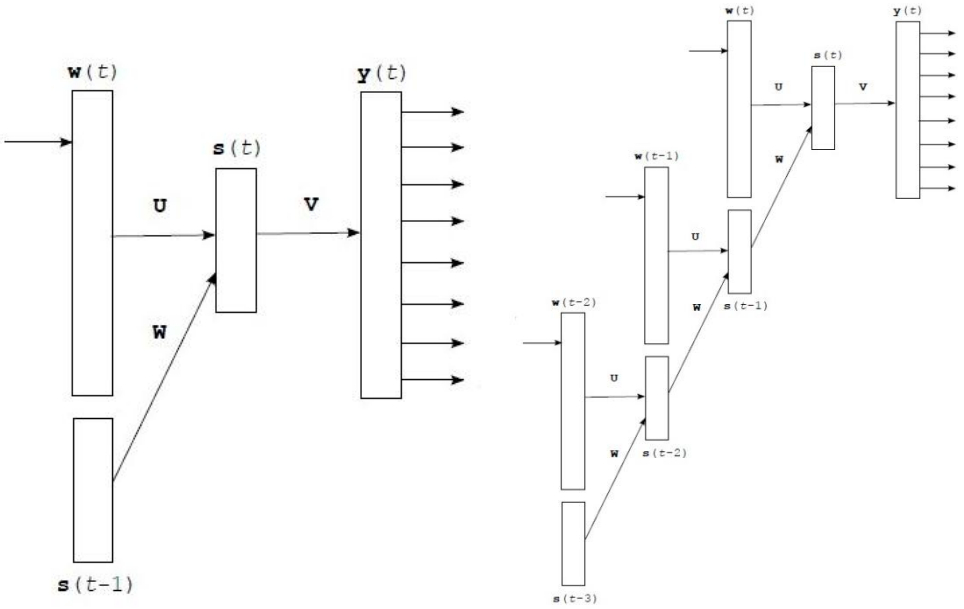

图 3.13

那么怎么评价词向量到底靠谱不靠谱呢？主要还是看它应用到具体任务的指标。既然将词表示成了向量，那么向量之间就可以计算内积、加减等运算，如果两个词有语义关系，那么它们就可以体现在向量上表示出来，就是计算内积（cos），我们试下找一些词，然后找出和它 cos 值最相似的 top 词看看结果，如下图 3.14 所示（在 60G 文本语料中训练得到），可以看出效果还可以，既然单个词可以计算相似度，那么很自然的想法就是对向量加减运算后也是向量，也可以计算相似性：C(中国)-C(美国)+C(华盛顿)的词向量应该和 C(北京)最相似。但是只能说热门词效果会比较好，对于不太用的词效果也并不是很理想，这个主要和语料有很大关系，语料越大，那么词向量就越好。很显然，数据越多（大数据），对统计方法来说越有利，而且如果在同一领域（topic）内的话，数据就相对不会太稀疏，效果又会更好，所以一般我们处理任务的方法就是，先在通用的领域上完成某个任务，要想达到更好的效果，就要细分领域来处理（所以分类问题是任何领域都会涉及到的问题），比如：google 很著名的 PageRank 算法也优化成了 Topic-Sensitive PageRank（主题敏感 PageRank）。

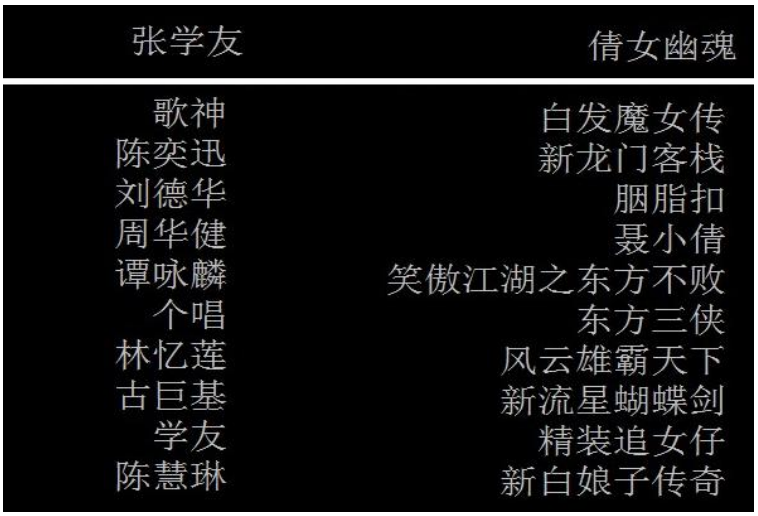

图 3.14

目前词向量一般也有两种用法：（1）直接把词向量作为神经网络的输入完成一些 NLP 任务，例如，Natural Language Processing (Almost) from Scratch 这篇文章的工作；（2）把词向量当做一个特征，加入到现有的 NLP 任务中。我们曾经尝试将词向量作为命名实体识别的一个额外特征，效果虽然有提升，但是并不像在图像和语音应用中那么明显，目前来说，深度学习在 NLP 的应用还不能用惊艳来形容，还有待大家共同努力尝试。

