HI，小伙伴你好~

我们在维护者**全网最大的计算机相关编程书籍分享仓库**，目前已有超过 **1000本** 的计算机经典书籍了。

其中涉及**C/C++、Java、Python、Go语言等各种编程语言，还有数据结构与算法、操作系统、后端架构、计算机系统知识、数据库、计算机网络、设计模式、前端、汇编以及校招社招各种面经**等~

只有你想不到，没有我们没分享到的计算机学习书籍，如果真的有我们没能分享到的书籍或者是你所需要的，欢迎添加下方联系方式来告诉我们，期待你的到来。

在此承诺本仓库**永不收费**，永远免费分享给有需要的人，希望自己的**幸苦结晶**能够帮助到曾经那些像我一样的小白、新手、在校生们，为那些曾经像我一样迷茫的人指明一条路。

告诉他们，你是可以的！

本仓库无偿分享各种计算机书籍、各种专业**PDF**资料以及个人笔记资料等，所有权归仓库作者**阿秀**（公众号【**拓跋阿秀**】）所有，如有疑问提请**issue**或者联系本人forthespada@foxmail.com，感谢~

衷心希望我以前踩过的坑，你们不要再踩，我走过的路，你们可以照着走下来。

因为从双非二本爬到字节跳动这种大厂来，太**TMD**难了。

QQ群：①群:1002342950、②群:826980895

目录

预备知识

概率论

最优化

算法

监督学习和无监督学习

监督学习

有监督学习：对具有概念标记（分类）的训练样本进行学习，以尽可能对训练样本集外的数据进行标记（分类）预测。这里，所有的标记（分类）是已知的。因此，训练样本的歧义性低。

无监督学习

无监督学习：对没有概念标记（分类）的训练样本进行学习，以发现训练样本集中的结构性知识。这里，所有的标记（分类）是未知的。因此，训练样本的歧义性高。聚类就是典型的无监督学习

解释：

机器学习中的方法或范式（paradigm）有很多种分类体系，例如从学习的方式分，有例子中学习、类比学习、分析学习等，但一般来说，现在研究得最多、被认为最有用的是从例子中学习。对从例子中学习，又有很多分类方法，例如从学习的主动性方面，可以分为主动学习和被动学习；从训练过程启动的早晚，可以分为迫切学习和惰性学习等等。最常见的对“从例子中学习”的方法的分类是监督学习、非监督学习和强化学习，这是从训练样本的歧义性

（ambiguity）来进行分类的。对监督学习来说，它通过对具有概念标记（concept label）的训练例进行学习，以尽可能正确地对训练集之外的示例的概念标记进行预测。这里所有训练例的概念标记都是已知的，因此训练样本的歧义性最低。
对非监督学习来说，它通过对没有概念标记的训练例进行学习，以发现训练例中隐藏的结构性知识。这里的训练例的概念标记是不知道的，因此训练样本的歧义性最高。对强化学习来说，它通过对没有概念标记、但与一个延迟奖赏或效用（可视为延迟的概念标记）

相关联的训练例进行学习，以获得某种从状态到行动的映射。这里本来没有概念标记的概念，但延迟奖赏可被视为一种延迟概念标记，因此其训练样本的歧义性介于监督学习和非监督学习之间。

需要注意的是，监督学习和非监督学习从一开始就是相对的，而强化学习在提出时并没有从训练样本歧义性的角度考虑其与监督学习和非监督学习的区别，因此，一些早期的研究中把强化学习视为一种特殊的非监督学习。事实上，对强化学习的定位到目前仍然是有争议的，有的学者甚至认为它是与“从例子中学习”同一级别的概念。

从训练样本歧义性角度进行的分类体系，在近几年可望有一些扩展，例如多示例学习（multi-instance learning）等从训练样本歧义性方面来看很特殊的新的学习框架有可能会进入该体系。但到目前为止，没有任何新的框架得到了公认的地位。另外，半监督学习（semi-supervised learning）也有一定希望，它的障碍是半监督学习中的歧义性并不是与生俱来的，而是人为的，即用户期望用未标记的样本来辅助对已标记样本的学习。这与监督学习、非监督学习、强化学习等天生的歧义性完全不同。半监督学习中人为的歧义性在解决工程问题上是需要的、有用的（对大量样本进行标记的代价可能是极为昂贵的），但可能不太会导致方法学或对学习问题视点的大的改变。

不同的分类体系是相交的，例如，监督学习方法既可能是迫切的（例如大多数神经网络、决策树等），也可能是惰性的（例如 k 近邻等）。另外，分类体系也不是绝对的，例如前面提到的强化学习的情况。

经验风险最小

定义

结构化风险 = 经验风险 + 置信风险

经验风险 = 分类器在给定样本上的误差(类似于训练误差)

置信风险 = 分类器在未知文本上分类的结果的误差（类似于测试误差）

样本数量，给定的样本数量越大，学习结果越有可能正确，此时置信风险越小；

分类函数的 VC 维，显然 VC 维越大，推广能力越差，置信风险会变大。 提高样本数量，降低 VC 维，降低置信风险。

以前机器学习的目标是降低经验风险，要降低经验风险，就要提高分类函数的复杂度，导致 VC 维很高，VC 维高，置信风险就高，所以，结构风险也高。---- 这是 SVM 比其他机器学习具有优势的地方。

当样本容量足够大时，经验风险最小化能保证有很好的学习效果，在现实中被广泛采用。例如，极大似然估计（MLE）就是经验风险最小化的一个例子。当模型是条件概率分布，损失函数是对数损失函数时，经验风险最小化就等于极大似然估计。 但是，当样本容量很小时，经验风险最小化学习的效果就未必很好，会产生过拟合现象。而结构风险最小化（structural risk minimization, SRM）是为了防止过拟合而提出的策略。结构风险最小化等价于正则化。

机器学习的关注点在于模型在测试集上的分类效果，这也称为泛化能力，泛化能力弱的话会产生欠拟合和过拟合。需要建立一个模型对欠拟合和过拟合进行说明。所以建立了 ERM 模型：

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

ERM 非凸，一般优化算法无法优化，因为它是 NP 的。这里引入 P 和 NP 的。P 就是能在多项式时间内解决的问题，NP 就是能在多项式时间验证答案正确与否的问题。用大白话讲大概就是这样。所以 P 是否等于 NP 实质上就是在问，如果对于一个问题我能在多项式时间内验证其答案的正确性，那么我是否能在多项式时间内解决它？这个表述不太严谨，但通俗来讲就是如此。

Logistic 和 SVM 都是对这种方法的凸近似。**可以证明最优化 ERM 能带来较小的泛化误**

差。首先要引入**联合界**与**一致收敛**定理，然后对其做相应的证明。

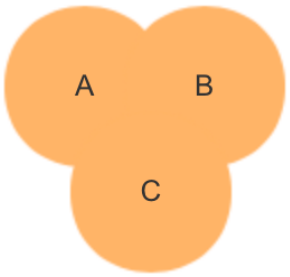
联合界(Union bound)和 Hoeffding 不等式

联合界定理

定义： 令 A1,A2,...,Ak 是 k 个时间，这 k 个事件可以相互独立也可以不相互独立，那么有下面结论：

P(A1∪A2∪...∪Uk)≤P(A1)+P(A2)+...+P(Ak)

该定理可以用 Venn 图来表示如下。



圆 A,B,C 分别代表着事件 A,B,C 发生的概率，之间有重叠，所以 ABC 任意一个发生的概念是小于三者发生的概率之和的。

Hoeffding 不等式

定义：令 Z1,Z2,...,Zk 为 k 个独立同分布变量，服从伯努利分布，即： P(Zi=1)=φ,P(Zi=0)=1−φ 。我们使用者 m 个变量的平均值来估计 φ ,得到

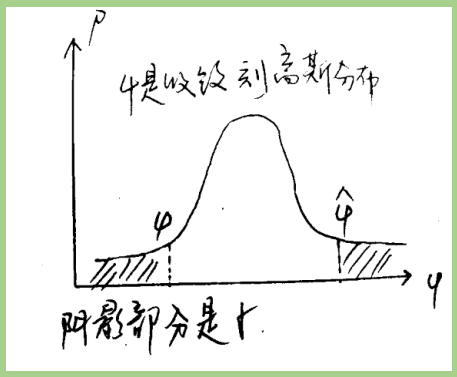
$$\hat{\varphi} = \frac{1}{m}\sum_{i=1}^{m} Z_i$$

那么 Hoeffding 不等式的定义即对任意固定的数值 $\gamma > 0$，存在

$$P(|\hat{\varphi} - \varphi| > \gamma) < 2\exp(-2\gamma^2 m)$$

这个定理表明当**样本数量 m 增大，对参数的估计越来越逼近真实值**。

含义：Hoeffding 不等式说的是**估值的差异有一个上界，即给定一个假设，训练误差会以一个很大的概率接近于一般误差。**可以通过下面的图像形象的看出来，



误差随着 m 的增大指数下降。也就是阴影部分的面积指数下降。

一致收敛

一致收敛就是训练误差和泛化误差随着样本的数目增大而更加接近。

- **证明对于所有的 h，$\hat{\varepsilon}$ 都是 ε 的一个很好的估计**

假设模型集合 $H=\{h_1,\ h_2,.....,h_k\}$，从模型中任意一个假设 h_j，会有

$$p(Z_i=1)=\varepsilon(h_j)\sim Bernoulli(\varphi),$$

训练误差是 m 个服从伯努利分布的随机变量之和，根据 Hoeffding 不等式引理，得到：

$$p(|\varepsilon(h_j)-\hat{\varepsilon}(h_j)|>\gamma)<2\exp(-2\gamma^2 m)$$

即证明。

- **使用 ERM 方法得到的 $\hat{h}$ 的泛化误差是有上限的**

令事件 A_j 为 $\varepsilon(h_j)-\hat{\varepsilon}(h_j)>\gamma$，那么有

$$p(A_j)<2\exp(-2\gamma^2 m)$$

可以推导出至少存在一个假设 h_i，使 $\varepsilon(h_j)-\hat{\varepsilon}(h_j)>\gamma$ 成立的概率为

$$P(\exists h\in H,|\varepsilon(h_i)-\hat{\varepsilon}(h_j)|>\gamma)=P(A_1\cup A_2\cup L\ A_k)\le P(A_1)+P(A_2)+L\ +P(A_k)\le 2\exp(-2\gamma^2 m)$$

两边 1 同时减去得到：

$$P(\exists h \in H, |\varepsilon(h_i) - \hat{\varepsilon}(h_j)| > \gamma) \geq 1 - 2\exp(-2\gamma^2 m)$$

至少有$1-2\exp(-2\gamma^2 m)$的概率，使得模型集合中所有假设，其泛化误差都在训练误差的γ范围内。

一致收敛推论

这里不详细证明过程，详细可以参考下面的博客

- 给定γ和$\sigma > 0$，需要多少样本，可以保证在至少有$1-\sigma$的概率，使得泛化误差概率在γ范围内？

$$1 - 2k\exp(-2\gamma^2 m) \geq 1 - \sigma$$

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

- 给定 m 和$\sigma > 0$，泛化误差会在什么范围之内？

$$\gamma = \sqrt{\frac{1}{2m}\log\frac{2k}{\sigma}}$$

ERM 参考：http://www.cnblogs.com/XBWer/p/4238995.html

KNN (最近邻)

定义

K 最近邻(k-Nearest Neighbour，KNN)分类算法，属于判别模型，是一个理论上比较成熟的方法，也是最简单的机器学习算法之一。该方法的思路是：**如果一个样本在特征空间中的 k 个最相似(即特征空间中最邻近)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**不具有显示的学习过程。**

在 KNN 中，通过计算对象间的距离来作为各个对象之间的非相似性指标，避免了对象之间的匹配问题，在这里距离一般使用**欧氏距离或曼哈顿距离，对于文本分类来说，使用余弦来计算相似度就比欧式距离更合适**：同时，KNN 通过依据 k 个对象中占优的类别进行决策，而不是单一的对象类别决策。这两点就是 KNN 算法的优势。

K 值的选取

K 较小容易导致过拟合，通常使用交叉验证来实现 K 的选择

算法描述

在训练集中数据和标签已知的情况下，输入测试数据，将测试数据的特征与训练集中对应的特征进行相互比较，找到训练集中与之最为相似的前 K 个数据，则该测试数据对应的类别就是 K 个数据中出现次数最多的那个分类，其算法的描述为：

1）计算测试数据与各个训练数据之间的距离；

2）按照距离的递增关系进行排序；

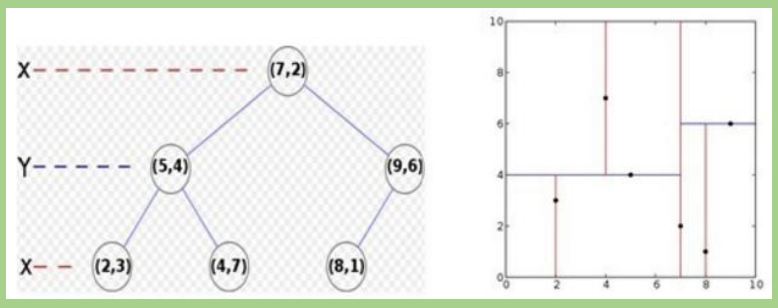
3）选取距离最小的 K 个点；

4）确定前 K 个点所在类别的出现频率；

5）返回前 K 个点中出现频率最高的类别作为测试数据的预测分类(**多数表决**)

KNN 实现

使用 kd 树的方法来实现，它是一种分割 k 维数据空间的数据结构。主要应用于多维空间关键数据的搜索（如：范围搜索和最近邻搜索）。K-D 树是二进制空间分割树的特殊的情况。



构造过程见 http://www.cnblogs.com/21207-iHome/p/6084670.html

注意：kd 树使用方差来选择分类特征，假设这个特征越离散越好区分，决策树使用信息增益来选择分类特征，假设这个特征信息增益越大分类能力越强

优缺点

K 近邻：算法采用测量不同特征值之间的距离的方法进行分类。

优点：

1.简单好用，容易理解，精度高，理论成熟，既可以用来做分类也可以用来做回归；

2.可用于数值型数据和离散型数据；

3.训练时间复杂度为 O(n)；无数据输入假定；

4.对异常值不敏感

缺点：

1.计算复杂性高；空间复杂性高；

2.样本不平衡问题（即有些类别的样本数量很多，而其它样本的数量很少）；

3.一般数值很大的时候不用这个，计算量太大。但是单个样本又不能太少 否则容易发生误分。

4.最大的缺点是无法给出数据的内在含义。

NB（朴素贝叶斯）

定义

朴素贝叶斯（Natvie Bayes）法是基于**贝叶斯定理**和**特征条件独立假设**的分类方法。

常用到的三个模型

高斯模型、多项式模型、伯努利模型

模型概述

朴素：**特征条件独立**　　贝叶斯：**基于贝叶斯定理**

根据贝叶斯定理，对一个分类问题，给定样本特征 x，样本属于类别 y 的概率是

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

通过给定的输入 x,通过学习得到的模型可以计算后验概率分布 $P(Y = c_k \mid X = x)$ 可以由贝叶斯定理得到：

$$P(Y=c_k \mid X=x)=\frac{P(X=x \mid Y=c_k)P(Y=c_k)}{\sum_k P(X=x \mid Y=c_k)P(Y=c_k)}$$

将

$$\begin{aligned} P(X=x \mid Y=c_k) &= P(X^{(1)}=x^{(1)},\cdots,X^{(n)}=x^{(n)} \mid Y=c_k) \\ &= \prod_{j=1}^{n} P(X^{(j)}=x^{(j)} \mid Y=c_k) \end{aligned}$$

带入到上式中可以得到

$$P(Y=c_k \mid X=x)=\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)} \mid Y=c_k)}{\sum_k P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)} \mid Y=c_k)}, \quad k=1,2,\cdots,K$$

所以贝叶斯分类器可表示为：

$$y=f(x)=\arg\max_{c_k}\frac{P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)} \mid Y=c_k)}{\sum_k P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)} \mid Y=c_k)}$$

简化后为

$$y=\arg\max_{c_k} P(Y=c_k)\prod_j P(X^{(j)}=x^{(j)} \mid Y=c_k)$$

分类

通过学到的概率，给定未分类新实例 X，就可以通过上述概率进行计算，得到该实例属于各类的后验概率 p(y=ck|X)

拉普拉斯平滑

$$p(y=c_k)=\frac{\sum_{i=1}^{N} I(y_i=c_k)+\lambda}{N+K\lambda}$$

$$p(x^j=a_{jl}|y=c_k)=\frac{\sum_{i=1}^{N} I(x_i^j=a_{jl},y_i=c_k)+\lambda}{\sum_{i=1}^{N} I(y_i=c_k)+L_j\lambda}$$

加入一个因子使得分母不为 0，以保证计算的可行性。

优缺点

优点：

在数据较少的情况下任然有效，可以处理多分类问题

缺点：

对输入数据的准备方式比较敏感

理论上，NBC 模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是因为 NBC 模型假设属性之间相互独立，这个假设在实际应用中往往是不成立的（可以考虑用聚类算法先将相关性较大的属性聚类），这给 NBC 模型的正确分类带来了一定影响。在属性个数比较多或者属性之间相关性较大时，NBC 模型的分类效率比不上决策树模型。而在属性相关性较小时，NBC 模型的性能最为良好。

需要知道先验概率。

分类决策存在错误率

适用数据类型：标称型数据

标称型：标称型目标变量的结果只在有限目标集中取值，如真与假(标称型目标变量主要用于分类)

数值型：数值型目标变量则可以从无限的数值集合中取值，如 0.100，42.001 等 (数值型目标变量主要用于回归分析)

MaxEnt（最大熵模型）

定义

最大熵原理是概率模型学习的一个准则。最大熵原理认为，在学习概率模型时，在所有可能的概率分布中，熵最大的模型是最好的模型。通常用约束条件来确定概率模型的集合，所以，最大熵模型也可以表述为在满足约束条件的模型集合中选取熵最大的模型。

特征的理解

特征就是我们说的那个 f(x,y)，描述的是输入 x 和输出 y 之间的某一个事实。

对于特征理解的核心是找到下面两者的联系：

1.仅仅对输入抽取特征。即特征函数为 f(x)

2.对输入和输出同时抽取特征。即特征函数为 f(x,y)

要看清二者的关系，一个简单的办法就是去考察题主提到的最大熵模型和 logistic 回归模型。确切地说，看看怎么把最大熵模型推导成 logistic 回归模型就可以了。最大熵模型定义了在给定输入变量 x 时，输出变量 y 的条件分布：

$$P(y|\mathbf{x},\boldsymbol{\theta}) = \frac{\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y)\right)}{\sum_{\mathbf{y}\in Dom(y)}\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y)\right)}$$

此处$Dom(y)$是y所有可能取值的集合。如果我们限定y为二元变量，即$Dom(y)=\{y_0,y_1\}$，那么就可以把最大熵模型转换为 logistic 回归模型。我们还需要定义特征函数为

$$\mathbf{f}(\mathbf{x},\mathbf{y}) = \begin{cases} \mathbf{g}(\mathbf{x}) & y=y_1 \\ \mathbf{0} & y=y_0 \end{cases}$$

即仅在 y=y1 时收取 x 的特征，在 y=y0 时不抽取任何特征((直接返回全为 0 的特征向量))

将这个特征函数带回最大熵模型，我们得到当$y=y_1$时

$$\begin{aligned}
P(y_1|\mathbf{x}) &= \frac{\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y_1)\right)}{\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y_0)\right)+\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y_1)\right)} && \text{最大熵模型定義} \\
&= \frac{\exp\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)}{\exp\left(\boldsymbol{\theta}\cdot\mathbf{0}\right)+\exp\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)} && \text{特徵函數 } \mathbf{f} \text{ 的定義} \\
&= \frac{\exp\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)}{1+\exp\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)} && e^0=1 \\
&= \frac{1}{\exp\left(-\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)+1} && \text{分子分母同除以} \exp\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right) \\
&= \sigma\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right) && \text{logistic 函數定義}
\end{aligned}$$

当 y=y0 时：

$$\begin{aligned}
P(y_0|\mathbf{x}) &= \frac{\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y_0)\right)}{\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y_0)\right)+\exp\left(\boldsymbol{\theta}\cdot\mathbf{f}(\mathbf{x},y_1)\right)} \\
&= \frac{\exp\left(\boldsymbol{\theta}\cdot\mathbf{0}\right)}{\exp\left(\boldsymbol{\theta}\cdot\mathbf{0}\right)+\exp\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)} \\
&= \frac{1}{1+\exp\left(\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)} \\
&= \frac{\exp\left(-\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)}{\exp\left(-\boldsymbol{\theta}\cdot\mathbf{g}(\mathbf{x})\right)+1} \\
&= 1-P(y_1|\mathbf{x})
\end{aligned}$$

我们发现，当类标签（class label）只有两个的时候，最大熵模型就是 logistic 回归模型。表面上看，logistic 回归模型里面的特征函数的确只考虑 x 不考虑 y，。然而通过上面的推导，

我们发现其实 g

抽取的特征仅仅在 y=y1 时被用到。另外，logistic 回归模型当然有特征的概念。给模型一句自然语言，它肯定不认识。我们必须抽出像 n 元祖-gram)、 词性（part-of-speech tag）等特征，才能把数据传给模型。特征函数无非就是模型的「眼睛」。

最大熵模型的学习

首先介绍一下：

特征函数 $f(x,y)$ 关于经验分布的 $\tilde{P}(X,Y)$ 的期望值，用 $E_{\tilde{p}}(f)$ 来表示：

$$E_p(f)=\sum_{x,y}\tilde{P}(x)f(x,y)$$

特征函数 $f(x,y)$ 关于模型 $P(Y|X)$ 与经验分布的 $\tilde{P}(X)$ 的期望值，用 $E_p(f)$ 来表示：

$$E_p(f)=\sum_{x,y}\tilde{P}(x)P(y|x)f(x,y)$$

两者是相等的，下面介绍最大熵模型的学习：

对于给定的训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$ 以及特征函数 $f_i(x,y)$，$i=1,2,\cdots,n$，最大熵模型的学习等价于约束最优化问题：

$$\begin{aligned}\max_{P\in C}\quad & H(P)=-\sum_{x,y}\tilde{P}(x)P(y|x)\log P(y|x)\\ \text{s.t.}\quad & E_P(f_i)=E_{\tilde{P}}(f_i),\quad i=1,2,\cdots,n\\ & \sum_y P(y|x)=1\end{aligned}$$

按照最优化问题的习惯，将求最大值问题改写为等价的求最小值问题：

$$\begin{aligned}\min_{P\in C}\quad & -H(P)=\sum_{x,y}\tilde{P}(x)P(y|x)\log P(y|x)\\ \text{s.t.}\quad & E_P(f_i)-E_{\tilde{P}}(f_i)=0,\quad i=1,2,\cdots,n\\ & \sum_y P(y|x)=1\end{aligned}$$

引入拉格朗日乘子后得到：

$$
\begin{aligned}
L(P,w) &\equiv -H(P) + w_0\left(1-\sum_y P(y|x)\right) + \sum_{i=1}^{n} w_i\left(E_{\tilde{P}}(f_i) - E_P(f_i)\right) \\
&= \sum_{x,y} \tilde{P}(x)P(y|x)\log P(y|x) + w_0\left(1-\sum_y P(y|x)\right) \\
&\quad + \sum_{i=1}^{n} w_i\left(\sum_{x,y}\tilde{P}(x,y)f_i(x,y) - \sum_{x,y}\tilde{P}(x)P(y|x)f_i(x,y)\right)
\end{aligned}
$$

通过对偶将原始问题 $\min_{P\in c}\max_{w} L(P,w)$ 转为对偶问题 $\max_{w}\min_{P\in c} L(P,w)$，凸函数，所以等价。

先求里面的：$L(P,w)$对$P(y|x)$ 的偏导：

$$
\begin{aligned}
\frac{\partial L(P,w)}{\partial P(y|x)} &= \sum_{x,y}\tilde{P}(x)\left(\log P(y|x)+1\right) - \sum_y w_0 - \sum_{x,y}\left(\tilde{P}(x)\sum_{i=1}^{n} w_i f_i(x,y)\right) \\
&= \sum_{x,y}\tilde{P}(x)\left(\log P(y|x) + 1 - w_0 - \sum_{i=1}^{n} w_i f_i(x,y)\right)
\end{aligned}
$$

令偏导等于 0，得到

$$
P(y|x) = \exp\left(\sum_{i=1}^{n} w_i f_i(x,y) + w_0 - 1\right) = \frac{\exp\left(\sum_{i=1}^{n} w_i f_i(x,y)\right)}{\exp(1-w_0)}
$$

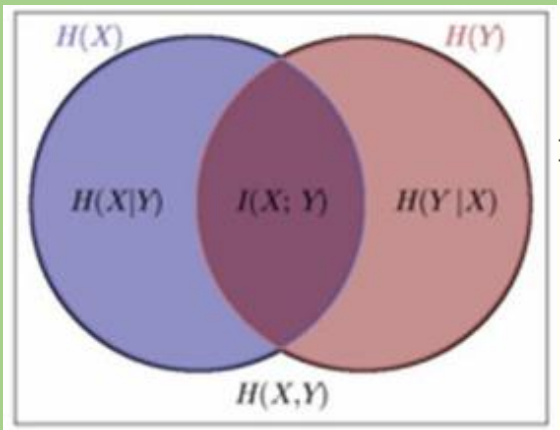
由于 $\sum_y P(y|x)=1$，得：

$$
P_w(y|x) = \frac{1}{Z_w(x)}\exp\left(\sum_{i=1}^{n} w_i f_i(x,y)\right)
$$

其中：

$$
Z_w(x) = \sum_y \exp\left(\sum_{i=1}^{n} w_i f_i(x,y)\right)
$$

然后求对偶问题的外部极大问题即可。



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

$$H(X|Y)=H(X,\ Y)-H(Y)$$

对偶式：

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

$$H(Y|X)=H(Y,\ X)-H(X)$$

$$\Rightarrow I(X,Y)=H(X)+H(Y)-H(X,Y)$$

$$H(X|Y)\le H(X),H(Y|X)\le H(Y)$$

熵 $H(X)=-\sum_x p(x)\log p(x)$

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

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

LR(逻辑回归)

定义

二项逻辑斯特回归是一种分类的概率模型，由条件概率 P（Y|X）表示，形式化为参数化的逻辑斯特分布，随机变量 X 的取值为实数，随机变量 Y 的值取 1 或 0，可以通过监督学习的方法来估计模型的参数。

回归的含义是最佳拟合，逻辑回归的思想是对分类边界建立回归公式。从而进行分类，使用的函数是 Sigmoid 函数。

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

损失函数

对数损失函数是逻辑回归的损失函数，在逻辑回归中，假设样本服从伯努利分布（0-1 分布），对样本的估计是用概率形式表达的，即：

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

然后去对数求最大似然估计，得到如下：

$$L(\theta) = \prod_{i=1}^{m} P(y_i \mid x_i;\theta) = \prod_{i=1}^{m} (h_\theta(x_i))^{y_i} (1-h_\theta(x_i))^{1-y_i}$$

而损失函数将似然函数取反，采用梯度下降使其最小。标准的对数损失函数如下：

$$J(\theta) = -L(\theta) = -\prod_{i=1}^{m} P(y_i \mid x_i;\theta) = -\prod_{i=1}^{m} (h_\theta(x_i))^{y_i} (1-h_\theta(x_i))^{1-y_i}$$

至于为什么能够将最大似然函数取负就等效为损失函数呢，主要是在于 $p(y \mid x)$ 服从指数分布。

模型概述

S 函数的输入 z 可以由下面的公式得出：

$$z = w_0 x_0 + w_1 x_1 + \cdots + w_n x_n$$

采用向量的形式可以得到

$$z = w^T x$$

可以看到 x 是出入数据，为了寻找到合适的 w，需要一些最优化理论方面的知识。

算法

梯度上升法

通过求解目标函数在当前点的导数获得迭代的方向。

随机梯度下降法(SGD)

随机梯度下降是一种对参数随着样本训练，一个一个的及时 update 的方式。常用于大规模训练集，当往往容易收敛到局部最优解。

优缺点

优点：计算代价不高，易于理解和实现

缺点：容易欠拟合，分类精度可能不高

适用数据类型：数值型和标称型数据

SVM(支持向量机)

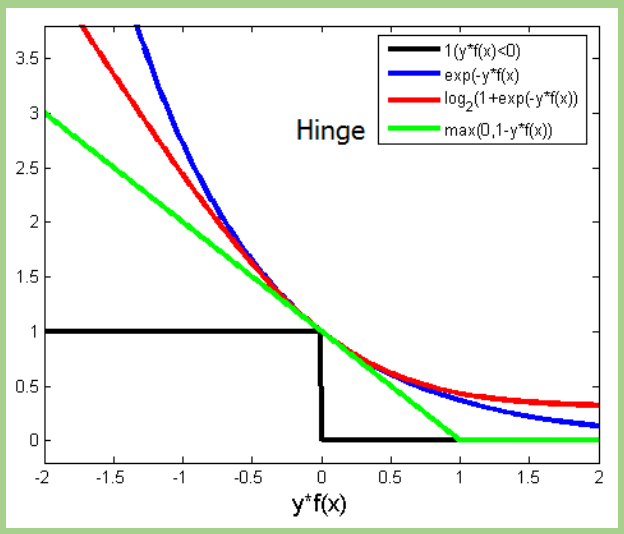
定义

支持向量机（SVM）是 90 年代中期发展起来的基于统计学习理论的一种机器学习方法，通过寻求结构化风险最小来提高学习机泛化能力，实现经验风险和置信范围的最小化，从而达到在统计样本量较少的情况下，亦能获得良好统计规律的目的。

通俗来讲，它是一种二类分类模型，其基本模型定义为特征空间上的间隔最大的线性分类器，即支持向量机的学习策略便是间隔最大化，最终可转化为一个凸二次规划问题的求解。

损失函数

SVM 使用的损失函数是合页损失函数，也就算 hinge 损失函数，也就下面的曲线 4



线性支持向量机通常还有另外一种解释，那就是最小化目标函数：

$$\sum_{i=1}^{N}[1-y_i(wx_i+b)]+\lambda\|w\|^2$$

其中 $\sum_{i=1}^{N}[1-y_i(wx_i+b)]=L(y(wx+b))$ 表示合页损失函数。这里要取正数，所以要在 $|[1-y_i(wx_i+b)]|$ 加一个绝对值使得变为正数。

定理 7.4 线性支持向量机原始最优化问题：

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i \tag{7.60}$$

$$\text{s.t.} \quad y_i(w\cdot x_i + b) \geqslant 1-\xi_i, \quad i=1,2,\cdots,N \tag{7.61}$$

$$\xi_i \geqslant 0, \quad i=1,2,\cdots,N \tag{7.62}$$

等价于最优化问题

$$\min_{w,b} \quad \sum_{i=1}^{N}[1-y_i(w\cdot x_i+b)]_+ + \lambda\|w\|^2 \tag{7.63}$$

证明 可将最优化问题 (7.63) 写成问题 (7.60) ～ (7.62)．令

$$1-y_i(w\cdot x_i+b)=\xi_i, \quad \xi_i \geqslant 0 \tag{7.64}$$

则 $y_i(w\cdot x_i+b)\geqslant 1$．于是 w, b, ξ_i 满足约束条件 (7.61) ～ (7.62)．由式 (7.64) 有，$[1-y_i(w\cdot x_i+b)]_+ = [\xi_i]_+ = \xi_i$，所以最优化问题 (7.63) 可写成

$$\min_{w,b} \quad \sum_{i=1}^{N}\xi_i + \lambda\|w\|^2$$

若取 $\lambda=\frac{1}{2C}$，则

$$\min_{w,b} \quad \frac{1}{C}\left(\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N}\xi_i\right)$$

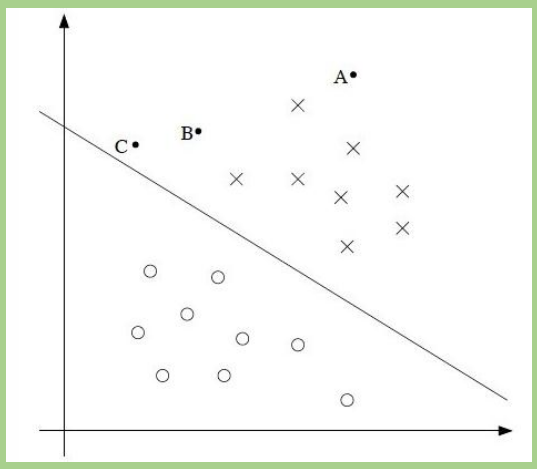
与式 (7.60) 等价．

在上述的情况中，**C 越大会造成过拟合。**

最大间隔分类器

假设、模型、符号

我们先假设所有数据都是线性可分的（之后会去除这个假设），通过下图直观地感受一下这个分类器。



虽然 A、B、C 三个点被分到了同一类，但我们认为点 A 的分类是最有把握的，因为它距离分类线最远（即间隔最大）。因此，这个分类器最基础的部分就是如何计算间隔。不过，在介绍间隔之前我们先看一下模型及其符号

$$h_{w,b}(x) = g(w^T x + b)$$
$$g(z) = 1 \text{ if } z \geq 0, \text{ and } g(z) = -1 \text{ otherwise}$$

可以看到它和逻辑回归的模型很像，不过 g(z)由逻辑函数变为了一个分段函数。另外，输出变量由{1,0}变为了{1,-1}，θTx 变为了 wTx+b，这些改动都是为了最后推导公司可以更加优雅。

Functional Margins

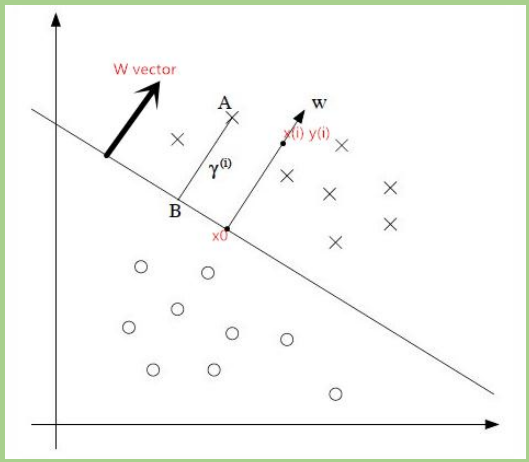
由最大间隔分类器的模型我们知道：wTx+b>0 ==> y=1；wTx+b<0 ==> y=-1；而 wTx+b=0 则被成为分隔超平面(separating hyperplane)。 当 wTx+b>0 时，wTx+b 越大分类为 1 的置信度越高；当 wTx+b<0 时，wTx+b 越小，分类为-1 的可信度也越高。因此，每个样本的 functional margins 被定义为：

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

这样，当每个样本都被正确分类时，functional margins 的值将总是大于 0，并且这个值越大代表分类的可信度越高。值得注意的是，我们可以对参数 w 和 b 同时放大/缩小任意相同的倍数，虽然不会影响分类的结果应因为 2wTx+2b=0 和 wTx+b=0 是一样的，但是 functional margins 的值却会发生变化。 整个训练集的 functional margins 则被定义为：

$$\hat{\gamma} = \min_{i=1,\ldots,m} \hat{\gamma}^{(i)}.$$

Geometric Margins



如图所示，geometric margins 就是数据点到分隔超平面的垂直距离，计算过程这里就忽略了，其中比较关键的一步是：分隔超平面 wTx+b=0 的法向量就是 w，推导如下：

$$x_0 = x^{(i)} - \gamma^{(i)} \frac{w}{\|w\|}$$

带入到 $W^T x + b = 0$ 中可以得到：

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

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

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

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

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

最终，每个样本的 geometric margins 公式为： 即 $y^{(i)}\gamma^{(i)}$

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

由这个公式首先我们可以得到 geometric margins 与 functional margins 的关系：

$$\gamma = \hat{\gamma}/||w||$$

相较于 functional margins, 即使对 w 和 b 同时放大/缩小任意相同的倍数, geometric margins 的值也不会改变。这一点很重要，会在之后的公式推导中用到。 整个训练集的 geometric

margins 为：

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

最大间隔分类器优化问题

我们的目标是找到能使 geometric margins 最大化的分隔超平面，转化为最优化问题就是：

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

由于||w||是一个非凸函数，将导致难以求解，因此我们需进行一些转化，首先，由之前的结论“随意同时缩放 w 和 b 不会改变 geometric margins 的大小”，我们可以让 functional margins 为 1

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

原始的最优化目标就变成了最大化 1/||w||，也相当于最小化||w||2，因此就转化为了凸优化问题：

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

可以用二次规划的软件求解。

拉格朗日对偶性

这一部分主要是最优化理论的内容，其核心思想是可以通过拉格朗日对偶性将原始最优化问题转化为它的对偶问题（这样做的目的是因为对偶问题可以提升求解的效率）。对偶函数是作为线性函数的逐点下确界，必然是一个凹函数，对偶问题是个凸规划问题。详细可看 点击这里

原始最优化问题

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

这个问题对应的广义拉格朗日公式为：

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

将原始问题进行适当的变形：

$$\theta_{\mathcal{P}}(w) = \max_{\alpha, \beta : \alpha_i \ge 0} \mathcal{L}(w, \alpha, \beta).$$

$$\theta_{\mathcal{P}}(w) = \begin{cases} f(w) & \text{if } w \text{ satisfies primal constraints} \\ \infty & \text{otherwise.} \end{cases}$$

$$\min_w \theta_{\mathcal{P}}(w) = \min_w \max_{\alpha, \beta : \alpha_i \ge 0} \mathcal{L}(w, \alpha, \beta),$$

可以看到如果 $g(w) > 0$，则要使得 $L(\alpha, \beta, w)$ 最大的的话，当然 α 要取一个很大的值。同理对于 $h(w) \neq 0$ 也是一样的。 引入对偶，使得原问题的求解变成了讨论“求原问题的下界”。

对偶最优化问题

$$\theta_{\mathcal{D}}(\alpha, \beta) = \min_w \mathcal{L}(w, \alpha, \beta).$$

$$\max_{\alpha, \beta : \alpha_i \ge 0} \theta_{\mathcal{D}}(\alpha, \beta) = \max_{\alpha, \beta : \alpha_i \ge 0} \min_w \mathcal{L}(w, \alpha, \beta).$$

对偶函数是一个凸规划问题，和原始最优化问题相比，对偶最优化问题只是颠倒了 max 和 min 的顺序，并且我们有：

$$d^* = \max_{\alpha, \beta : \alpha_i \ge 0} \min_w \mathcal{L}(w, \alpha, \beta) \le \min_w \max_{\alpha, \beta : \alpha_i \ge 0} \mathcal{L}(w, \alpha, \beta) = p^*.$$

KKT 条件

关于 KKT 条件可以看文档的介绍。为了让 d*=p*，即可以通过求对偶问题来解决原始优化问

题，需要满足以下假设：

f and the g_i's are convex
h_i's are affine
g_i are (strictly) feasible; this means that there exists some w so that $g_i(w) < 0$ for all i.

在这个假设下，一定会存在 w*、α*、β*使得 w*是原始问题的解并且 α*、β*是对偶问题的解。此外，还有一些有用的关系会成立，它们被称为：Karush-Kuhn-Tucker (KKT) conditions：

$$\begin{aligned}
\frac{\partial}{\partial w_i}\mathcal{L}(w^*,\alpha^*,\beta^*) &= 0, \quad i=1,\ldots,n \\
\frac{\partial}{\partial \beta_i}\mathcal{L}(w^*,\alpha^*,\beta^*) &= 0, \quad i=1,\ldots,l \\
\alpha_i^* g_i(w^*) &= 0, \quad i=1,\ldots,k \\
g_i(w^*) &\le 0, \quad i=1,\ldots,k \\
\alpha^* &\ge 0, \quad i=1,\ldots,k
\end{aligned}$$

其中 αi*gi(w)=0 被称为 KKT dual complementarity condition。它暗示了只有当 gi(w)=0 时才有可能有 αi>0，gi(w)被称为活动约束。这是 SVM 只有很少数 support vectors 的原因；在之后的 SMO 算法的收敛验证中也会用到这个条件。

SVM 求解

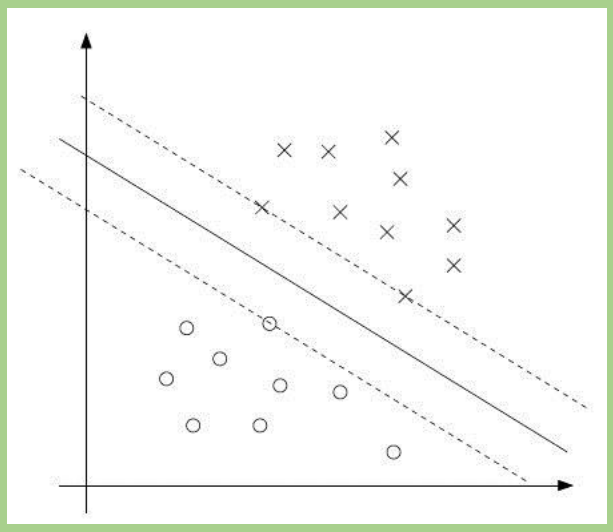
让我们回到前面讨论的最大间隔分类器：

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

利用拉格朗日对偶性，求解最大间隔分类器最优化问题的对偶问题，就是 SVM 了，让我们来进行推导：首先，将限制条件转化为可使用广义拉格朗日的标准形式让我们回到前面讨论的最大间隔分类器：

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

回忆一下 KKT dual complementarity condition，只有当 gi(w)=0 时才可能有 αi>0，同时 gi(w)=0 对应着距离分隔超平面最近的点（回顾推导最大间隔分类器最优化公式的过程），即下图中虚线穿过的三个点：



而这三个点就是 SVM 中的 SV：support vector。 将目前的最优化问题构造为拉格朗日的形式：

$$L(w,b,\alpha)=\frac{1}{2}\|w\|^2-\sum_{i=1}^{N}\alpha_i(y_i(wx_i+b)-1)$$
$$L(w,b,\alpha)=\frac{1}{2}\|w\|^2-\sum_{i=1}^{N}\alpha_i y_i(wx_i+b)+\sum_{i=1}^{N}\alpha_i \quad (1)$$

其中 $\alpha=(\alpha_1,\alpha_2,\alpha_3\mathrm{L}\ \alpha_N)^T$ 为拉格朗日乘子向量。根据拉格朗日对偶性，原始问题的对偶问题是极大极小问题：

$$\max_{a}\min_{w,b}L(w,b,\alpha)$$

所以为了得到对偶问题的解，需要先求 $L(w,b,\alpha)$ 对 w,b 的极小，然后再对 α 求极大。

(1)求 $\min_{w,b}L(w,b,\alpha)$

将拉格朗日函数 $L(w,b,\alpha)$ 分别对 w,b 求偏导数并为 0

$$\nabla_w L(w,b,\alpha)=w-\sum_{i=1}^{N}\alpha_i y_i wx_i=0$$
$$\nabla_b L(w,b,\alpha)=\sum_{i=1}^{N}\alpha_i y_i=0 \quad (2)$$