3.6 其他：kNN， k-means，决策树，SVM

3.6.1 kNN

kNN（k 最近邻）算法的思想比较简单：如果一个样本在特征空间中的 k 个最相似（即特征空间中“距离”最近）的样本中的大多数属于某一个类别，则该样本也属于这个类别。它是一种监督学习的分类算法。

3.6.2 k-means

k-means（k 均值）聚类算法是假定在欧式空间下，并且 k 是事先确定的。首先随机选取 k 个质心（一般会选择尽可能相互远的点），然后将各个数据项分配给“距离”最近的质心点，分配后，该类下的质心就会要更新（比如：变成该类下所有节点的平均值），该分配过程一直下去，直到聚类结果不再变化。

3.6.3 决策树

决策树是一个树结构。其每个非叶子节点表示一个特征属性上的判断，每个分支代表这个特征属性在某个值域上的输出，而每个叶节点存放一个类别。使用决策树进行决策的过程就是从根节点开始，判断待分类项中相应的特征属性，并按照其值选择输出分支，直到到达叶子节点，将叶子节点存放的类别作为决策结果。其实就是从跟节点开始，进行 if-else 判断一直到叶子节点，就得到了分类结果。

决策树的构造，也就是如何根据特征属性确定每个分支，指导思想就是这个分支一定要能最大化的区分不同类。为了确定哪个属性最适合用来拆分，算法会计算相应的信息增益，所谓某个特征的信息增益，就是指该特征对数据样本的分类的不确定性的减少的程度，也就是说信息增益越大的特征具有更强的区分能力，是指当前熵与两个新群组经加权平均后的熵之间的差值。算法会针对每个特征属性计算相应的信息增益，然后从中选出信息增益最大的特征属性。具体算法有 ID3 算法和 C4.5 算法。

3.6.4 SVM

SVM（支持向量机）自从 Cortes 和 Vapnik 于 1995 年提出来以后，由于它的理论完备，效果也很不错，逐渐风光起来了，它是一个很经典的分类模型。

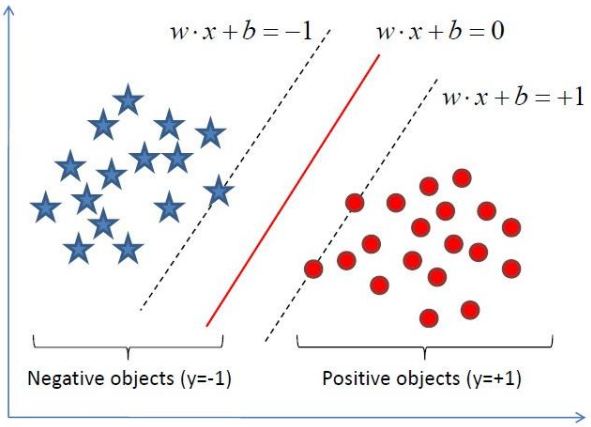

图 3.15

SVM 既然是一个机器学习模型，那么它就会有目标函数和约束条件，如图 3.15 所示，蓝色的是负样本点，都满足$w \cdot x + b \leq -1$；红色的是正样本点，都满足$w \cdot x + b \geq 1$，中间的红线是分类面$w \cdot x + b = 0$，那么要想使得红色和蓝色分的越好，就要使正负样本**线**上的点（正负样本线上的点都分开了，那么它们两侧的点自然也分开了）离分类面越远越好，即（点到面的距离公式）：$2 \times |w \cdot x + b|/||w||$，又因为$|w \cdot x + b| = 1$（因为是在正负样本线$w \cdot x + b = \pm 1$上），所以它的优化目标函数就是最大化几何间隔$2/||w||$（相当于最小化$||w||^2/2$），它的约束条件就是样本点必须在负样本线（$w \cdot x + b = -1$）和正样本线$w \cdot x + b = +1$的两侧，不能落入两者中间，即：