得到

$$w=\sum_{i=1}^{N}\alpha_i y_i wx_i$$
$$\sum_{i=1}^{N}\alpha_i y_i=0 \quad (3)$$

将(3)带入到(1)中得到

$$L(w,b,\alpha)=\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \mathrm{g} x_j)-\sum_{i=1}^{N}\alpha_i y_i((\sum_{j=1}^{N}\alpha_j y_j x_j)x_i+b)+\sum_{i=1}^{N}\alpha_i$$
$$=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \mathrm{g} x_j)+\sum_{i=1}^{N}\alpha_i \quad (4)$$

即

$$\min_{w,b} L(w,b,\alpha)=-\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \mathrm{g} x_j)+\sum_{i=1}^{N}\alpha_i$$

(2)求 $\min_{w,b} L(w,b,\alpha)$ 对 α 的极大，即是对偶问题

$$\begin{aligned} \max_{a} \quad & -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \mathrm{g} x_j)+\sum_{i=1}^{N}\alpha_i \\ s.t. \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\ & \alpha_i \geq 0, i=1,2,3\mathrm{L}\ N \end{aligned} \quad (5)$$

将(5)的目标函数由求极大转化为求极小，就得到下面等价的对偶优化问题：

$$\begin{aligned} \min_{a} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y_i y_j (x_i \mathrm{g} x_j)-\sum_{i=1}^{N}\alpha_i \\ s.t. \quad & \sum_{i=1}^{N}\alpha_i y_i = 0 \\ & \alpha_i \geq 0, i=1,2,3\mathrm{L}\ N \end{aligned}$$

最终，SVM 对应的最优化问题为：

$$w=\sum_{i=1}^{N}\alpha_i y^{(\mathrm{i})} x^{(\mathrm{i})}$$
$$b^{*}=y_{i}-\sum_{i=1}^{N}\alpha_i^{*}(y_i x_i)$$

最终得到的决策结果是：

$$f(x)=sign(\sum_{i=1}^{N}\alpha_i^{*} y_i (x \mathrm{g} x_i)+b^{*})=sign(\sum_{i=1}^{N}\alpha_i^{*} y_i <x,x_i>+b^{*})$$

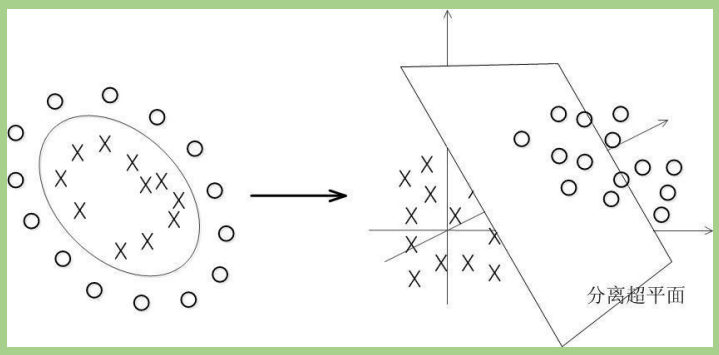
尖括号表示的是内积。计算的时候只需要为支持向量计算即可，所以减低了很大运算量。具体的证明过程可以查看 李航-统计学方法的 104 页 有详细的推导过程。

核函数

定义

通过使用恰当的核函数来替代内积，可以隐式得将非线性的训练数据映射到高维空间，而不增加可调参数的个数(当然，前提是核函数能够计算对应着两个输入特征向量的内积)。

在线性不可分的情况下，支持向量机首先在低维空间中完成计算，然后通过核函数将输入空间映射到高维特征空间，最终在高维特征空间中构造出最优分离超平面，从而把平面上本身不好分的非线性数据分开。如图所示，一堆数据在二维空间无法划分，从而映射到三维空间里划分：



核函数相当于把原来的分类函数：

$$f(x)=\sum_{i=1}^{n}\alpha_i y_i\langle x_i,x\rangle+b$$

映射为：

$$f(x)=\sum_{i=1}^{n}\alpha_i y_i\langle \phi(x_i),\phi(x)\rangle+b$$

其中的 α 可以通过求解对偶问题来得到：

$$\begin{aligned}&\max_{\alpha}\sum_{i=1}^{n}\alpha_i-\frac{1}{2}\sum_{i,j=1}^{n}\alpha_i\alpha_j y_i y_j\langle\phi(x_i),\phi(x_j)\rangle\\&s.t.,\alpha_i\geq 0,i=1,\ldots,n\\&\sum_{i=1}^{n}\alpha_i y_i=0\end{aligned}$$

对于一个如下图的分类问题可以看到在二维中是不可分的，但是我们知道有一个圆是可以将

其分开的，在二维空间中我们定义这个圆的方程：

$$a_1X_1 + a_2X_1^2 + a_3X_2 + a_4X_2^2 + a_5X_1X_2 + a_6 = 0 \quad (1)$$

可以看到，我们可以构造一个 5 维的空间

$$Z_1=X_1\ Z_2=X_1^{\ 2}\ Z_3=X_2\ Z_4=X_2^{\ 2}\ Z_5=X_1X_2$$

上式可以写为：

$$\sum_{i=1}^{5} a_iZ_i + a_6 = 0$$

于新的坐标 Z，这正是一个 hyper plane 的方程！也就是说，如果我们做一个映射 ϕ:R2→R5，将 X 按照上面的规则映射为 Z，那么在新的空间中原来的数据将变成线性可分的，从而使用之前我们推导的线性分类算法就可以进行处理了。这正是 Kernel 方法处理非线性问题的基本思想。对一个二维空间做映射，选择的新空间是原始空间的所有一阶和二阶的组合，得到了五个维度；如果原始空间是三维，那么我们会得到 19 维的新空间，这个数目是呈爆炸性增长的，这给 的计算带来了非常大的困难，而且如果遇到无穷维的情况，就根本无从计算了。所以就需要 Kernel 出马了。

不妨还是从最开始的简单例子出发，设两个向量 $x_1 = (\eta_1, \eta_2)^T \quad x_2 = (\xi_1, \xi_2)^T$，前面说的五维空间的映射，因此映射过后的内积为：

$$\langle \phi(x_1), \phi(x_2) \rangle = \eta_1\xi_1 + \eta_1^2\xi_1^2 + \eta_2\xi_2 + \eta_2^2\xi_2^2 + \eta_1\eta_2\xi_1\xi_2$$

另外，我们又注意到

$$(\langle x_1, x_2 \rangle + 1)^2 = 2\eta_1\xi_1 + \eta_1^2\xi_1^2 + 2\eta_2\xi_2 + \eta_2^2\xi_2^2 + 2\eta_1\eta_2\xi_1\xi_2 + 1$$

二者有很多相似的地方，实际上，我们只要把某几个维度线性缩放一下，然后再加上一个常数维度，具体来说，上面这个式子的计算结果实际上和映射。

$$\varphi(X_1, X_2) = (\sqrt{2}X_1, X_1^2, \sqrt{2}X_2, X_2^2, \sqrt{2}X_1X_2, 1)^T$$

之后的内积 $\langle \varphi(x_1), \varphi(x_2) \rangle$ 的结果是相等的（自己验算一下）。区别在于什么地方呢？一个是映射到高维空间中，然后再根据内积的公式进行计算；而另一个则直接在原来的低维空间中进行计算，而不需要显式地写出映射后的结果。回忆刚才提到的映射的维度爆炸，在前一种方法已经无法计算的情况下，后一种方法却依旧能从容处理，甚至是无穷维度的情况也没有问题。

我们把这里的计算两个向量在映射过后的空间中的内积的函数叫做核函数 (Kernel Function)，例如，在刚才的例子中，我们的核函数为：

$$\kappa(x_1, x_2) = (\langle x_1, x_2 \rangle + 1)^2$$

核函数就是为了简化运算的。它先直接在原来的空间中进行计算，然后再做平方什么的。时间复杂度为$O(n)$，相比于$O(n^2)$是快很多的。

核函数的满足条件

通过所说的核函数就是正定核，如果一个对称函数$K(x,z)$是定义在R上的对称函数，如果$K(x,z)$对应的 Gram 矩阵是半正定的，那么$K(x,z)$就是一个正定核。

常见核函数

- 线性核

$$K(x_1, x_2) = x_1 x_2 + c$$

- 多项式核函数

$$K(x_1, x_2) = (x_1 x_2 + c)^d$$

- Sigmod 核函数 感知器核

$$K(x_1, x_2) = \tanh(\beta(x_1 x_2) + c)$$

- RBF 径向基函数 也称**高斯核**

$$K(x_1, x_2) = \exp(-\frac{\|x_1 - x_2\|^2}{2\sigma^2})，\text{也记作} \exp(-\gamma\|x_1 - x_2\|^2)\ \ \gamma = \frac{1}{2\sigma^2}$$

高斯核函数可以映射到无穷维，原因是因为

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \cdots + \frac{x^n}{n!}$$

可以得到

$$K(x_1, x_2) = 1 + (-\frac{\|x_1 - x_2\|^2}{2\sigma^2}) + \frac{(-\frac{\|x_1 - x_2\|^2}{2\sigma^2})^2}{2!} + \cdots + \frac{(-\frac{\|x_1 - x_2\|^2}{2\sigma^2})^3}{3!} + \cdots + \frac{(-\frac{\|x_1 - x_2\|^2}{2\sigma^2})^n}{n!}$$

或者可以看如下(林轩田 SVM)

$$
\begin{aligned}
\text{when } \mathbf{x} = (x),\ K(x, x') &= \exp(-(x - x')^2) \\
&= \exp(-(x)^2)\exp(-(x')^2)\exp(2xx') \\
&\overset{\text{Taylor}}{=} \exp(-(x)^2)\exp(-(x')^2)\left(\sum_{i=0}^{\infty}\frac{(2xx')^i}{i!}\right) \\
&= \sum_{i=0}^{\infty}\left(\exp(-(x)^2)\exp(-(x')^2)\sqrt{\frac{2^i}{i!}}\sqrt{\frac{2^i}{i!}}(x)^i(x')^i\right) \\
&= \Phi(x)^T\Phi(x') \\
\text{with infinite dimensional } \Phi(x) &= \exp(-x^2)\cdot\left(1, \sqrt{\tfrac{2}{1!}}x, \sqrt{\tfrac{2^2}{2!}}x^2, \ldots\right)
\end{aligned}
$$

可以看出公式中的的泰勒展开式其实是 0-n 维的多项式核函数的和。

我们知道多项式核函数将低维数据映射到高维(维度是有限的)，那么 对于无限个 不同维的多项式核函数之和 的高斯核，其中也包括 无穷维度 的 多项式核函数

- 拉普拉斯核

$$K(x_1, x_2) = \exp(-\frac{\|x_1 - x_2\|^2}{2\sigma^2})$$

核函数的选择

一般用线性核和高斯核，也就是 Linear 核与 RBF 核，一般的使用还要结合自己的数据的情况：

需要注意的是需要对数据归一化处理，很多使用者忘了这个小细节然后。

一般情况下 RBF 效果是不会差于 Linear 但是时间上 RBF 会耗费更多，其他同学也解释过了

下面是吴恩达的见解：

1. 如果 Feature 的数量很大，跟样本数量差不多，这时候选用 LR 或者是 Linear Kernel 的 SVM。因为往往是线性可分的，可以采用线性的。

2. 如果 Feature 的数量比较小，样本数量一般，不算大也不算小，选用 SVM+Gaussian Kernel。特征数理少的话，求内积的时候需要消耗很多计算。

3. 如果 Feature 的数量比较小，而样本数量很多，需要手工添加一些 feature 变成第一种情况。

SVM 优缺点

优点：

1.可用于线性/非线性分类，也可以用于回归，泛化错误率低，计算开销不大，结果容易解释；

2.可以解决小样本情况下的机器学习问题，可以解决高维问题 可以避免神经网络结构选择和局部极小点问题。

3.SVM 是最好的现成的分类器，现成是指不加修改可直接使用。并且能够得到较低的错误率，SVM 可以对训练集之外的数据点做很好的分类决策。

缺点：对参数调节和和函数的选择敏感，原始分类器不加修改仅适用于处理二分类问题。

DT（决策树）

约于 1984 年提出的 CART 树开始，决策树已经开始研究了，到 ID3 和 C4.5 算法

定义

决策树模型是一种描述对实例进行分类或者回归的树形结构，它有结点(node)和有向边(directed edge)组成，结点有两种类型：内部结点(internal node)和叶结点(leaf node)。内部结点表示一个特征或者属性，叶结点表示一个类别。可以认为决策数是 if-then 规则的集合。

其主要优点是模型具有很好的**可读性，且分类速度快**；缺点是可能会产生**过度匹配**的问题(所以一般都会有决策树的剪枝过程)。决策树在学习时，利用训练数据，根据**损失函数最小化原则**建立决策树模型，其学习过程包括 3 个步骤：**特征选择、决策树生成和决策树修剪**。

根据数据的属性采用树状结构建立决策模型。决策树模型常常用来解决分类和回归问题。常见的算法包括 CART (Classification And Regression Tree)、ID3、C4.5、随机森林 (Random Forest) 等。

//----------

决策树的搜索策略是贪婪的（这也正是它效率高的原因），但它的可搜索空间 却能代表所有离散函数。ID3 算法中每个属性只考虑一次，所以规则的数目主要取 决于属性集规模和属性值的个数而非数据集大小，至于“局部最优，性能就没保障，除了理想贝叶斯，其他

的模型都有可能陷入局部最优值问题，所以性能还要看实 际的问题和应用。

损失函数

决策树损失函数通常是正则化的极大似然函数，它的学习策略是以损失函数为目标的最小化

决策树生成

通常采用的是信息增益、信息增益比、基尼指数这三个标准，分别对应的是下面的 ID3, C4.5 和 CART。

- ID3 算法容易产生过拟合，因为它只包含了树的生成，抗噪性差，训练例子中正例和反例的比例较难控制，ID3 是单变量决策树(在分枝节点上只考虑单个属性)，许多复杂概念的表达困难，属性相互关系强调不够，容易导致决策树中子树的重复或有些属性在决策树的某一路径上被检验多次。**ID3 相当于用极大似然法进行概率模型的选择。**
- C4.5 采用信息增益比来选择特征，克服了用信息增益选择属性时偏向选择取值多的属性的不足；在树构造过程中进行剪枝；能够完成对连续属性的离散化处理；能够对不完整数据进行处理。
- CART 的假设是二叉树，采用基尼指数作为特征选择的标准，内部特征的取值为“是”或者“否”。 CART，又名分类回归树，是在 ID3 的基础上进行优化的决策树，学习 CART 记住以下几个关键点：(1) CART 既能是分类树，又能是分类树；(2) **当 CART 是分类树时，采用 GINI 值作为节点分裂的依据；当 CART 是回归树时，采用样本的最小方差作为节点分裂的依据**。(3) CART 是一棵二叉树。

  给定下面的特征

ID	年龄	有工作	有自己的房子	信贷情况	类别
1	青年	否	否	一般	否
2	青年	否	否	好	否
3	青年	是	否	好	是
4	青年	是	是	一般	是
5	青年	否	否	一般	否
6	中年	否	否	一般	否
7	中年	否	否	好	否
8	中年	是	是	好	是
9	中年	否	是	非常好	是
10	中年	否	是	非常好	是
11	老年	否	是	非常好	是
12	老年	否	是	好	是
13	老年	是	否	好	是
14	老年	是	否	非常好	是
15	老年	否	否	一般	否

- **信息增益法**

随机变量 X 的熵定义为：

$$H(X)=-\sum_{i=1}^{n} p_i \log p_i$$

通常是以 2 或者 e 作为对数的底数

$$H(p)=-\sum_{i=1}^{n} p_i \log p_i$$

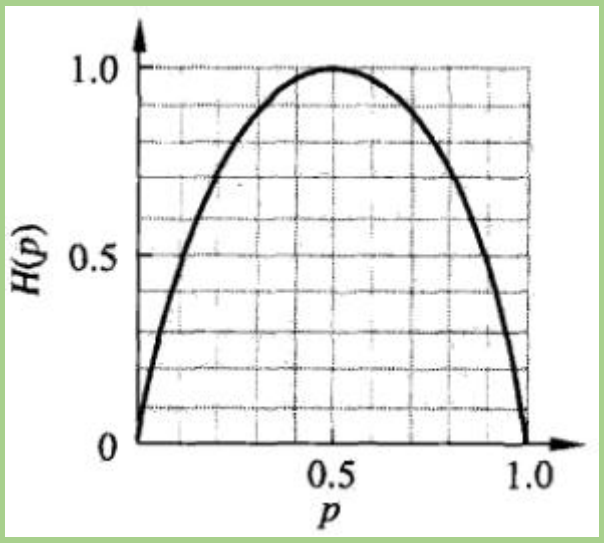
熵越大，随机系统的不确定性就越大，当随机变量只取 2 个值 0 和 1 的时候，

$$P(X=1)=p\,,\quad P(X=0)=1-p\,,\quad 0\leqslant p\leqslant 1$$

熵为

$$H(p)=-p\log_2 p-(1-p)\log_2(1-p)$$

熵随概率 P 的变化曲线如下图所示：



当熵和条件熵中的概率由数据估计（特别是极大似然估计）得到时，所对应的熵和条件熵分别成为经验熵和经验条件熵。

信息增益（information gain）表示得知特征 X 的信息而使得类 Y 的信息的不确定性减少的程度。

定义：特征 A 对训练数据集 D 的信息增益 g（D,A），定义集合 D 的**经验熵 H（D）**与特征 A 给定条件下 D 的**经验条件熵 H（D|A）**之差，即

$$g(D,A)=H(D)-H(D\mid A)$$

一般地，熵与条件熵之差成为**互信息**（mutual information），决策树学习中的信息增益等价于训练数据集中类与特征的**互信息**。

根据信息增益准则的特征选择方法是：对训练数据集（或子集）D，计算其每个特征的信息增益，并比较它们的大小，选择信息增益最大的特征。

信息增益计算算法：

(1) 计算数据集 D 的经验熵

$$H(D)=-\sum_{k=1}^{K}\frac{|C_k|}{|D|}\log_2\frac{|C_k|}{|D|}$$

（2）计算特征 A 对数据集 D 的经验条件熵 H(D|A)

$$H(D\mid A)=\sum_{i=1}^{n}\frac{|D_i|}{|D|}H(D_i)=-\sum_{i=1}^{n}\frac{|D_i|}{|D|}\sum_{k=1}^{K}\frac{|D_{ik}|}{|D_i|}\log_2\frac{|D_{ik}|}{|D_i|}$$

(3) 计算信息增益

$$g(D,A)=H(D)-H(D\mid A)$$

信息增益比的计算是

$$g_R(D,A)=\frac{g(D,A)}{H(D)}$$

对于上面的例子

$$H(D)=-\frac{9}{15}\log_2\frac{9}{15}-\frac{6}{15}\log_2\frac{6}{15}=0.971$$

解释：最后的类别中有 9 个是和 7 个否，所以这里选择的是 9/15 和 6/15

以 A1,A2,A3,A4 表示年龄，有工作，有自己的房子，信贷情况 4 个特征，然后计算信息增益：

$$\begin{aligned}g(D,A_1)&=H(D)-\left[\frac{5}{15}H(D_1)+\frac{5}{15}H(D_2)+\frac{5}{15}H(D_3)\right]\\&=0.971-\left[\frac{5}{15}\left(-\frac{2}{5}\log_2\frac{2}{5}-\frac{3}{5}\log_2\frac{3}{5}\right)\right.\\&\quad\left.+\frac{5}{15}\left(-\frac{3}{5}\log_2\frac{3}{5}-\frac{2}{5}\log_2\frac{2}{5}\right)+\frac{5}{15}\left(-\frac{4}{5}\log_2\frac{4}{5}-\frac{1}{5}\log_2\frac{1}{5}\right)\right]\\&=0.971-0.888=0.083\end{aligned}$$

年龄中有 5 个青年，5 个中年，5 个老年，所以是 3 个 5/15，青年中有 2 个标签是“是”,3 个是“否”;中年中是 3 和“是”，2 个“否”，老年中有 4 个“是”，1 个“否”。

依照这种思路计算可以得到

$$\begin{aligned}g(D,A_2)&=H(D)-\left[\frac{5}{15}H(D_1)+\frac{10}{15}H(D_2)\right]\\&=0.971-\left[\frac{5}{15}\times 0+\frac{10}{15}\left(-\frac{4}{10}\log_2\frac{4}{10}-\frac{6}{10}\log_2\frac{6}{10}\right)\right]=0.324\end{aligned}$$

$$\begin{aligned}g(D,A_3)&=0.971-\left[\frac{6}{15}\times 0+\frac{9}{15}\left(-\frac{3}{9}\log_2\frac{3}{9}-\frac{6}{9}\log_2\frac{6}{9}\right)\right]\\&=0.971-0.551=0.420\end{aligned}$$

基尼指数法

基尼指数：分类问题中，假设有 K 个类，样本点属于第 k 类的概率为 pk，则概率分布的基尼指数定义为

$$\mathrm{Gini}(p)=\sum_{k=1}^{K}p_k(1-p_k)=1-\sum_{k=1}^{K}p_k^2$$

对于给定的样本集合 D，其基尼指数为

$$\text{Gini}(D)=1-\sum_{k=1}^{K}\left(\frac{|C_k|}{|D|}\right)^2$$

对于一个二分类问题：

$$\text{Gini}(P)=2p(1-p)$$

如果样本集合 D 根据特征 A 是否取某一可能值 a 被分割成 D1 和 D2 两部分，则在特征 A 的条件下，集合 D 的基尼指数定义为

$$\text{Gini}(D,A)=\frac{|D_1|}{|D|}\text{Gini}(D_1)+\frac{|D_2|}{|D|}\text{Gini}(D_2) \quad D_1=\{(x,y)\in D\,|\,A(x)=a\}, \quad D_2=D-D_1$$

以 A1,A2,A3,A4 表示年龄，有工作，有自己的房子，信贷情况 4 个特征，然后计算基尼指数;

$$\text{Gini}(D,A_1=1)=\frac{5}{15}\left(2\times\frac{2}{5}\times\left(1-\frac{2}{5}\right)\right)+\frac{10}{15}\left(2\times\frac{7}{10}\times\left(1-\frac{7}{10}\right)\right)=0.44$$
$$\text{Gini}(D,A_1=2)=0.48$$
$$\text{Gini}(D,A_1=3)=0.44$$

A1=1 和 A1=3 表示年龄是青年和老年，他们的基尼指数**最小，**都可以作为切分点

$$\text{Gini}(D,A_2=1)=0.32$$
$$\text{Gini}(D,A_3=1)=0.27$$

同样可以求得特征 4 的基尼指数

$$\text{Gini}(D,A_4=1)=0.36$$
$$\text{Gini}(D,A_4=2)=0.47$$
$$\text{Gini}(D,A_4=3)=0.32$$

决策树的剪枝

因为我们在生成树的过程中，如果没有剪枝的操作的话，就会长成**每一个叶都是单独的一类的样子**。这样对我们的训练集是完全拟合的，但是对测试集则是非常不友好的，**泛化能力**不行。因此，我们要减掉一些枝叶，使得模型泛化能力更强。根据剪枝所出现的时间点不同，分为预剪枝和后剪枝。预剪枝是在决策树的生成过程中进行的；后剪枝是在决策树生成之后进行的。

1.对于 ID3 和 C4.5 的生成的决策树：

决策树的损失函数为：

$$C_\alpha(T)=\sum_{t=1}^{T}N_tH_t(T)+\alpha|T| \qquad (1)$$

其中经验熵为

$$H_t(T) = -\sum_k \frac{N_{ik}}{N_t} \log \frac{N_{tk}}{N_t} \qquad (2)$$

在损失函数中，将(1)右端第一项记作

$$C(T) = \sum_{t=1}^{|T|} N_t H_t(T) = -\sum_{t=1}^{|T|} \sum_{k=1}^{K} N_{tk} \log \frac{N_{tk}}{N_t} \quad (3)$$

这时有

$$C_\alpha(T) = C(T) + \alpha |T| \quad (4)$$

C(T)为预测误差（如）基尼指数。|T|为树的叶子节点数。

过程如下：

输入：生成算法产生的整个树 T，参数 α；
输出：修剪后的子树 T_α.
（1）计算每个结点的经验熵.
（2）递归地从树的叶结点向上回缩.

设一组叶结点回缩到其父结点之前与之后的整体树分别为 T_B 与 T_A，其对应的损失函数值分别是 $C_\alpha(T_B)$ 与 $C_\alpha(T_A)$，如果

$$C_\alpha(T_A) \leqslant C_\alpha(T_B)$$

则进行剪枝，即将父结点变为新的叶结点.
（3）返回 (2)，直至不能继续为止，得到损失函数最小的子树 T_α.

2.对于 CART 树的剪枝

$$C_\alpha(T) = C(T) + \alpha |T| \quad (4)$$

C(T)为预测误差（如）基尼指数。|T|为树的叶子节点数。这个公式是上面的，但是对于这里的 CART 同样适用。

常用的剪枝方法有 Reduced-Error Pruning(**REP**,错误率降低剪枝） Pesimistic-Error Pruning(**PEP**,悲观错误剪枝）Cost-Complexity Pruning（**CCP**，代价复杂度剪枝)。这里 CART 采用 CCP（代价复杂度）剪枝方法。比较如下

比较项目和枝剪方法	CCP	REP	PEP	MEP
独立剪枝集	CV 方式:不需要	需要	不需要	不需要
剪枝方式	自底向上	自底向上	自顶向下	自底向上
误差估计	使用 CV 或标准误差	利用剪枝集	使用连续性校正	基于 m2概率估计
计算复杂性(n2非叶节点数)	O(n^2)	O(n)	O(n)	O(n)

CCP 剪枝算法分为两个步骤：

1.对于完全决策树 T 的每个非叶结点计算 α 值，α 值的定义如下 循环剪掉具有最小 α 值的子树，直到剩下根节点。在该步可得到一系列的剪枝树 {T0，T1，T2.....Tm}，其中 T0 为原有的完全决策树，Tm 为根结点，Ti+1 为对 Ti 进行剪枝的结果

2.从子树序列中，根据真实的误差估计选择最佳决策树。

该算法为子树 Tt 定义了**代价（cost）和复杂度（complexity）**，以及一个可由用户设置的衡量代价与复杂度之间关系的参数 α，其中，代价指在剪枝过程中因子树 Tt 被叶节点替代而增加的错分样本，复杂度表示剪枝后子树 Tt 减少的叶结点数，**α 则表示剪枝后树的复杂度降低程度与代价间的关系**，定义为

$$\alpha = \frac{R(t) - R(T_t)}{|N_1| - 1}$$

其中，

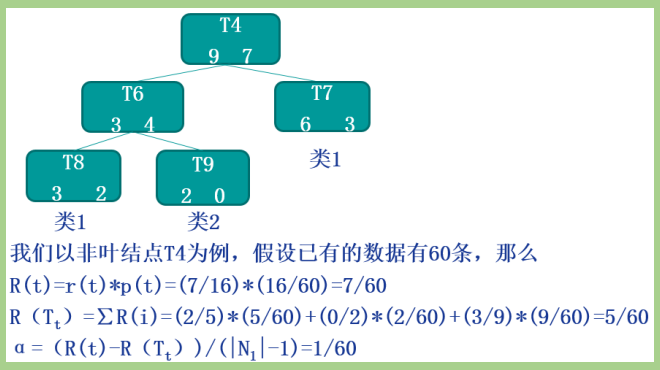
|N1|：子树 Tt 中的叶节点数；

R（t）：结点 t 的错误代价，计算公式为 R（t）=r（t）*p（t），

r（t）为结点 t 的错分样本率，p（t）为落入结点 t 的样本占所有样本　　的比例；

R（Tt）：子树 Tt 错误代价，计算公式为 R（Tt）=∑R(i)，i 为子树 Tt 的叶节点。

例子：



决策树处理缺失值

- 采用抛弃缺失值

抛弃极少量的缺失值的样本对决策树的创建影响不是太大。但是如果属性缺失值较多或是关键属性值缺失,创建的决策树将是不完全的,同时可能给用户造成知识上的大量错误信息,所以抛弃缺失值一般不采用。只有在数据库具有极少量的缺失值同时缺失值不是关键的属性值时,且为了加快创建决策树的速度,才采用抛弃属性缺失值的方式创建决策树。

- 补充缺失值

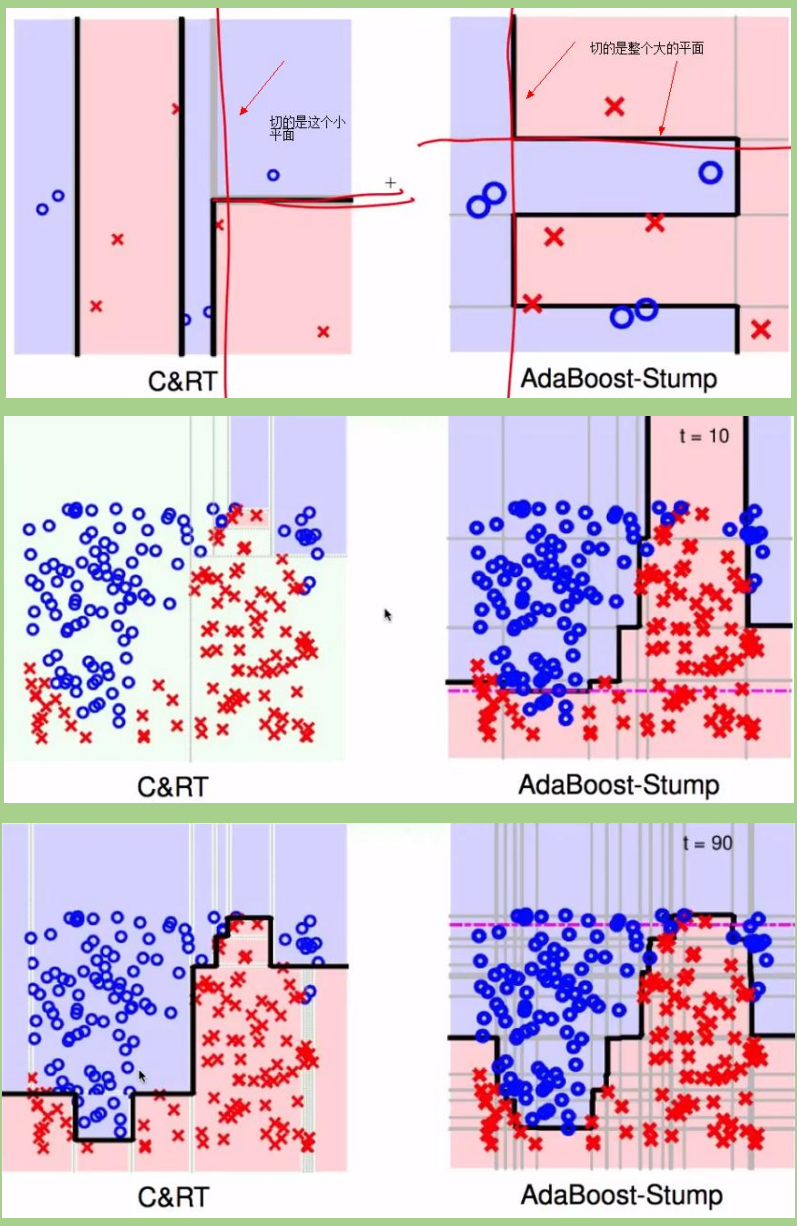
缺失值较少时按照我们上面的补充规则是可行的。但如果数据库的数据较大,缺失值较多(当然,这样获取的数据库在现实中使用的意义已不大,同时在信息获取方面基本不会出现这样的数据库),这样根据填充后的数据库创建的决策树可能和根据正确值创建的决策树有很大变化。

- 概率化缺失值

对缺失值的样本赋予该属性所有属性值的概率分布,即将缺失值按照其所在属性已知值的相对概率分布来创建决策树。用系数 F 进行合理的修正计算的信息量,F=数据库中缺失值所在的属性值样本数量去掉缺失值样本数量/数据库中样本数量的总和,即 F 表示所给属性具有已知值样本的概率。

决策树与 Adaboost 切分的区别

来自机器学习台大。



DT 切分的时候是在条件下切的，Adaboost 是把整个平面都切分开了。DT 比 Adaboost 有效率。

决策树的优缺点

优点：

1)决策树**易于理解和实现**，人们在在学习过程中不需要使用者了解很多的背景知识，这同时是它的能够直接体现数据的特点，只要通过解释后都有能力去理解决策树所表达的意义。

2) 对于决策树，数据的准备往往是简单或者是不必要的，而且能够同时处理数据型和常规型属性，在相对短的时间内能够对大型数据源做出可行且效果良好的结果。

3）计算量不是很大

4) 可以处理连续和种类字段

5）决策树可以清晰的显示哪些字段比较重要

6) 可以处理**类别资料**

缺点：

1) 对连续性的字段比较难预测。

2) 对有时间顺序的数据，需要很多预处理的工作。

3) 当类别太多时，错误可能就会增加的比较快。

4) 一般的算法分类的时候，只是根据一个字段来分类。

5）容易过拟合

6）当检验数据集中对象的属性有缺失值，树的性能可能有问题

Random Forest(随机森林)

定义

是用随机的方式建立一个森林，森林里面有很多的决策树组成，随机森林的每一棵决策树之间是没有关联的。在得到森林之后，当有一个新的输入样本进入的时候，就让森林中的每一棵决策树分别进行一下判断，看看这个样本应该属于哪一类（对于分类算法），然后看看哪一类被选择最多，就预测这个样本为那一类。

随机森林的生成

在建立每一棵决策树的过程中，有两点需要注意 - **采样与完全分裂**。首先是两个随机采样的过程，random forest 对输入的数据要进行**行、列的采样**。对于行采样，采用有放回的方式，也就是在采样得到的样本集合中，可能有重复的样本。假设输入样本为 N 个，那么采样的样本也为 N 个。这样使得在训练的时候，每一棵树的输入样本都不是全部的样本，使得相对不容易出现 over-fitting。然后进行列采样，从 M 个 feature 中，选择 m 个(m << M)。之后就是对采样之后的数据使用完全分裂的方式建立出决策树，这样决策树的某一个叶子节点要么是无法继续分裂的，要么里面的所有样本的都是指向的同一个分类。一般很多的决策树算法都一个重要的步骤 - 剪枝，但是这里不这样干，由于之前的两个随机采样的过程保证了随机性，所以就**算不剪枝，也不会出现 over-fitting**。

Adaboost（自适应提升）

定义

Adaboost 是一种**基于加法模型和前向分步算法的迭代学习算法**，其核心思想是针对同一个训练集训练不同的分类器(弱分类器)，然后把这些弱分类器集合起来，构成一个更强的最终分类器 (强分类器)。其算法本身是通过改变数据分布来实现的，它根据每次训练集之中每个样本的分类是否正确，以及上次的总体分类的准确率，来确定每个样本的权 值。将修改过权值的新数据集送给下层分类器进行训练，最后将每次训练得到的分类器最后融合起来，作为最后的决策分类器。使用 adaboost 分类器可以排 除一些不必要的训练数据特徵，并将关键放在关键的训练数据上面。

目前，对 adaboost 算法的研究以及应用大多集中于分类问题，同时近年也出 现了一些

在回归问题上的应用。就其应用 adaboost 系列主要解决了：两类问题、 多类单标签问题、多类多标签问题、大类单标签问题，回归问题。它用全部的训练样本进行学习。

损失函数

损失函数是指数函数：$L(y, f(x)) = \exp[-yf(x)]$

优化的目标是使得到的α_m和G_m在训练数据集T上指数损失函数最小，即：

$$(\alpha_m, G_m(x)) = \arg\min_{\alpha,G} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \alpha G(x_i))]$$

又等价于：

$$G_m(x)) = \arg\min_{\alpha,G} \sum_{i=1}^{N} w_{mi} \exp[-y_i \alpha G(x_i))]$$

其中的$w_{mi} = \exp[-y_i f_{m-1}(x_i)]$，不依赖于$\alpha, G$但是依赖于$f_{m-1}(x)$，所以每次都会变。

详细见参考 李航 P145

算法过程

给定训练集：$(x_1, y_1), \cdots, (x_N, y_N)$,其中$y_i \in \{-1,1\}$，表示$x_i$的训练正确的类别标签，$i = 1, \cdots, N$,训练集上的初始样本分布：

$$D_1(i) = \frac{1}{N}$$

对$t = 1, \cdots, T$，在分布寻找D_t上寻找具有最小错误率的弱分类器$h_t : \mathrm{X} \to \{-1,1\}$,其中，某弱分类在分布$D_t$上的错误率为：

$$\varepsilon_t = P_{D_t}(h_t(x) \neq y_i)$$

计算改弱分类器的权重系数：

$$\alpha_t = \frac{1}{2} \ln(\frac{1-\varepsilon_t}{\varepsilon_t})$$

更新训练样本的分布：

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

最后的强分类器为：

$$H_{final}(x) = sign(\sum_{t=1}^{T} \alpha_t h_t(x))$$

记

$$\varepsilon_t = \frac{1}{2} - \gamma_t$$

由于弱分类器的错误率比随机猜测的好，所以

$$\gamma_t > 0$$

则训练误差为：

$$R_{tr}(H_{final}) \le \exp(-2\sum_{t=1}^{T} \gamma_t^2)$$

可以看到训练误差的上界将不断减少。

注意：其中 $\alpha_i h_i(x)$ 相当于 SVM 中的 wx 中的 margin，这个是越大越好，希望 $D = \exp(-\alpha_t y_i h(x_i))$ 越小越好。就是每个点的权重越来越小，就是让每个点的 Margin 越来越大，可以的达到较大的 Margin 理论的效果。Adaboost 是存在过拟合的，至今没有得到一个很好的证明，只是没有过拟合的那么厉害而已，可以看一下 margin 里面的理论来解释。

算法优缺点

优点

1)Adaboost 是一种有很高精度的分类器

2)可以使用各种方法构建子分类器，Adaboost 算法提供的是框架

3)当使用简单分类器时，计算出的结果是可以理解的，而且弱分类器构造极其简单

4)简单，不用做特征筛选

5)不易出现 overfitting(过度拟合)

缺点

1)容易受到噪声干扰，这也是大部分算法的缺点

2)训练时间过长

3)执行效果依赖于弱分类器的选择

GBDT(梯度提升树)

1999 年由 Jerome Friedman 提出，最早用于 CTR，最早见于 Yahoo

定义

目前 GBDT 有两个不同的描述版本，两者各有支持者，读文献时要注意区分。残差版本把 GBDT 说成一个残差迭代树，认为每一棵回归树都在学习前 N-1 棵树的残差，之前我写的 GBDT 入门教程主要在描述这一版本，ELF 开源软件实现中用的也是这一版本。Gradient 版本把 GBDT 说成一个梯度迭代树，使用梯度下降法求解，认为每一棵回归树在学习前 N-1 棵树的梯度下降值，之前 leftnoteasy 的博客中介绍的为此版本，umass 的源码实现中用的则是这一版本（准确的说是 LambdaMART 中的 MART 为这一版本，MART 实现则是前一版本）。

对 GBDT 无基础的朋友可以先分别看一下前面两篇博文教程。总的来说两者相同之处在于，都是迭代回归树，都是累加每颗树结果作为最终结果（Multiple Additive Regression Tree），每棵树都在学习前 N-1 棵树尚存的不足，从总体流程和输入输出上两者是没有区别的；两者的不同主要在于每步迭代时，是否使用 Gradient 作为求解方法。前者不用 Gradient 而是用残差----残差是全局最优值，Gradient 是局部最优方向*步长，即前者每一步都在试图让结果变成最好，后者则每步试图让结果更好一点。

两者优缺点。看起来前者更科学一点--有绝对最优方向不学，为什么舍近求远去估计一个局部最优方向呢？原因在于灵活性。前者最大问题是，由于它依赖残差，cost function 一般固定为反映残差的均方差，因此很难处理纯回归问题之外的问题。而后者求解方法为梯度下降，只要可求导的 cost function 都可以使用，所以用于排序的 LambdaMART 就是用的后者。

GBDT(Gradient Boosting Decision Tree) 又叫 MART（Multiple Additive Regression Tree），是一种迭代的决策树算法，是一种组合算法，也叫做梯度提升回归树（gradient boosting regression tree）。该算法由多棵决策树组成，所有树的结论累加起来做最终答案。它在被提出之初就和 SVM 一起被认为是泛化能力较强的算法。**GBDT 中的树是回归树（不是分类树）**，GBDT 用来做回归预测，调整后也可以用于分类。GBDT 的思想使其具有天然优势可以发现多种有区分性的特征以及特征组合。业界中，Facebook 使用其来自动发现有效的特征、特征组合，来作为 LR 模型中的特征，以提高 CTR 预估（Click-Through Rate Prediction）的准

确性；GBDT 在淘宝的搜索及预测业务上也发挥了重要作用。

Boosting

提升方法采用**加法模型**（即基函数的线性组合）与**前向分布算法**，adaboost 可以表示为 boosting 的**前向分布算法**(Forward stagewise additive modeling)的一个特例实际为贪心算法，boosting 最终可以表示为：

$$f(\mathrm{x}) = w_0 + \sum_{m=1}^{M} w_m \psi_m(\mathrm{x})$$

不同提升方法

基于不同的损失函数提升方法有不同的形式，主流的还是 Adaboost

Name	Loss	Derivative	f^*	Algorithm
Squared error	$\frac{1}{2}(y_i - f(\mathbf{x}_i))^2$	$y_i - f(\mathbf{x}_i)$	$\mathbb{E}[y\|\mathbf{x}_i]$	L2Boosting
Absolute error	$\|y_i - f(\mathbf{x}_i)\|$	$\mathrm{sgn}(y_i - f(\mathbf{x}_i))$	$\mathrm{median}(y\|\mathbf{x}_i)$	Gradient boosting
Exponential loss	$\exp(-\tilde{y}_i f(\mathbf{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\mathbf{x}_i))$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	AdaBoost
Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$	$\frac{1}{2}\log\frac{\pi_i}{1-\pi_i}$	LogitBoost

GBDT 推导广义上来讲，所谓的 Gradient Boosting 其实就是在更新的时候选择梯度下降的方向来保证最后的结果最好，一些书上讲的“残差” 方法其实就是 L2Boosting，因为它所定义的残差其实就是 L2 Boosting 的 Derivative，接下来我们着重讲一下弱回归器(不知道叫啥了，自己编的)是决策树的情况，也就是 GBDT

算法过程

总之所谓 Gradient 就是去拟合 Loss function 的梯度，或者说的是，每棵树是从先前所有树的残差中来学习。利用的是当前模型中损失函数的负梯度值将其作为新的弱回归树加入到总的算法中即可。Gradient Boost 与传统的 Boost 有着很大的区别，它的**每一次计算都是为了减少上一次的残差(residual)，而为了减少这些残差，可以在残差减少的梯度(Gradient)方向上建立一个新模型**。所以说，在 Gradient Boost 中，每个新模型的建立是为了使得先前模型残差往梯度方向减少，与传统的 Boosting 算法对正确、错误的样本进行加权有着极大的区别。算法如下：(PS:这里的是基于残差的版本的)

对于一个典型的回归问题，采用前向分布算法：

$$
\begin{aligned}
&f_0(x) = 0 \\
&f_m(x) = f_{m-1}(x) + T(x;\Theta_m), m = 1, 2, \cdots M \\
&f_M(x) = \sum_{m=1}^{M} T(x;\Theta_m)
\end{aligned}
$$

在前向分布算法的第 m 步，给定当前模型 $f_{m-1}(x)$，需求解

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i;\Theta_m))$$

得到Θ_m，即第 m 棵树的参数，当采用平方误差损失函数时

$$
\begin{aligned}
&L(y, f(x)) = (y - f(x))^2 \\
&L(y, f_{m-1}(x) + T(x;\Theta_m)) = (y - f_{m-1}(x) - T(x;\Theta_m))^2
\end{aligned}
$$

其损失变为

$$(r - T(x,\Theta_m))^2$$

其中

$$r = y - f_{m-1}(x_i)$$

是拟合当前模型的残差，对于平方损失函数，拟合的就是残差；对于一般损失函数（梯度下降），拟合的就是残差的近似值

对于非残差版本的：

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute
   $$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$
   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute
   $$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$
   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

或者参考李航博士的统计学习方法中的

提升树利用加法模型与前向分步算法实现学习的优化过程. 当损失函数是平方损失和指数损失函数时，每一步优化是很简单的. 但对一般损失函数而言，往往每一步优化并不那么容易. 针对这一问题，Freidman 提出了梯度提升（gradient boosting）算法. 这是利用最速下降法的近似方法，其关键是利用损失函数的负梯度在当前模型的值

$$-\left[\frac{\partial L(y, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

作为回归问题提升树算法中的残差的近似值，拟合一个回归树.