$$\min_{w,b} \quad ||w||^2/2$$

$$s.t. \quad y_i(w \cdot x_i + b) \geq 1 \quad (i = 1..N)$$

但是样本集免不了会有噪声，也就是会有样本不满足约束条件，落入了正负样本线之间，因此就要引入一个松弛变量，允许一些样本不满足约束条件，但是要惩罚，这样整个最优化函数就又变了，即：

$$\min_{w,b,\varepsilon} \quad ||w||^2/2 + C\sum_{i=1..N}\varepsilon_i$$

$$s.t. \quad y_i(w\cdot x_i + b) \geq 1 - \varepsilon_i \quad (i = 1..N)$$

$$\varepsilon_i \geq 0 \quad (i = 1..N)$$

现在 SVM 求解线性分类没什么问题了，那么如果线性方法无法区分呢？如果将线性无法区分的问题映射到高维空间（x->ϕ(x)），然后在高维空间可以线性区分的话，那就可以同样使用之前的线性方法了（如图 3.16），即：

$$\min_{w,b} \quad ||w||^2/2$$

$$s.t. \quad y_i(w\cdot \phi(x_i) + b) \geq 1 \quad (i = 1..N)$$

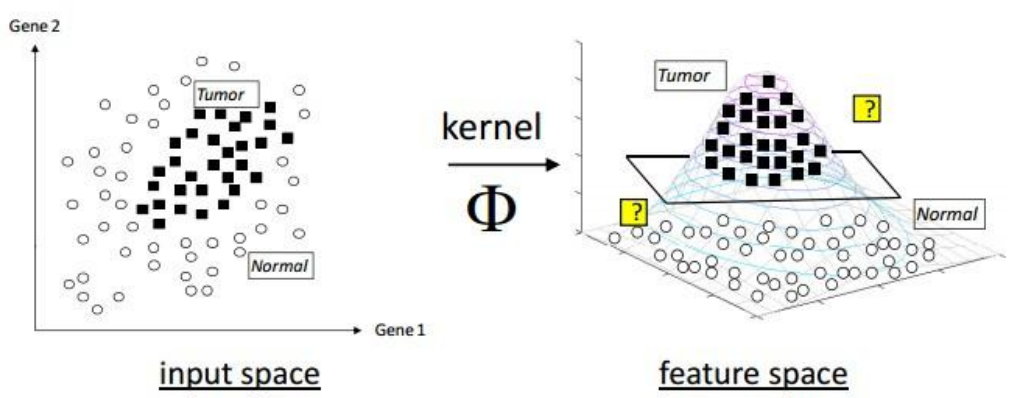

图 3.16

求解上面那个问题，可以使用对偶问题（还记得最大熵时讲的对偶问题吗？）来求解，最后就转化为下面的最优化问题了：

$$\min_{\alpha} \quad \frac{1}{2}\sum_{i=1..N}\sum_{i=1..N}\alpha_i\alpha_j y_i y_j\left(\phi(x_i)\cdot\phi(x_j)\right) - \sum_{i=1..N}\alpha_i$$

$$s.t. \quad \sum_{i=1..N}\alpha_i y_i = 0$$

$$C \geq \alpha_i \geq 0 \quad (i = 1..N)$$

问题来了，如何从低维空间映射到高维空间呢？也就是ϕ(x)如何选取？而且还要在高维空间计算$\phi(x_i)\cdot\phi(x_j)$。这时就要引入核函数，

它的作用就是把两个低维空间的向量映射到高维空间，而且它同时把它们在高维空间里的向量内积值都算好了，也就是核函数$K(\mathrm{x_i},\mathrm{x_j})=\phi(\mathrm{x_i})\cdot\phi(\mathrm{x_j})$，一个函数干了两件事，先映射到高维空间然后计算好了内积，使得我们不用关心高维空间到底是什么了，因为我们直接通过核函数拿到了结果。这样，在核函数$K(\mathrm{x_i},\mathrm{x_j})$给定的条件下，可以利用求解线性分类问题的方法求解非线性分类问题。

那么有了最优化模型，而且是个凸二次规划问题，就可以使用 SMO 算法求解了，SMO 算法其实就是分而治之的思想。SVM 具体细节可以参考相关论文（很多），下面一篇是 SVM 开创原论文，一篇是讲的很好的 ppt：

Support-Vector Networks

A Gentle Introduction to Support Vector Machines in Biomedicine

应用篇

由于工作繁忙，会晚些时候完成…