算法 8.4（梯度提升算法）

输入：训练数据集 $T=\{(x_1,y_1),(x_2,y_2),\cdots,(x_N,y_N)\}$，$x_i \in \mathcal{X} \subseteq \mathbf{R}^n$，$y_i \in \mathcal{Y} \subseteq \mathbf{R}$；损失函数 $L(y, f(x))$；

输出：回归树 $\hat{f}(x)$.

（1）初始化

$$f_0(x)=\arg\min_c \sum_{i=1}^{N} L(y_i, c)$$

（2）对 $m=1,2,\cdots,M$

（a）对 $i=1,2,\cdots,N$，计算

$$r_{mi}=-\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x)=f_{m-1}(x)}$$

（b）对 r_{mi} 拟合一个回归树，得到第 m 棵树的叶结点区域 R_{mj}，$j=1,2,\cdots,J$

（c）对 $j=1,2,\cdots,J$，计算

$$c_{mj}=\arg\min_c \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i)+c)$$

（d）更新 $f_m(x)=f_{m-1}(x)+\sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$

（3）得到回归树

$$\hat{f}(x)=f_M(x)=\sum_{m=1}^{M}\sum_{j=1}^{J} c_{mj} I(x \in R_{mj})$$

■

算法第 1 步初始化，估计使损失函数极小化的常数值，它是只有一个根结点的树. 第 2 (a) 步计算损失函数的负梯度在当前模型的值，将它作为残差的估计. 对于平方损失函数，它就是通常所说的残差；对于一般损失函数，它就是残差的近似值. 第 2 (b) 步估计回归树叶结点区域，以拟合残差的近似值. 第 2 (c) 步利用线性搜索估计叶结点区域的值，使损失函数极小化. 第 2 (d) 步更新回归树. 第 3 步得到输出的最终模型 $\hat{f}(x)$.

$$\min_{\eta} \quad \frac{1}{N}\sum_{n=1}^{N}((y_n - s_n) - \eta g_t(\mathbf{x}_n))^2$$

1. $(\sum_{n=1}^{N} g_t(\mathbf{x}_n)(y_n - s_n)) \cdot (\sum_{n=1}^{N} g_t^2(\mathbf{x}_n))$
2. $(\sum_{n=1}^{N} g_t(\mathbf{x}_n)(y_n - s_n)) \ / \ (\sum_{n=1}^{N} g_t^2(\mathbf{x}_n))$

防止过拟合

1 、控制 tree 的棵树，即迭代次数 M。An optimal value of M is often selected by monitoring prediction error on a separate validation data set.

2、控制 shrink，Empirically it has been found that using small learning rates (such as) yields dramatic improvements in model' s generalization ability over gradient boosting without shrinking ().

3、随机采样迭代。类 bagging 方法。经验来说随机采样率 f 在 0.5<=f<=0.8 比较合适。即可以帮助避免过拟合又可以提高训练速度。

4、控制叶子节点中的最少样本个数。

5、惩罚树的复杂性（复杂性定义为叶子数占树所有节点的比例），用一个后验剪枝算法来对 loss 和树的复杂度进行联合优化，该方法为去掉那些降低 loss 幅度小于指定阈值的分支。

6.加入正则，譬如特征的正向正则或者负向正则（譬如，在分裂的时候，除了满足最小平方差之外，要保证左子树的 lable 平均值小于右子树的 lable 平均值，反之为负向正则）合

Xgboost

介绍

Xgboost 是一种基于树的学习模型，是目前最好的开源的额 boostd tree，比常见的工具包要快大约 10 倍左右，可以跨平台，支持多种语言，在比赛和工业界也是用的很多

算法过程

无论如何，还是要从涉及到理论推导的，下面从简单的开始说起：一个机器学习模型优化的

问题主要是如下：

$$Obj(\Theta) = L(\Theta) + \Omega(\Theta)$$

其中，L 是损失函数，Ω 用来衡量模型拟合训练数据的好坏程度；称之为正则项，用来衡量学习到的模型的复杂度。

提升树的算法可以看成是由很多树组成的树组成：

$$\hat{y}_i = \sum_{k=1}^{K} f_k(x_i), f_k \in F$$

目标函数为：

$$Obj = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \sum_{k=1}^{K} \Omega(f_k)$$

可以用前向分布算法（forward stagewise algorithm）。因为学习的是加法模型，如果能够从前往后，每一步只学习一个基函数及其系数（结构），逐步逼近优化目标函数，那么就可以简化复杂度。这一学习过程称之为 Boosting。具体地，我们从一个常量预测开始，每次学习一个新的函数，过程如下：

$$\begin{aligned}
\hat{y}_i^0 &= 0 \\
\hat{y}_i^1 &= f_1(x_i) = \hat{y}_i^0 + f_1(x_i) \\
\hat{y}_i^2 &= f_1(x_i) + f_2(x_i) = \hat{y}_i^1 + f_2(x_i) \\
&\cdots \\
\hat{y}_i^t &= \sum_{k=1}^{t} f_k(x_i) = \hat{y}_i^{t-1} + f_t(x_i)
\end{aligned}$$

在第步，模型对的预测为：$\hat{y}_i^t = \hat{y}_i^{t-1} + f_t(x_i)$，其中为这一轮我们要学习的函数（决策树）。这个时候目标函数可以写为：

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} l(y_i, \hat{y}_i^t) + \sum_{i=i}^{t} \Omega(f_i) \\
&= \sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{t-1} + f_t(x_i)\right) + \Omega(f_t) + constant
\end{aligned}$$

假设我们的损失函数为平方损失，则有如下的目标函数：

$$\begin{aligned}
Obj^{(t)} &= \sum_{i=1}^{n} \left(y_i - (\hat{y}_i^{t-1} + f_t(x_i))\right)^2 + \Omega(f_t) + constant \\
&= \sum_{i=1}^{n} \left[2(\hat{y}_i^{t-1} - y_i) f_t(x_i) + f_t(x_i)^2\right] + \Omega(f_t) + constant
\end{aligned}$$

通过泰勒展开可以得到：

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

由于函数中的常量在函数最小化的过程中不起作用，因此我们可以得到

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

一颗生成好的决策树，假设其叶子节点个数为T，该决策树是由所有叶子节点对应的值组成的向量$w \in R^T$，以及一个把特征向量映射到叶子节点索引（Index）的函数$q: R^d \to \{1, 2, \cdots, T\}$组成的。因此，策树可以定义为$f_t(x) = w_{q(x)}$。

决策树的复杂度可以由正则项$\Omega(f_t) = \gamma T + \frac{1}{2}\lambda \sum_{j=1}^{T} w_j^2$来定义，即决策树模型的复杂度由生成的树的叶子节点数量和叶子节点对应的值向量的L2范数决定。

定义集合$I_j = \{i | q(x_i) = j\}$为所有被划分到叶子节点j的训练样本的集合。等式(5)可以根据树的叶子节点重新组织为T个独立的二次函数的和：

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

定义$G_j = \sum_{i \in I_j} g_i$，$H_j = \sum_{i \in I_j} h_i$，则等式(6)可写为：

$$Obj^{(t)} = \sum_{j=1}^{T}\left[G_i w_j + \frac{1}{2}(H_i + \lambda) w_j^2\right] + \gamma T$$

假设树的结构是固定的，即函数$q(x)$确定，令函数$Obj^{(t)}$的一阶导数等于0，即可求得叶子节点j对应的值为：

$$w_j^* = -\frac{G_j}{H_j + \lambda}$$

此时，目标函数的值为

$$Obj = -\frac{1}{2}\sum_{j=1}^{T}\frac{G_j^2}{H_j + \lambda} + \gamma T$$

综上，为了便于理解，单颗决策树的学习过程可以大致描述为：

1. 枚举所有可能的树结构 q
2. 用上等式 Obj 为每个计算其对应的分数，分数越小说明对应的树结构越好
3. 根据上一步的结果，找到最佳的树结构，用等式 Wj 树的每个叶子节点计算预测值

然而，可能的树结构数量是无穷的，所以实际上我们不可能枚举所有可能的树结构。通常情况下，我们采用贪心策略来生成决策树的每个节点。

1. 从深度为 0 的树开始，对每个叶节点枚举所有的可用特征

2. 针对每个特征，把属于该节点的训练样本根据该特征值升序排列，通过线性扫描的方式来决定该特征的最佳分裂点，并记录该特征的最大收益（采用最佳分裂点时的收益）

3. 选择收益最大的特征作为分裂特征，用该特征的最佳分裂点作为分裂位置，把该节点生长出左右两个新的叶节点，并为每个新节点关联对应的样本集

4. 回到第 1 步，递归执行到满足特定条件为止

计算每次分裂的收益：

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

EM 算法

定义

最大期望算法（Expectation Maximization Algorithm，又译期望最大化算法），是一种迭代算法，用于含有隐变量（latent variable）的概率参数模型的最大似然估计或极大后验概率估计。EM 经常用在机器学习和计算机视觉的数据聚类（Data Clustering）领域。

步骤

EM 算法经过两个步骤交替进行计算：

第一步是计算期望（E），利用对隐藏变量的现有估计值，计算其最大似然估计值；

第二步是最大化（M），最大化在 E 步上求得的最大似然值来计算参数的值。

M 步上找到的参数估计值被用于下一个 E 步计算中，这个过程不断交替进行。

总体来说，EM 的算法流程如下：

1.初始化分布参数

2.重复直到收敛：

E 步骤：估计未知参数的期望值，给出当前的参数估计。

M 步骤：重新估计分布参数，以使得数据的似然性最大，给出未知变量的期望估计。

将 EM 算法要优化的目标看作是以 Q, θ 为参数的函数，那么 **E-step 就是保持 θ 不变，优化 Q； M-step 就是保持 Q 不变优化 θ。**

算法过程

这里的 Y 是观测遍历，Z 是隐变量

E 步：

$\theta^{(i)}$为第i次迭代参数θ的估计值，在$i+1$次迭代的E步，计算

$$Q\left(\theta,\theta^{(i)}\right)=E_Z[\log P(Y,Z\mid\theta)\mid Y,\theta^{(i)}]=\sum_z \log P(Y,Z\mid\theta)P(Z\mid Y,\theta^{(i)})$$

M 步：

求使得$Q\left(\theta,\theta^{(i)}\right)$极大化的$\theta$，确定第$i+1$次的参数估计值$\theta^{(i+1)}$

$$\theta^{(i+1)}=\arg\max_\theta Q\left(\theta,\theta^{(i)}\right)$$

详细的公式如下，过程中要用到 Jensen 不等式来进行求解：

首先我们要求得是式(1)的最大化，但是由于是和的对数（难求），所以转化为(2),然后到(3)。

$$\begin{aligned}
\sum_i \log p(x^{(i)};\theta) &= \sum_i \log \sum_{z^{(i)}} p(x^{(i)},z^{(i)};\theta) && (1)\\
&= \sum_i \log \sum_{z^{(i)}} Q_i(z^{(i)})\frac{p(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})} && (2)\\
&\geq \sum_i \sum_{z^{(i)}} Q_i(z^{(i)}) \log \frac{p(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})} && (3)
\end{aligned}$$

得到（3）可以发现：

$\sum_{z^{(i)}} Q_i(z^{(i)})[\frac{p(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}]$是$[\frac{p(x^{(i)},z^{(i)};\theta)}{Q_i(z^{(i)})}]$的期望，又因为$\sum_z Q_i(z^{(i)})=1$，可以得到

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

首先第一个问题，在 Jensen 不等式中说到，当自变量 X 是常数的时候，等式成立。而在这里，即

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

得到：

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

我们推出了在固定参数θ后，使下界拉升的 Q(z)的计算公式就是**后验概率**，解决了 Q(z)如何选择的问题。这一步就是 E 步，**建立 L(θ)的下界**。接下来的 M 步，就是在给定 Q(z)后，调整 θ，去极大化 L(θ)的下界 J（在固定 Q(z)后，下界还可以调整的更大）。那么一般的 EM 算法的步骤如下：

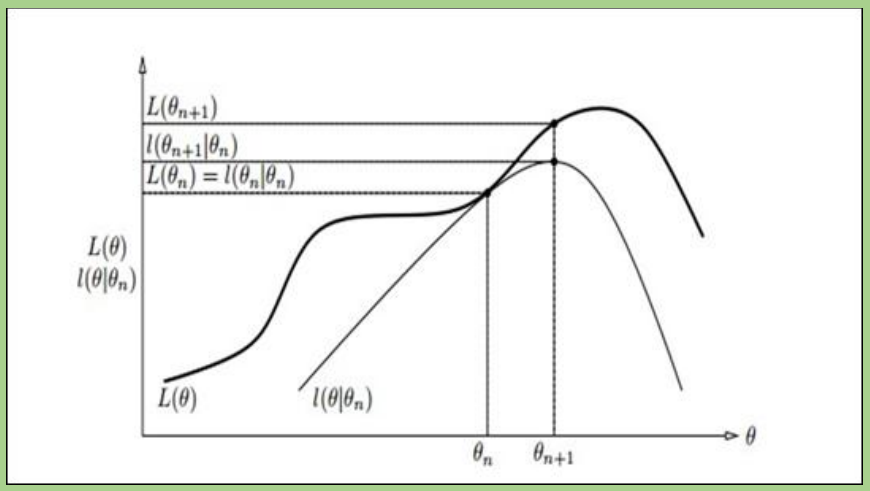
E 步骤：根据参数初始值或上一次迭代的模型参数来计算出隐性变量的后验概率，其实就是隐性变量的期望。作为隐藏变量的现估计值：

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

M 步骤：将似然函数最大化以获得新的参数值：

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

具体的逼近下界以及求更新θ的过程如下：



可见 l(θ)的下界为 φ(θn,θ)，φ(θn,θ)值越大， l(θ)的下界也将越大(其中的 θn 为已知变量)，具体 l(θ)和 φ(θn,θ)的关系可从图 2 可看出：φ(θn,θ)增长的方向也是 l(θ)增长的方向，也就是任意增长 φ(θn,θ)值的 θ，都将使 l(θ)下界增大，从而迭代使 l(θ)逼近理想值。当然，在此最好的 φ(θn,θ)值的 θ 便是，max(φ(θn,θ))处的 θ，计算如下：

$$
\begin{aligned}
\theta_{n+1} &= \arg\max_{\theta}(\varphi(\theta_n,\theta)) \\
&= \arg\max_{\theta}(l(\theta_n)+\sum_z P(x|X,\theta_n)\cdot\log(\frac{P(X|z,\theta)\cdot P(z|\theta)}{P(z|X,\theta_n)\cdot P(X|\theta_n)})) \\
&= \arg\max_{\theta}(\sum_z P(x|X,\theta_n)\cdot\log(P(X|z,\theta)\cdot P(z|\theta))---(\text{去掉}\theta\text{无关向}) \\
&= \arg\max_{\theta}(\sum_z P(x|X,\theta_n)\cdot\log(\frac{P(X|z,\theta)}{p(z,\theta)}\cdot\frac{P(z|\theta)}{P(\theta)})) \\
&= \arg\max_{\theta}(\sum_z P(z|X,\theta_n)\cdot\log(P(X,z|\theta)) \\
&= \arg\max_{\theta}(E_{Z|X,\theta_n}(\log(P(X,z|\theta)))
\end{aligned}
$$

记：$F(\theta)=E_{Z|X,\theta_n}(\log(P(X,z|\theta))$

收敛性

PS：EM 算法的敛散性，由计算中的下界迭代，可很清晰的看到，EM 算法收敛，但可能收敛于局部最优解，证明不述。EM 算法对初始值的选择比较敏感，使用时要注意。

应用和推广

用于 GMM 或者称为 MoG（高斯混合模型）MoNB (混合贝叶斯) 以及聚类等无监督学习的生成模型

1.

混合贝叶斯模型的 EM 推导结果与朴素贝叶斯的极大似然估计十分相似。区别在于在朴素贝叶斯中，类别属性是已知的，而在混合贝叶斯中，类别属性是未知的，因而其多了一个概率表示w

使用注意

虽然 EM 算法很好很强大，可以很好的拟合混合模型，但是在上面的 MoG 和 MoNB 模型中，要想得到一个较好的结果，需要有一个前提条件，即足够的数据量， **m>>n**， m 为样本数目， n 为每个样本的维度。 当 m ≈ n 或者 m ≪ n 时， 再应用 MoG 模型甚至是 Gaussian 模型，就会出现问题。以数据符合高斯分布为例，那么使用极大似然估计，可以得到参数：

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

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

因为样本的数目小于维度，所以得到的方差Σ是奇异矩阵，所谓奇异矩阵，即特征值为0，不满秩的矩阵。所以|Σ|1/2 = 0，因而不能写出其概率密度函数。为什么会这样呢？这相当于线性方程组求解，未知数的个数比方程的数目多，因而不能完全的求出所有未知数。解决问题的方法是对方差做一些限制，但是会导致不同维度的相关性丢失。而**因子分析方法**就是用来解决上述的问题的，但是任然不能拟合出完整的协方差矩阵。

EL（集成学习）

集成学习法（Ensemble Learning）

① 将多个分类方法聚集在一起，以提高分类的准确率。

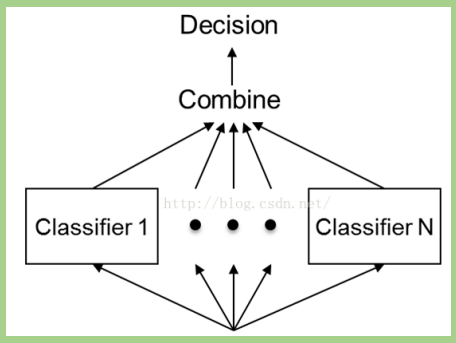
（这些算法可以是不同的算法，也可以是相同的算法。）

② 集成学习法由训练数据构建一组基分类器，然后通过对每个基分类器的预测进行投票来进行分类

③ 严格来说，集成学习并不算是一种分类器，而是一种分类器结合的方法。

④ 通常一个集成分类器的分类性能会好于单个分类器

⑤ 如果把单个分类器比作一个决策者的话，集成学习的方法就相当于多个决策者共同进行一项决策。



集成学习方法图

要掌握集成学习法，我们会提出以下两个问题：

1）怎么训练每个算法？

2）怎么融合每个算法？

因此，bagging 方法和 boosting 方法应运而生

Bagging

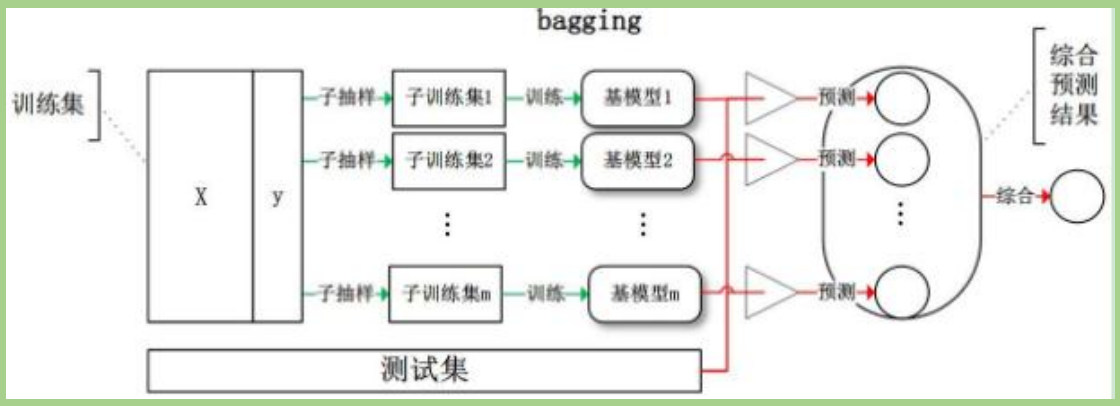
bagging 是一种用来提高学习算法准确度的方法，这种方法通过构造一个预测函数系列，然后以一定的方式将它们组合成一个预测函数。Bagging 要求“不稳定”（不稳定是指数据集的小的变动能够使得分类结果的显著的变动）的分类方法。比如：决策树，神经网络算法。

① Bagging 又叫自助聚集(Bootstrap Aggregating)，是一种根据均匀概率分布从数据中重复抽样（有放回）的技术。

② 每个抽样生成的自助样本集上，训练一个基分类器；对训练过的分类器进行投票，将测试样本指派到得票最高的类中。

③ 每个自助样本集都和原数据一样大

④ 有放回抽样，一些样本可能在同一训练集中出现多次，一些可能被忽略。



图解如上所示

例子：

X 表示一维属性，Y 表示类标号（1 或-1）测试条件：当 x<=k 时，y=？；当 x>k 时，y=？；k 为最佳分裂点。下表为属性 x 对应的唯一正确的 y 类别

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	-1	1	1

现在进行 5 轮随机抽样，结果如下

x	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
y	1	1	1	-1	-1	-1	-1	-1	1	1

第一轮：k=0.75，x<=k , y = -1; x > k , y = 1; 准确率 = 70%										
x	0.1	0.4	0.5	0.6	0.6	0.7	0.8	0.8	0.9	0.9
y	1	-1	-1	-1	-1	-1	-1	-1	1	1

第二轮：k=0.65，x<=k，y = -1; x > k，y = 1; 准确率 = 60%										
x	0.1	0.2	0.3	0.4	0.5	0.8	0.9	1	1	1
y	1	1	1	-1	-1	-1	1	1	1	1

第三轮：k=0.35，x<=k，y = 1; x > k，y = -1; 准确率 = 90%										
x	0.1	0.2	0.3	0.4	0.4	0.5	0.7	0.7	0.8	0.9
y	1	1	1	-1	-1	-1	-1	-1	-1	1

第四轮：k=1，x<=k，y = 1; x > k，y = -1; 准确率 = 50%										
x	0.1	0.1	0.2	0.5	0.6	0.7	0.7	0.8	0.9	0.9
y	1	1	1	-1	-1	-1	-1	-1	1	1

第五轮：k=0.4，x<=k，y = 1; x > k，y = -1; 准确率 = 70%										
x	0.1	0.1	0.2	0.5	0.6	0.6	0.6	1	1	1
y	1	1	1	-1	-1	-1	-1	-1	-1	-1

每一轮随机抽样后，都生成一个分类器 然后再将五轮分类融合

轮	k	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
1	0.75	-1	-1	-1	-1	-1	-1	-1	1	1	1
2	0.65	-1	-1	-1	-1	-1	-1	1	1	1	1
3	0.35	1	1	1	-1	-1	-1	-1	-1	-1	-1
4	1	1	1	1	1	1	1	1	1	1	1
5	0.4	1	1	1	1	-1	-1	-1	-1	-1	-1
和	—	1	1	1	-1	-3	-3	-1	1	1	1
符号	—	1	1	1	-1	-1	-1	-1	1	1	1
实际类	—	1	1	1	-1	-1	-1	-1	-1	1	1

对比符号和实际类，我们可以发现：在该例子中，Bagging 使得准确率可达 90%

由此，总结一下 bagging 方法：

① **Bagging 通过降低基分类器的方差**，改善了泛化误差

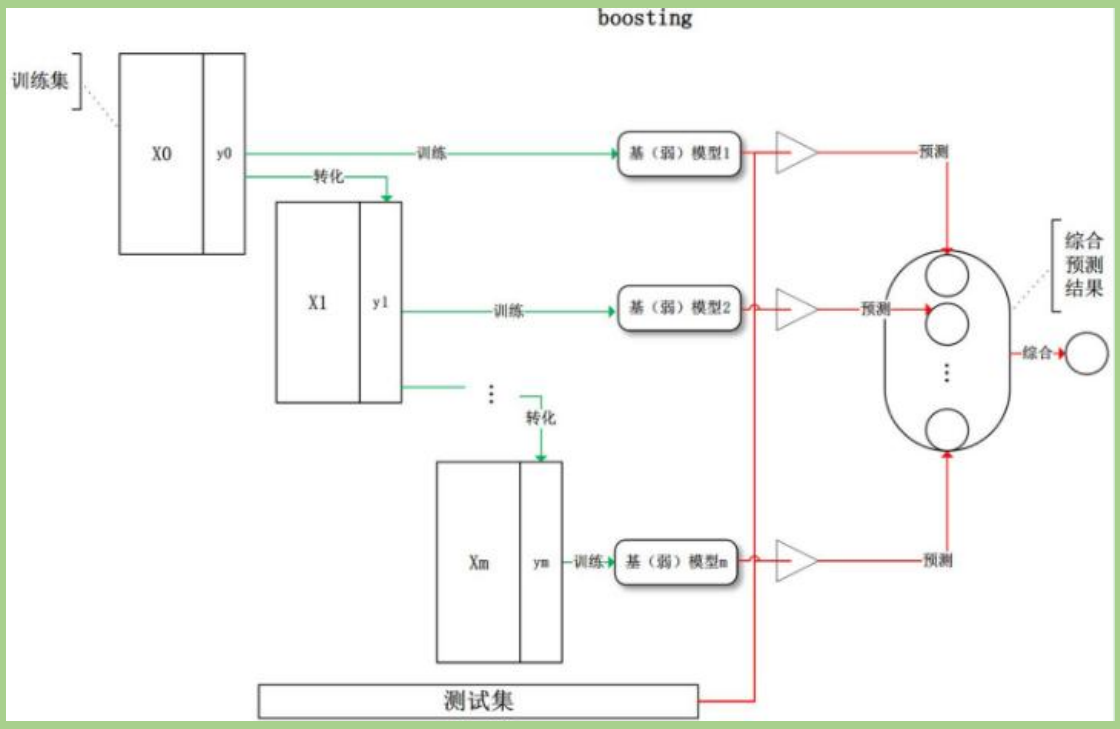
② 其性能依赖于基分类器的稳定性；如果基分类器不稳定，bagging 有助于降低训练数据的随机波动导致的误差；如果稳定，则集成分类器的误差主要由基分类器的偏倚引起

③ 由于每个样本被选中的概率相同，因此 bagging 并不侧重于训练数据集中的任何特定实例

Boosting

Boosting 方法是一种用来提高弱分类算法准确度的方法,这种方法通过构造一个预测函数系列,然后以一定的方式将他们组合成一个预测函数。Boosting 是一种提高任意给定学习算法准确度的方法。它的思想起源于 Valiant 提出的 PAC (Probably Approxi mately Correct) 学习模型。

Booting 的训练过程为阶梯状，基模型按次序一一进行训练（实现上可以做到并行），基模型的训练集按照某种策略每次都进行一定的转化。对所有基模型预测的结果进行线性综合产生最终的预测结果：



① boosting 是一个迭代的过程，用于自适应地改变训练样本的分布，使得基分类器聚焦在那些很难分的样本上

② boosting 会给每个训练样本赋予一个权值，而且可以再每轮提升过程结束时自动地调整权值。开始时，所有的样本都赋予相同的权值 1/N，从而使得它们被选作训练的可能性都一样。根据训练样本的抽样分布来抽取样本，得到新的样本集。然后，由该训练集归纳一个分类器，并用它对原数据集中的所有样本进行分类。每轮提升结束时，更新训练集样本的权值。增加被错误分类的样本的权值，减小被正确分类的样本的权值，这使得分类器在随后的迭代中关注那些很难分类的样本。

Bagging 和 Boosting 的对比

模型对比

1）样本选择上：

Bagging：训练集是在原始集中有放回选取的，从原始集中选出的各轮训练集之间是独立的。

Boosting：每一轮的训练集不变，只是训练集中每个样例在分类器中的权重发生变化。而权值是根据上一轮的分类结果进行调整。

2）样例权重：

Bagging：使用均匀取样，每个样例的权重相等

Boosting：根据错误率不断调整样例的权值，错误率越大则权重越大。

3）预测函数：

Bagging：所有预测函数的权重相等。

Boosting：每个弱分类器都有相应的权重，对于分类误差小的分类器会有更大的权重。

4）并行计算：

Bagging：各个预测函数可以并行生成

Boosting：各个预测函数只能顺序生成，因为后一个模型参数需要前一轮模型的结果。

偏差、方差对比

- Bagging

对于 bagging 来说，每个基模型的权重等于 1/m 且期望近似相等（子训练集都是从原训练集中进行子抽样），故我们可以进一步化简得到：

$$
\begin{aligned}
&\mathrm{E}(F) = \gamma * \sum_{i}^{m} E(f_i) \\
&= \frac{1}{m} * m * \mu \\
&= \mu \\
&\mathrm{Var}(F) = m^2 * \gamma^2 * \sigma^2 * \rho + m * \gamma^2 * \sigma^2 * (1-\rho) \\
&= m^2 * \frac{1}{m^2} * \sigma^2 * \rho + m * \frac{1}{m^2} * \sigma^2 * (1-\rho) \\
&= \sigma^2 * \rho + \frac{\sigma^2 * (1-\rho)}{m}
\end{aligned}
$$

根据上式我们可以看到，**整体模型的期望近似于基模型的期望**，这也就意味着整体模型的偏

差和基模型的偏差近似。同时，整体模型的方差小于等于基模型的方差（当相关性为 1 时取等号），随着基模型数（m）的增多，整体模型的方差减少，从而防止过拟合的能力增强，模型的准确度得到提高。但是，模型的准确度一定会无限逼近于 1 吗？并不一定，当基模型数增加到一定程度时，方差公式第二项的改变对整体方差的作用很小，防止过拟合的能力达到极限，这便是准确度的极限了。另外，在此我们还知道了为什么 bagging 中的基模型一定要为强模型，否则就会导致整体模型的偏差度低，即准确度低。

Random Forest 是典型的基于 bagging 框架的模型，其在 bagging 的基础上，进一步降低了模型的方差。Random Fores 中基模型是树模型，在树的内部节点分裂过程中，不再是将所有特征，而是随机抽样一部分特征纳入分裂的候选项。这样一来，基模型之间的相关性降低，从而在方差公式中，第一项显著减少，第二项稍微增加，整体方差仍是减少。

- Boosting

对于 boosting 来说，基模型的训练集抽样是强相关的，那么模型的相关系数近似等于 1，故我们也可以针对 boosting 化简公式为：

$$\mathrm{E}(F) = \gamma * \sum_{i}^{m} E(f_i)$$

$$\begin{aligned}\mathrm{Var}(F) &= m^2 * \gamma^2 * \sigma^2 * \rho + m * \gamma^2 * \sigma^2 * (1-\rho) \\ &= m^2 * \gamma^2 * \sigma^2 * 1 + m * \gamma^2 * \sigma^2 * (1-1) \\ &= m^2 * \gamma^2 * \sigma^2\end{aligned}$$

通过观察整体方差的表达式，我们容易发现，若基模型不是弱模型，其方差相对较大，这将导致整体模型的方差很大，即无法达到防止过拟合的效果。因此，boosting 框架中的基模型必须为弱模型。

因为基模型为弱模型，导致了每个基模型的准确度都不是很高（因为其在训练集上的准确度不高）。随着基模型数的增多，整体模型的期望值增加，更接近真实值，因此，整体模型的准确度提高。但是准确度一定会无限逼近于 1 吗？仍然并不一定，因为训练过程中准确度的提高的主要功臣是整体模型在训练集上的准确度提高，而随着训练的进行，整体模型的方差变大，导致防止过拟合的能力变弱，最终导致了准确度反而有所下降。

基于 boosting 框架的 Gradient Tree Boosting 模型中基模型也为树模型，同 Random Forrest，我们也可以对特征进行随机抽样来使基模型间的相关性降低，从而达到减少方差的效果。

- 知乎说法

Bagging 对样本重采样，对每一重采样得到的子样本集训练一个模型，最后取平均。由于子样本集的相似性以及使用的是同种模型，因此各模型有近似相等的 bias 和 variance（事实上，各模型的分布也近似相同，但不独立）。由于$E[\frac{\sum X_i}{n}]=E[X_i]$所以 bagging 后的 bias 和单个子模型的接近，一般来说不能显著降低 bias。另一方面，若各子模型独立，则有$Var(\sum X_i/n)=Var(X_i)/n$，此时可以显著降低 variance。若各子模型完全相同，则$Var(\sum X_i/n)=Var(X_i)$，此时不会降低 variance。bagging 方法得到的各子模型是有一定相关性的，属于上面两个极端状况的中间态，因此可以一定程度降低 variance。为了进一步降低 variance，Random forest 通过随机选取变量子集做拟合的方式 de-correlated 了各子模型（树），使得 variance 进一步降低。（用公式可以一目了然：设有 i.d.的 n 个随机变量，方差记为σ^2，两两变量之间的相关性为ρ，则$\sum X_i/n$的方差为$\rho*\sigma^2+(1-\rho)*\sigma^2/n$，bagging 降低的是第二项，random forest 是同时降低两项。详见 ESL p588 公式 15.1）。

boosting 从优化角度来看，是用 forward-stagewise 这种贪心法去最小化损失函数$L(y,\sum_i \alpha_i f_i(x))$，例如，常见的 AdaBoost 即等价于用这种方法最小化 exponential loss：$L(y,f_i(x))=\exp(-yf(x))$。所谓 forward-stagewise，就是在迭代的第 n 步，求解新的子模型 f(x)及步长 a（或者叫组合系数），来最小化$L(y,f_{n-1}(x)+\alpha f(x))$，这里$f_{n-1}(x)$是前 n-1 步得到的子模型的和。因此 boosting 是在 sequential 地最小化损失函数，其 bias 自然逐步下降。但由于是采取这种 sequential、adaptive 的策略，各子模型之间是强相关的，于是子模型之和并不能显著降低 variance。所以说 boosting 主要还是靠降低 bias 来提升预测精度。

结论：bagging 主要是减少 variance，而 boosting 主要是减少 bias 来提升精度

模型选择

交叉检验

保留交叉验证

保留交叉验证(hold-out cross validation or simple cross validation)的做法如下：

1) 将标注数据集随机切分为 Strain(如 70%)和 Scv(如 30%)

2) 在 Strain 上训练模型；

3) 在 Scv 上进行测试；

4) 去测试误差最小的那个模型。

通过在非训练集上进行测试，我们得到了一个对模型更好的估计。在实际使用过程中，模型将在全部的数据集上重新训练，以利用更多的数据，达到更好的效果。

该方法的劣势在于分出过多的数据用来测试，对于标注数据难得的实际问题来说，这是不能容忍的。因而产生了如下的改进方法，成为 k 重交叉检验。

K 重交叉验证

k 重交叉检验(k-fold cross validation)，做法如下：

1) 将标注数据集随机平均切分为 k 份；

2) 对于每一份来说，

   2.1) 以该份为测试集，其余份为训练集；

   2.2) 在训练集上得到模型；

   2.3) 在测试集上得到误差结果，这样就对每个样例都有一个预测结果。

3) 计算误差结果；

4) 取误差最小的模型

常用的做法是取 k=10。极端的做法取 k=m， m 为样例数，这样就变成了留一交叉验证(leave-one-out cross validation)。

VC 维

对一个模型集合$\mathcal{H}$来说，它的 VC 维，记为 VC($\mathcal{H}$)，是其能够分散的最大集合的大小。它

度量的是类$\mathcal{H}$的学习能力。VC 维反映了函数集的学习能力，VC 维越大则学习机器越复杂（容量越大），遗憾的是，目前尚没有通用的关于任意函数集 VC 维计算的理论，只对一些特殊的函数集知道其 VC 维。例如在 N 维空间中线性分类器和线性实函数的 VC 维是 N+1。形象的解释可以参考：

http://blog.163.com/wangfuwangbaiyang@126/blog/static/900913252012681083246/

VC 维推理 1

对于模型集合$\mathcal{H}$，令 d=VC($\mathcal{H}$),那么至少以有$1-\sigma$的概率，对于模型集合中的所有假设 h 来说有：

$$|\varepsilon(\mathrm{h})-\hat{\varepsilon}(\mathrm{h})|\leq \mathrm{O}(\sqrt{\frac{d}{m}\log\frac{m}{d}+\frac{1}{m}\log\frac{1}{\sigma}})$$

从而，得到在至少$1-\sigma$的概率下，有

$$\varepsilon(\hat{\mathrm{h}})\leq\hat{\varepsilon}(\mathrm{h}^*)+\mathrm{O}(\sqrt{\frac{d}{m}\log\frac{m}{d}+\frac{1}{m}\log\frac{1}{\sigma}})$$

可以看到，**当一个模型的 VC 维是有限的时候，随着样本数目变大，训练误差与泛化误差将会一致收敛**。

一般情况下，为了使模型可以达到较好的效果，需要的样本数必须与模型的 VC 维在同一数量级。

VC 维与 SVM

SVM 通过核函数将数据映射到高维空间，相应的 VC 维会变大。但是其有最大间隔分类器的假设，而对于最大间隔分类器来说，其 VC 维不依赖于 x 的维度。当训练样本给定时，分类间隔越大，则对应的分类超平面集合的 VC 维就越小。(分类间隔的要求，对 VC 维的影响)。对于最大间隔为γ的分类器来说，其在γ半径内的数据点数目维 k，则分类器的 VC 维服从如下公式：

$$VC(\mathrm{H})\leq\left[\frac{k^2}{\gamma}\right]+1$$

SVM 算法会自动选择一个具有较小 VC 维的假设类，这样反而降低了 VC 维，使数据量变得相对更加充分，提高了模型的效果。

特征工程

特征提取和特征选择是 DimensionalityReduction（降维）的两种方法，针对于 the curse of dimensionality(维灾难)，都可以达到降维的目的。但是这两个有所不同。

特征提取（Feature Extraction）:Creatting a subset of new features by combinations of the exsiting features.也就是说，特征抽取后的新特征是原来特征的一个**映射**。

特征选择（Feature Selection）:choosing a subset of all the features(the ones more informative)。也就是说，特征选择后的特征是原来特征的一个**子集**。

特征选择

特征选择属于特征工程中的一个分支，关于特征工程可以参考 Evernote 笔记。

一般认为特征工程中包括特征构建、特征提取、**特征选择**三个部分，特征提取就是下面有一章说的降维。

在机器学习的实际应用中，特征数量可能较多，其中可能存在不相关的特征，特征之间也可能存在相关性，容易导致如下的后果：

（1） 特征个数越多，分析特征、训练模型所需的时间就越长，模型也会越复杂。

（2） 特征个数越多，容易引起“维度灾难”，其推广能力会下降。

（3）特征个数越多，容易导致机器学习中经常出现的特征稀疏的问题，导致模型效果下降。

（4）对于模型来说，可能会导致不适定的情况，即是解出的参数会因为样本的微小变化而出现大的波动。

特征选择，能剔除不相关、冗余、没有差异刻画能力的特征，从而达到减少特征个数、减少训练或者运行时间、提高模型精确度的作用。

特征选择方法

特征选择之所以要引起重视的原因，那就是随着科技发展，很多领域能采集到的特征变量数以万计，而能作为训练集的样本量却往往远小于特征数量（如基因测序、文本分类）。

特征选择主要有两个**功能**：

1.减少特征数量、降维，使模型泛化能力更强，减少过拟合

2.增强对特征和特征值之间的理解

特征选择的**好处**包括：便于理解和可视化数据，降低计算及存储压力，对抗维度灾难以提高模型预测准确率等等。特征选择的三类主流方法为：过滤式、包裹式、嵌入式。

过滤式（Filter）

变量排序就是一种典型的过滤式方法，该方法独立于后续要使用的模型。这种方法的关键就是找到一种能度量特征重要性的方法，比如 pearson 相关系数，信息论理论中的互信息等。《机器学习》（Peter Flach）中还提到了卡方统计量，但未作详细介绍。变量排序方法的主要问题在于忽略了特征之间可能存在的相互依赖关系。一方面，即便排序靠前的特征，如果相关性较强，则引入了冗余的特征；另一方面，排序靠后的特征，虽然独立来看作用不明显，但可能与其它特征组合起来，就有很好的预测作用，如此就损失了有价值的特征。主要介绍如下

1. 方差选择法

使用方差选择法，先要计算各个特征的方差，然后根据阈值，选择方差大于阈值的特征。使用 feature_selection 库的 VarianceThreshold 类来选择特征

2．皮尔逊系数

皮尔森相关系数是一种最简单的运算比较快适用于线性相关的，能帮助理解特征和响应变量之间关系的方法，该方法衡量的是变量之间的线性相关性，结果的取值区间为[-1，1]，-1 表示完全的负相关(这个变量下降，那个就会上升)，+1 表示完全的正相关，0 表示没有线性相关。Pearson 相关系数的一个明显缺陷是，作为特征排序机制，他只对线性关系敏感。如果关系是非线性的，即便两个变量具有一一对应的关系，Pearson 相关性也可能会接近 0。Scikit-learn 提供的 f_regrssion 方法能够批量计算特征的 p-value，非常方便，参考 sklearn 的 pipeline。

3. 互信息和最大信息系

互信息是用来评价一个事件的出现对于另一个事件的出现所贡献的信息量，具体的计算公式为：

$$I(U;C)=\sum_{e_t\in\{1,0\}}\sum_{e_c\in\{1,0\}}P(U=e_t,C=e_c)\log_2\frac{P(U=e_t,C=e_c)}{P(U=e_t)P(C=e_c)},$$

4. 卡方检验

在统计学中，卡方检验用来评价是两个事件是否独立，也就是 P(AB) = P(A)*P(B)

$$X^2(\mathbb{D},t,c)=\sum_{e_t\in\{0,1\}}\sum_{e_c\in\{0,1\}}\frac{(N_{e_te_c}-E_{e_te_c})^2}{E_{e_te_c}}$$

其中 E 代表当两者独立时期望的数量，例如 E11 代表两个事件独立时，共同出现的期望值。具体的计算公式为：

$$E_{11} = N \times P(t) \times P(c) = N \times \frac{N_{11} + N_{10}}{N} \times \frac{N_{11} + N_{01}}{N}$$
$$= N \times \frac{49 + 141}{N} \times \frac{49 + 27652}{N} \approx 6.6$$

最后通过查表的方式确定当前算出的值是不是符合的。

5. 距离相关系数

距离相关系数是为了克服 Pearson 相关系数的弱点而生的。在 x 和 x^2 这个例子中，即便 Pearson 相关系数是 0，我们也不能断定这两个变量是独立的（有可能是非线性相关）；但如果距离相关系数是 0，那么我们就可以说这两个变量是独立的。

6. 频次
7. 信息增益
8. 期望交叉熵

包裹式 (Wrapper)

这类方法的核心思想在于，给定了某种模型，及预测效果评价的方法，然后针对特征空间中的不同子集，计算每个子集的预测效果，效果最好的，即作为最终被挑选出来的特征子集。注意集合的子集是一个指数的量级，故此类方法计算量较大。故而针对如何高效搜索特征空间子集，就产生了不同的算法。其中有一种简单有效的方法叫贪婪搜索策略，包括前向选择与后向删除。在前向选择方法中，初始化一个空的特征集合，逐步向其中添加新的特征，如果该特征能提高预测效果，即得以保留，否则就扔掉。后向删除即是说从所有特征构成的集合开始，逐步删除特征，只要删除后模型预测效果提升，即说明删除动作有效，否则就还是保留原特征。要注意到，包裹式方法要求针对每一个特征子集重新训练模型，因此计算量还是较大的。使用 feature_selection 库的 RFE 类来选择特征的代码如下：

```
1 from sklearn.feature_selection import RFE
2 from sklearn.linear_model import LogisticRegression
3
4 #递归特征消除法，返回特征选择后的数据
5 #参数 estimator 为基模型
6 #参数 n_features_to_select 为选择的特征个数
7 RFE(estimator=LogisticRegression(), n_features_to_select=2).fit_transform(iris.data, iris.targ
```

嵌入式 (Embedded)

嵌入式方法将特征选择融合在模型训练的过程中，比如决策树在分枝的过程中，就是使用的嵌入式特征选择方法，其内在还是根据某个度量指标对特征进行排序。

1. L1 正则化/Lasso

L1 正则化将系数 w 的 l1 范数作为惩罚项加到损失函数上，由于正则项非零，这就迫使那些弱的特征所对应的系数变成 0。因此 L1 正则化往往会使学到的模型很稀疏（系数 w 经常为 0），这个特性使得 L1 正则化成为一种很好的特征选择方法。Scikit-learn 为线性回归提供了 Lasso，为分类提供了 L1 逻辑回归。

2. L2 正则化

L2 正则化将系数向量的 L2 范数添加到了损失函数中。由于 L2 惩罚项中系数是二次方的，这使得 L2 和 L1 有着诸多差异，最明显的一点就是，L2 正则化会让系数的取值变得平均。对于关联特征，这意味着他们能够获得更相近的对应系数。还是以 Y=X1+X2 为例，假设 X1 和 X2 具有很强的关联，如果用 L1 正则化，不论学到的模型是 Y=X1+X2 还是 Y=2X1，惩罚都是一样的，都是 2 alpha。但是对于 L2 来说，第一个模型的惩罚项是 2 alpha，但第二个模型的是 4*alpha。可以看出，系数之和为常数时，各系数相等时惩罚是最小的，所以才有了 L2 会让各个系数趋于相同的特点。

可以看出，L2 正则化对于特征选择来说一种稳定的模型，不像 L1 正则化那样，系数会因为细微的数据变化而波动。所以 L2 正则化和 L1 正则化提供的价值是不同的，L2 正则化对于特征理解来说更加有用：表示能力强的特征对应的系数是非零。

3. 基于树模型的方法

3.1 平均不纯度减少

随机森林由多个决策树构成。决策树中的每一个节点都是关于某个特征的条件，为的是将数据集按照不同的响应变量一分为二。利用不纯度可以确定节点（最优条件），对于分类问题，通常采用 基尼不纯度 或者 信息增益 ，对于回归问题，通常采用的是 方差 或者最小二乘拟合。当训练决策树的时候，可以计算出每个特征减少了多少树的不纯度。对于一个决策树森林来说，可以算出每个特征平均减少了多少不纯度，并把它平均减少的不纯度作为特征选择的值。

3.2 平均精确度减少

另一种常用的特征选择方法就是直接度量每个特征对模型精确率的影响。主要思路是打乱每个特征的特征值顺序，并且度量顺序变动对模型的精确率的影响。很明显，对于不重要的变量来说，打乱顺序对模型的精确率影响不会太大，但是对于

重要的变量来说，打乱顺序就会降低模型的精确率。Sklearn 没有直接提供，但是可以自己去实现的。

其它

除了直接从原始特征中进行选择，我们还可以对特征进行变换、组合，这种思路称为特征构造。其中主要的方法总结如下。

1、聚类

可以使用 kmeas、层次聚类后的聚类中心点来代替相应簇中的一组特征。

2、线性组合

SVD（singular value decomposition） 、PCA 均可视为此类方法，核心思想就是对原有特征进行线性组合，使用组合后的特征作为新的特征输入到训练模型中。这类方法的好处是，考虑到了变量之间可能存在的互补作用，进行组合后，有可能生成了一个更有效的新特征。

参考博客：

1. http://blog.csdn.net/ly_ysys629/article/details/53641569
2. http://blog.csdn.net/a_step_further/article/details/51058784
3. http://www.tuicool.com/articles/aArU7nm

关于实际工程中常用的可以参考网址

https://sanwen8.cn/p/5c8KMe1.html

http://www.17bigdata.com/%E7%BB%93%E5%90%88scikit-learn%E4%BB%8B%E7%BB%8D%E5%87%A0%E7%A7%8D%E5%B8%B8%E7%94%A8%E7%9A%84%E7%89%B9%E5%BE%81%E9%80%89%E6%8B%A9%E6%96%B9%E6%B3%95.html

特征提取

降维

主成分分析算法（PCA）-线性

LDA（Linear Discriminant Analysis）-线性

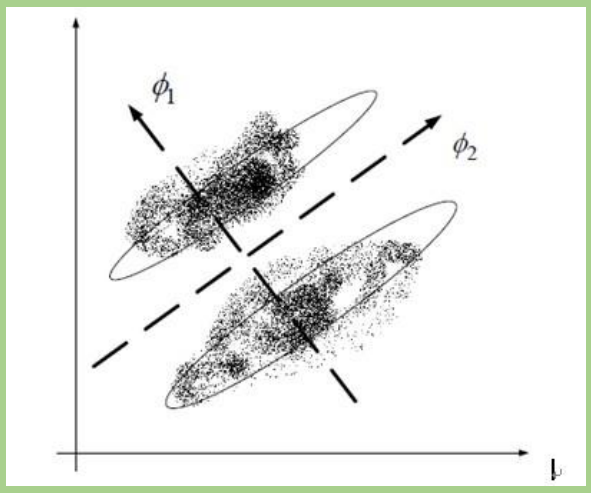
局部线性嵌入（LLE）-非线性-(流行结构-KNN)

Laplacian Eigenmaps 拉普拉斯特征映射-非线性-（局部角度看问题-KNN）

ICA(独立成分分析)

PCA

- ✓ PCA 追求的是在降维之后能够最大化保持数据的内在信息，并通过衡量在投影方向上的数据方差的大小来衡量该方向的重要性。但是这样投影以后对数据的区分作用并不大，反而可能使得数据点揉杂在一起无法区分。这也是 PCA 存在的最大一个问题，这导致使用 PCA 在很多情况下的分类效果并不好。具体可以看下图所示，若使用 PCA 将数据点投影至一维空间上时，PCA 会选择 2 轴，这使得原本很容易区分的两簇点被揉杂在一起变得无法区分；而这时若选择 1 轴将会得到很好的区分结果。

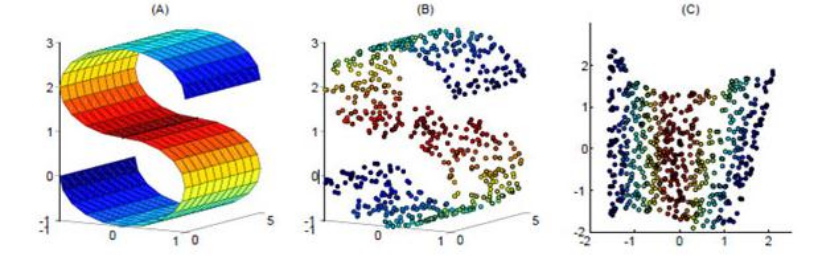


LDA

- ✓ LDA 用的就是轴 1（PCA 结果为轴 2），LDA 最后也是转化成为一个求矩阵特征向量的问题，和 PCA 很像，事实上很多其他的算法也是归结于这一类，一般称之为**谱（spectral）方法**。LDA 思路还是非常清楚的，目标函数就是映射后的中心用来评估类间距。

LLE

- ✓ Locally linear embedding（LLE）是一种非线性降维算法，它能够使降维后的数据较好地保持原有流形结构。LLE 可以说是流形学习方法最经典的工作之一。很多后续的流形学习、降维方法都与 LLE 有密切联系。映射后的数据仍能保持原有的数据流形



LLE 算法认为每一个数据点都可以由其近邻点的线性加权组合构造得到。算法的主要步骤分为三步：(1)寻找每个样本点的 k 个近邻点；(2) 由每个样本点的近邻点计算出该样本点的局部重建权值矩阵；(3) 由该样本点的局部重建权值矩阵和其近邻点计算出该样本点的输出值。具体的算法流程如图 2 所示：

LE

- ✓ LE（Laplacian Eigenmaps）看问题的角度和 LLE 有些相似，也是用局部的角度去构建数据之间的关系。它的直观思想是希望相互间有关系的点（在图中相连的点）在降维后的空间中尽可能的靠近。Laplacian Eigenmaps 可以反映出数据内在的流形结构。步骤为构建图和确定权重。

PCA

定义

PCA（Principal Component Analysis）是一种常用的数据分析方法。PCA 通过**线性**变换将原始数据变换为一组**各维度线性无关**的表示，可用于**提取数据的主要特征分量**，常用于**高维数据的降维。**

推导

设$x(i)$为数据集中的点， u 是要求解的单位向量，那么方差最大化可以形式化为最大化:

$$\frac{1}{m}\sum_{i=1}^{m}(x^{(i)T}u)^2 = \frac{1}{m}\sum_{i=1}^{m}u^T x^{(i)}x^{(i)T}u = u^T\left(\frac{1}{m}\sum_{i=1}^{m}x^{(i)}x^{(i)T}\right)u$$

注意到，对于归一化后的数据，其投影点的均值也为 0，因而在方差计算中直接平方。该公式有一个约束条件，即||u||2 = 1。 这个最大化问题的解就是矩阵 Σ = 1/m ∑m i=1$x(i)x(i)T$的特征向量。这是如何得到的呢？且看下式。

使用拉格朗日方程来求解该最大化问题，则：

$$\ell = u^T\left(\frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T}\right)u - \lambda\left(||u||_2 - 1\right) = u^T\Sigma u - \lambda(u^T u - 1)$$

对 u 求导。

$$\begin{aligned}\nabla_u \ell &= \nabla_u(u^T(\Sigma)u - \lambda(u^T u - 1)) = \nabla_u u^T\Sigma u - \lambda\nabla_u u^T u \\ &= \nabla_u tr(u^T\Sigma u) - \lambda\nabla_u tr(u^T u) = (\nabla_{u^T} tr(u^T\Sigma u))^T - \lambda(\nabla_{u^T} tr(u^T u))^T \\ &= (\Sigma u)^{T^T} - \lambda u^{T^T} = \Sigma u - \lambda u\end{aligned}$$

令导数为 0，可知 u 就是 Σ 的特征向量。那个 12 步的推导相似，在此不赘述了。因为Σ = 1/$m\sum m\ i$=1$x(i)x(i)T$ 是对称矩阵，因而可以得到相互正交的 n 个特征向量*u1,u2,⋯,un+，那么，如何达到降维的效果呢？选取最大的 k 个特征值所对应的特征向量即可。降维后的数据可以用下式来表达：

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

具体更加详细的推导可以参考一篇博客

http://blog.codinglabs.org/articles/pca-tutorial.html

其中主要讲述了从投影、基变换、方差、协方差、对角化等方面更加形象地解释 PCA 的原理。可以得到

降维问题的优化目标：将一组 N 维向量降为 K 维（K 大于 0，小于 N），其目标是选择 K 个单位（模为 1）正交基，使得原始数据变换到这组基上后，各字段两两间协方差为 0，而字段的方差则尽可能大（在正交的约束下，取最大的 K 个方差）。

协方差矩阵：设我们有 m 个 n 维数据记录，将其按列排成 n 乘 m 的矩阵 X，

$$X = \begin{pmatrix} a_1 & a_2 & \mathrm{K} & a_m \\ b_1 & b_2 & \mathrm{K} & b_m \end{pmatrix}$$

设 $C = \frac{1}{m} X * X^T$，则 C 是一个对称矩阵，其对角线分别个各个字段的方差，而第 i 行 j 列和 j 行 i 列元素相同，表示 i 和 j 两个字段的协方差。

协方差矩阵对角化：

优化目标变成了寻找一个矩阵 P，满足 PCP^T 是一个对角矩阵，并且对角元素按从大到小依次排列，那么 P 的前 K 行就是要寻找的基，用 P 的前 K 行组成的矩阵乘以 X 就使得 X 从 N 维降到了 K 维并满足上述优化条件。P 为协方差矩阵的特征向量单位化后按行排列出的矩

阵，其中每一行都是 C 的一个特征向量。如果设 P 按照对角矩阵中特征值的从大到小，将特征向量从上到下排列，则用 P 的前 K 行组成的矩阵乘以原始数据矩阵 X，就得到了我们需要的降维后的数据矩阵 Y。

实例

1. 给定数据
2. 预处理

$$\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}$$

3. 求协方差矩阵

$$C = \frac{1}{5}\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix}\begin{pmatrix} -1 & -2 \\ -1 & 0 \\ 0 & 0 \\ 2 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} \frac{6}{5} & \frac{4}{5} \\ \frac{4}{5} & \frac{6}{5} \end{pmatrix}$$

4. 求特征向量

$$\lambda_1=2,\lambda_2=2/5$$

5. 特征向量

$$c_1\begin{pmatrix} 1 \\ 1 \end{pmatrix}, c_2\begin{pmatrix} -1 \\ 1 \end{pmatrix}$$

6. 标准化

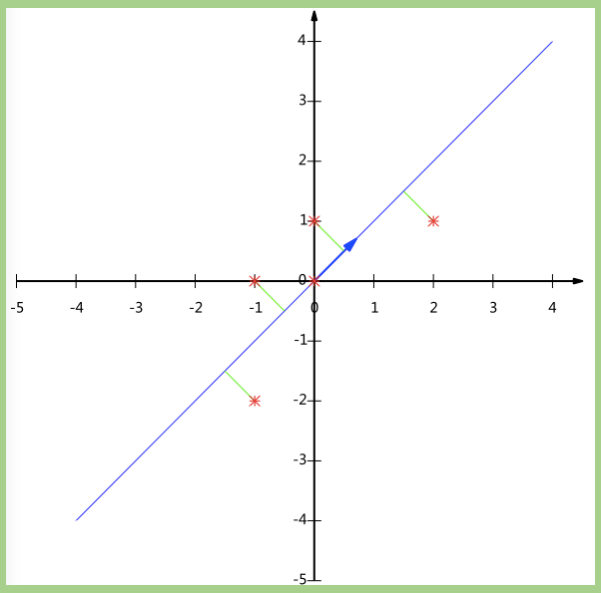
$$P = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

7. 降维度

   后我们用 P 的第一行乘以数据矩阵，就得到了降维后的表示：

$$Y = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}\begin{pmatrix} -1 & -1 & 0 & 2 & 0 \\ -2 & 0 & 0 & 1 & 1 \end{pmatrix} = \begin{pmatrix} -3/\sqrt{2} & -1/\sqrt{2} & 0 & 3/\sqrt{2} & -1/\sqrt{2} \end{pmatrix}$$

8. 表示为



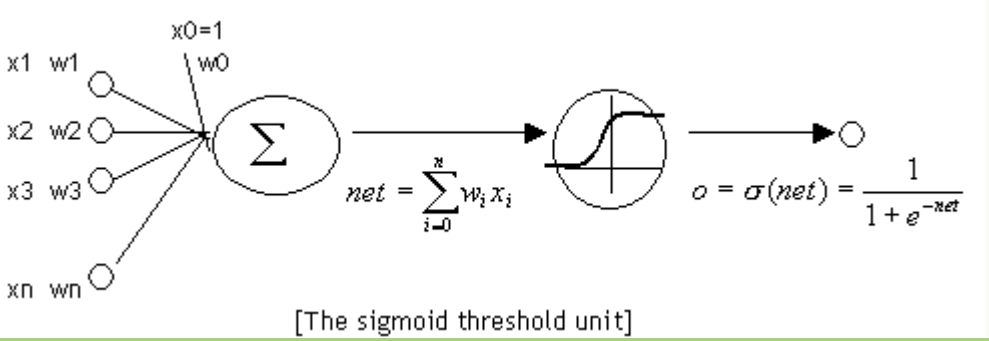
缺点

PCA 本质是将方差最大的方向作为主要特征，并且在各个正交方向上将数据“离相关”，也就是让它们在不同正交方向上没有相关性。因此，PCA 也存在一些限制，例如它可以很好的**解除线性相关**，但是对于高阶相关性就没有办法了，对于存在高阶相关性的数据，可以考虑 Kernel PCA，通过 Kernel 函数将非线性相关转为线性相关，关于这点就不展开讨论了。另外，PCA 假设数据各主特征是分布在正交方向上，如果在非正交方向上存在几个方差较大的方向，PCA 的效果就大打折扣了。

NN(神经网络)

从感知器引入

神经元和感知器本质上是一样的，只不过我们说感知器的时候，它的激活函数是阶跃函数；而当我们说神经元时，激活函数往往选择为 sigmoid 函数或 tanh 函数。如下图所示：

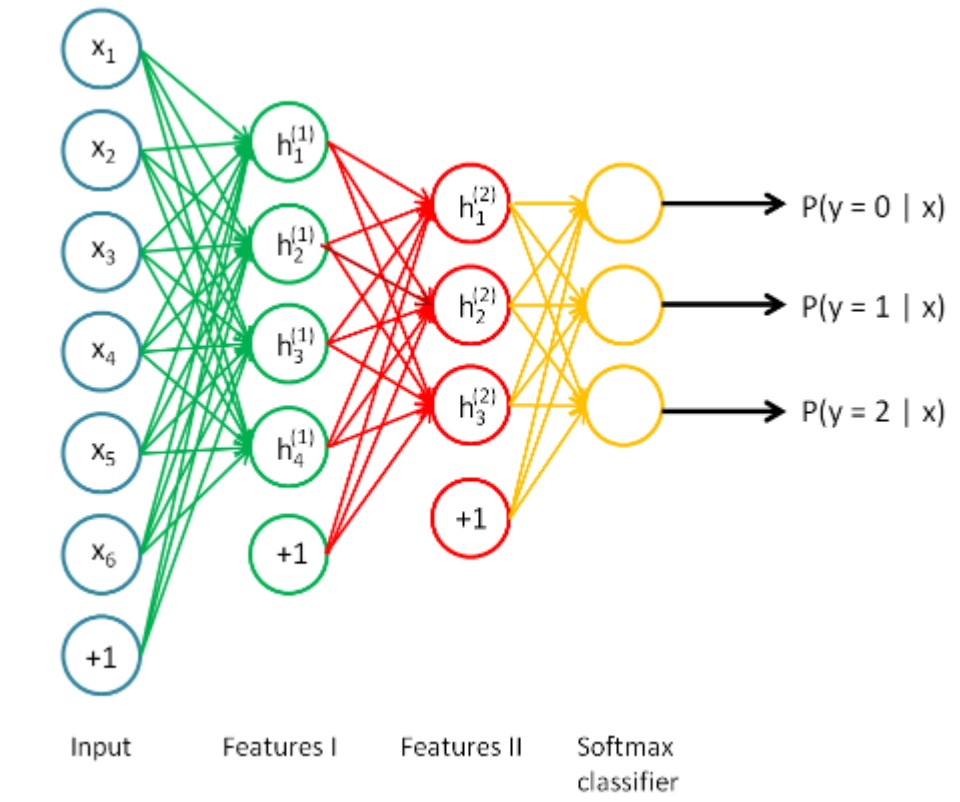


[The sigmoid threshold unit]

sigmoid 函数或 tanh 函数可以保证在数据处理时候的非线性以及归一化，保证结果在 0 和 1 之间，为后面的 CNN 等有下一个基础。

神经网络定义

它是一种模范动物神经网络行为特征，进行分布式并行信息处理的算法数学模型。这种网络依靠系统的复杂程度，通过调整内部大量节点之间相互连接的关系，从而达到处理信息的目的。



- 神经元按照层来布局。最左边的层叫做输入层，负责接收输入数据；最右边的层叫输出层，我们可以从这层获取神经网络输出数据。输入层和输出层之间的层叫做隐藏层，因

为它们对于外部来说是不可见的。

- 同一层的神经元之间没有连接。
- 第 N 层的每个神经元和第 N-1 层的所有神经元相连(这就是 full connected 的含义)，第 N-1 层神经元的输出就是第 N 层神经元的输入。
- 每个连接都有一个权值。

激活函数

来自：http://blog.csdn.net/gzq0723/article/details/51483230

激活函数通常有如下一些性质：

* 非线性： 当激活函数是线性的时候，一个两层的神经网络就可以逼近基本上所有的函数了。但是，如果激活函数是恒等激活函数的时候（即 f(x)=x），就不满足这个性质了，而且如果 MLP 使用的是恒等激活函数，那么其实整个网络跟单层神经网络是等价的。

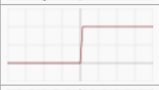
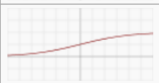
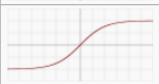
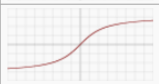
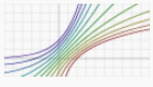
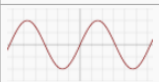
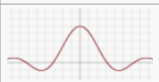
* 可微性： 当优化方法是基于梯度的时候，这个性质是必须的。

* 单调性： 当激活函数是单调的时候，单层网络能够保证是凸函数。

* f(x)≈x： 当激活函数满足这个性质的时候，如果参数的初始化是 random 的很小的值，那么神经网络的训练将会很高效；如果不满足这个性质，那么就需要很用心的去设置初始值。

* 输出值的范围： 当激活函数输出值是 有限 的时候，基于梯度的优化方法会更加稳定，因为特征的表示受有限权值的影响更显著；当激活函数的输出是 无限 的时候，模型的训练会更加高效，不过在这种情况小，一般需要更小的 learning rate.

最近出现新的 serelu 激活函数

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU)[2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU)[3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$
Bent identity		$f(x) = \frac{\sqrt{x^2+1}-1}{2} + x$	$f'(x) = \frac{x}{2\sqrt{x^2+1}} + 1$
SoftExponential		$f(\alpha, x) = \begin{cases} -\frac{\log_e(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x}-1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(\alpha+x)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$
Sinc		$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$

通常我们采用的是 relu 函数:

Relu 函数作为激活函数，有下面几大优势：

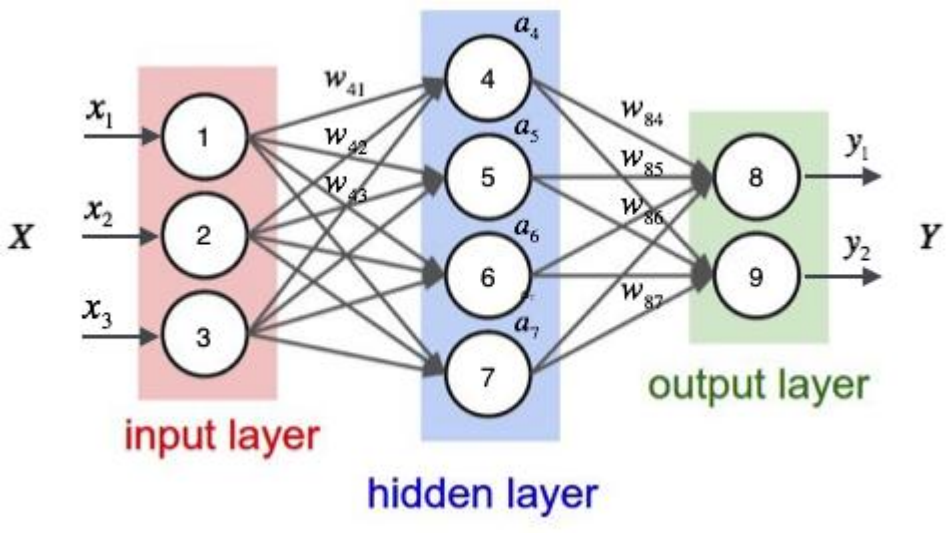
* 速度快 和 sigmoid 函数需要计算指数和倒数相比，relu 函数其实就是一个 max(0,x)，计算代价小很多。

* 减轻梯度消失问题 回忆一下计算梯度的公式。其中，是 sigmoid 函数的导数。在使用反向传播算法进行梯度计算时，每经过一层 sigmoid 神经元，梯度就要乘上一个。从下图可以看出，函数最大值是 1/4。因此，乘一个会导致梯度越来越小，这对于深层网络的训练是个很大的问题。而 relu 函数的导数是 1，不会导致梯度变小。当然，激活函数仅仅是导致梯度减小的一个因素，但无论如何在这方面 relu 的表现强于 sigmoid。使用 relu 激活函数可

以让你训练更深的网络。

* 稀疏性 通过对大脑的研究发现，大脑在工作的时候只有大约 5%的神经元是激活的，而采用 sigmoid 激活函数的人工神经网络，其激活率大约是 50%。有论文声称人工神经网络在 15%-30%的激活率时是比较理想的。因为 relu 函数在输入小于 0 时是完全不激活的，因此可以获得一个更低的激活率。

神经网络的输出



可以得到

$$
\begin{aligned}
a_4 &= sigmoid(w_{41}x_1 + w_{42}x_2 + w_{43}x_3 + w_{4b}) \\
a_5 &= sigmoid(w_{51}x_1 + w_{52}x_2 + w_{53}x_3 + w_{5b}) \\
a_6 &= sigmoid(w_{61}x_1 + w_{62}x_2 + w_{63}x_3 + w_{6b}) \\
a_7 &= sigmoid(w_{71}x_1 + w_{72}x_2 + w_{73}x_3 + w_{7b})
\end{aligned}
$$

又可以得到

$$
\vec{a} = \begin{bmatrix} a_4 \\ a_5 \\ a_6 \\ a_7 \end{bmatrix}, \quad W = \begin{bmatrix} \vec{w}_4 \\ \vec{w}_5 \\ \vec{w}_6 \\ \vec{w}_7 \end{bmatrix} = \begin{bmatrix} w_{41}, w_{42}, w_{43}, w_{4b} \\ w_{51}, w_{52}, w_{53}, w_{5b} \\ w_{61}, w_{62}, w_{63}, w_{6b} \\ w_{71}, w_{72}, w_{73}, w_{7b} \end{bmatrix}, \quad f(\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ . \\ . \\ . \end{bmatrix}) = \begin{bmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ . \\ . \\ . \end{bmatrix}
$$

其中

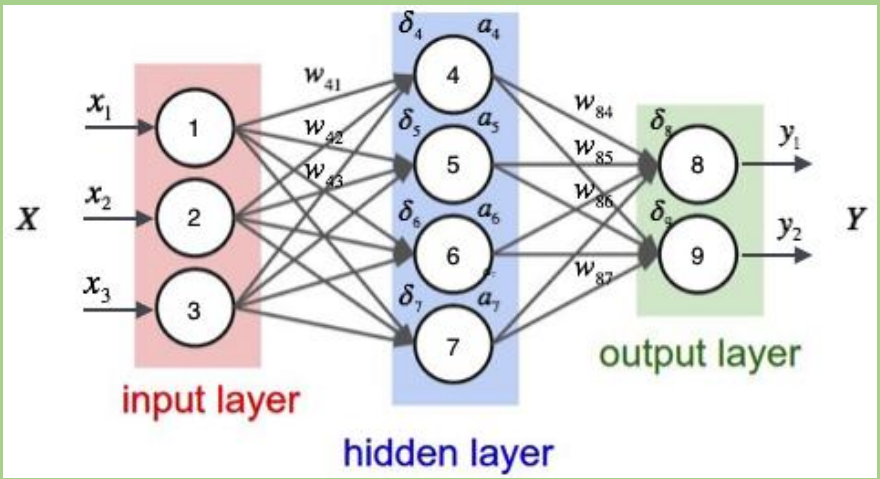
$$
\vec{a} = f(W \cdot \vec{x})
$$

神经网络的训练需要反向转播算法，所以就引入了下面的关于神经网络的训练。

神经网络的训练（反向传播）

神经网络中的反向传播 Back Propagation 指的是使用类似梯度法使得成本函数最小的方法，在神经网络中被称为反向传播

下面举一个例子简单说明一下 BP 算法的基本的过程：



我们假设每个训练样本为$(\vec{x},\vec{t})$，其中向量$\vec{x}$是训练样本的特征，而是$\vec{t}$样本的目标值。

首先，我们根据上一节介绍的算法，用样本的特征$\vec{x}$，计算出神经网络中每个隐藏层节点的输出a_i，以及输出层每个节点的输出y_i。然后，我们按照下面的方法计算出每个节点的误差项δ_i.

- 对于输出层节点 i,

$$\delta_i = y_i(1-y_i)(t_i-y_i)$$

其中，δ_i是节点的误差项，y_i是节点的输出值，t_i是样本对应于节点的目标值。举个例子，根据上图，对于输出层节点 8 来说，它的输出值是y_1，而样本的目标值是t_1，带入上面的公式得到节点 8 的误差项应该是：

$$\delta_8 = y_1(1-y_1)(t_1-y_1)$$

- 对于隐藏节点

$$\delta_i = a_i(1-a_i)\sum_{k\in outputs} w_{ki}\delta_k$$

其中，a_i是节点i的输出值，w_{ki}是节点到它的下一层节点的连接的权重，δ_k是节点的下一

层节点k的误差项。例如，对于隐藏层节点 4 来说，计算方法如下：

$$\delta_4 = a_4(1-a_4)(w_{84}\delta_8 + w_{94}\delta_9)$$

最后，更新每个连接上的权值：

$$w_{ji} \leftarrow w_{ji} + \eta\delta_j x_{ji}$$

其中，w_{ij}是节点i到节点j的权重，η是一个成为学习速率的常数，δ_j是节点j的误差项，x_{ji}是节点i传递给节点j的输入。例如，权重w_{84}的更新方法如下：

$$w_{84} \leftarrow w_{84} + \eta\delta_8 a_4$$

偏置项的输入值永远为 1。例如，节点 4 的偏置项w_{4b}应该按照下面的方法计算：

$$w_{4b} \leftarrow w_{4b} + \eta\delta_4$$

反向传播算法其实就是链式求导法则的应用。然而，这个如此简单且显而易见的方法，却是在 Roseblatt 提出感知器算法将近 30 年之后才被发明和普及的。对此，Bengio 这样回应道：很多看似显而易见的想法只有在事后才变得显而易见

按照机器学习的通用套路，我们先确定神经网络的目标函数，然后用随机梯度下降优化算法去求目标函数最小值时的参数值。我们取网络所有输出层节点的误差平方和作为目标函数。

$$E_d \equiv \frac{1}{2}\sum_{i\in outputs}(t_i - y_i)^2$$

通常采用的是随机梯度下降法对其进行优化计算。

代价函数

二次函数

$$C(w,b) \equiv \frac{1}{2n}\sum_x \|y(x) - a\|^2$$

对权重和偏置的偏导数是：

$$\frac{\partial C}{\partial w} = (a-y)\sigma'(z)x = a\sigma'(z)$$
$$\frac{\partial C}{\partial b} = (a-y)\sigma'(z) = a\sigma'(z)$$

σ 是 sigmoid 激活函数，可以看到当神经元的输出接近 1 的时候，变化平缓，学习率很小。

交叉熵函数

$$C = -\frac{1}{n}\sum_x [y \ln a + (1-y)\ln(1-a)]$$

y 是目标输出，a 是神经元的输出。交叉熵是非负的，在神经元达到很好的学习正确率的时候会接近 0，比二次函数更好的特性就是它可以避免学习速度下降的问题。因为：

$$\frac{\partial C}{\partial w_j} = \frac{1}{n}\sum_x x_j(\sigma(z)-y)$$

更大的误差，更大的学习速度。

SoftMax 和对数似然代价函数

SoftMax：

$$\sum_j a_j^L = \frac{\sum_j e^{z_j^L}}{\sum_k e^{z_k^L}} = 1$$

对数代价似然函数：

$$C \equiv -\ln a_y^L$$

当网络表现良好的时候，$a_y{}^L$ 很接近 1，代价函数就小，反之，则很大。这种组合等效于我们的交叉熵+S 函数的神经网络。

过拟合、规范化

过拟合就是泛化能力很差，在训练集上表现很好，在测试集上表现很差。

规划化：有时称为权重衰减或者 L2 规范化。例如规范化的交叉熵：

$$C = -\frac{1}{n}\sum_{xj}[y_j \ln a_j^L + (1-y_j)\ln(1-a_j^L)] + \frac{\lambda}{2n}\sum_w w^2$$

规范化的效果是让网络倾向于学习小一点的权重，大的权重只有能够给出代价函数第一项足够的提升时才被允许。换言之，规范化可以当作一种寻找小的权重和最小化原始代价函数的折中。这两部分的重要性就由 λ 来控制了，λ 越小，就偏向于最小化原始代价函数，反之，倾向于小的权重。

应用

可以实现非线性分类器，也在系统辨识．模式识别，智能控制等领域有着广泛而吸引人的前景。

非凸性

神经网络存在非凸性，什么是非凸性呢，非凸性：非凸性是指系统的能量函数有多个极值,即系统有多个稳定的平衡态。也就是说比较难找到一个全局最优，而只能陷入局部最优。没有支持向量机（SVM）好。需要做很多工作才可以使用。

优缺点

优点：分类的准确度高,并行分布处理能力强,分布存储及学习能力强，对噪声神经有较强的鲁棒性和容错能力，能充分逼近复杂的非线性关系，具备联想记忆的功能等。
缺点：神经网络需要大量的参数，如网络拓扑结构、权值和阈值的初始值；不能观察之间的学习过程，输出结果难以解释，会影响到结果的可信度和可接受程度；学习时间过长,甚至可能达不到学习的目的。

CNN(卷积神经网络)

引入

20 世纪 60 年代，Hubel 和 Wiesel 在研究猫脑皮层中用于局部敏感和方 向选择的神经元时发现其独特的网络结构可以有效地降低反馈神经网络的复杂性，继而提出了卷积神经网（Convolutional Neural Networks-简称 CNN）。

结构

CNN 的基本结构包括两层，其一为**特征提取层**，每个神经元的输入与前一层的局部接受域相连，并提取该局部的特征。一旦该局部特征被提取 后，它与其它特征间的位置关系也随之确定下来；其二是**特征映射层**，网络的每个计算层由多个特征映射组成，每个特征映射是一个平面，平面上所有神经元的权值 相等。特征映射结构采用影响函数核小的 sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。此外，由于一个映射面上的神经元共享权 值，因而减少了网络自由参数的个数。卷积神经网络中的每一个卷积层都紧跟着一个用来求局部平均与二次提取的计算层，这种特有的两次特征提取结构减小了特征 分辨率。

特点

1.local receptive fields(局部连接)

用于提取局部特征，感受野是指 CNN 结构中某个特征映射到输入空间的区域大小。对于某一特征的感受野，可以通过它的中心位置和它的尺寸大小来描述。

2. shared weights（权值共享）

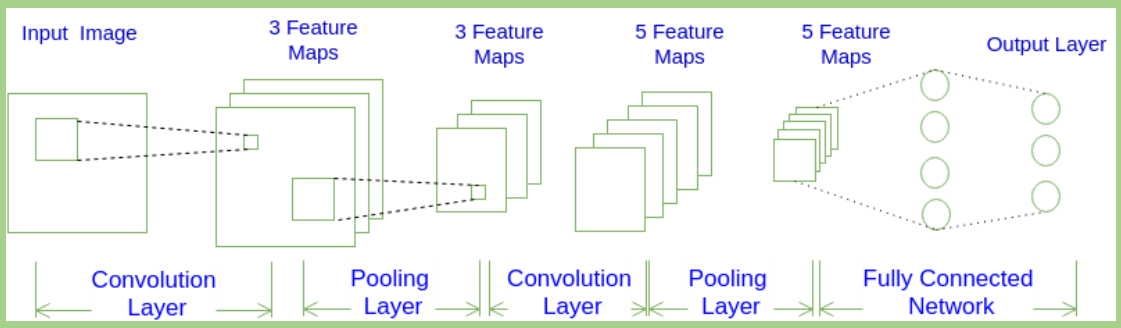
权值共享是指用相同的 filter 去扫一遍图像，相当于提一次特征，得到一个 feature map，为什么用 filter 是因为假设图像具有局部相关性，即一个 pixel 和周围离的近的 pixels 的相关性大，远的 pixels 相关性小

3. pooling（池化）

Pooling 的意义，主要有两点：其中一个显而易见，就是减少参数。通过对 Feature Map 降维，有效减少后续层需要的参数,防止过拟合。

另一个则是 Translation Invariance。它表示对于 Input，当其中像素在邻域发生微小位移时，Pooling Layer 的输出是不变的。这就使网络的鲁棒性增强了，有一定抗扰动的作用.

基本结构



注意：多核卷积可以提取多个特征，上图是提取 3 个特征。

非平衡数据方法

定义

不平衡数据集：在分类等问题中，正负样本，或者各个类别的样本数目不一致。

例子：在人脸检测中，比如训练库有 10 万张人脸图像，其中 9 万没有包含人脸，1 万包含人脸，这个数据集就是典型的不平衡数据集。

直观的影响就是，用这些不平衡的数据训练出来的模型，其预测结果偏向于训练数据中数据比较多的那一类，在人脸检测的例子中，就是检测器的检测结果大部分都偏向于没有检测到人脸图像。

另外一个不平衡数据集，就是信用卡欺诈交易，如果平均的抽取数据，则大部分的数据都是非欺诈交易，只有非常少的部分数据是欺诈交易

影响：不平衡的数据集上做训练和测试，其得到的准确率是虚高的，比如在不平衡数据中，正负样本的比例为 9：1 时，当它的精度为 90%时，我们很有理由怀疑它将所有的类别都判断为数据多的那一类。

不平衡程度相同（即正负样本比例类似）的两个问题，解决的难易程度也可能不同，因为问题难易程度还取决于我们所拥有数据有多大。比如在预测微博互动数的问题中，虽然数据不平衡，但每个档位的数据量都很大——最少的类别也有几万个样本，这样的问题通常比较容易解决；而在癌症诊断的场景中，因为患癌症的人本来就很少，所以数据不但不平衡，样本

数还非常少，这样的问题就非常棘手。综上，可以把问题根据难度从小到大排个序：**大数据+分布均衡<大数据+分布不均衡<小数据+数据均衡<小数据+数据不均衡**。说明:对于小数据集,机器学习的方法是比较棘手的。对于需要解决的问题，拿到数据后，首先统计可用训练数据有多大，然后再观察数据分布情况。经验表明，训练数据中每个类别有 5000 个以上样本，其实也要相对于特征而言，来判断样本数目是不是足够，数据量是足够的，正负样本差一个数量级以内是可以接受的,不太需要考虑数据不平衡问题(完全是经验,没有理论依据,仅供参考)。

解决方法

数据收集

收集更多的数据： 好处：更够揭露数据类别的本质差别，增加样本少的数目以便后面的数据重采样。

采样

采样方法是通过对训练集进行处理使其从不平衡的数据集变成平衡的数据集,在大部分情况下会对最终的结果带来提升。采样分为**上采样**（Oversampling）和**下采样**（Undersampling），上采样是把小众类复制多份，下采样是从大众类中剔除一些样本，或者说只从大众类中选取部分样本。下采样又称为随机采样，上采样又称为过采样。常见的上采样方法有**基于聚类的采样、信息性过采样(合成少数类过采样技术（SMOTE），改进的合成少数类过采样技术（MSMOTE）方法**。

随机采样最大的优点是简单，但缺点也很明显。上采样后的数据集中会反复出现一些样本，训练出来的模型会有一定的过拟合；而下采样的缺点显而易见，那就是最终的训练集丢失了数据，模型只学到了总体模式的一部分。

上采样会把小众样本复制多份，一个点会在高维空间中反复出现，这会导致一个问题，那就是运气好就能分对很多点，否则分错很多点。为了解决这一问题，可以在每次生成新数据点时加入轻微的随机扰动，经验表明这种做法非常有效。

因为下采样会丢失信息，如何减少信息的损失呢？第一种方法叫做 **EasyEnsemble**，利用模型融合的方法（Ensemble）：多次下采样（放回采样，这样产生的训练集才相互独立）

产生多个不同的训练集，进而训练多个不同的分类器，通过组合多个分类器的结果得到最终的结果。第二种方法叫做 **BalanceCascade**，利用增量训练的思想（Boosting）：先通过一次下采样产生训练集，训练一个分类器，对于那些分类正确的大众样本不放回，然后对这个更小的大众样本下采样产生训练集，训练第二个分类器，以此类推，最终组合所有分类器的结果得到最终结果。第三种方法是利用 KNN 试图挑选那些最具代表性的大众样本，叫做 NearMiss，这类方法计算量很大，感兴趣的可以参考“Learning from Imbalanced Data”这篇综述的 3.2.1 节。

性能评价指标

当数据不平衡时，准确度已经失去了它原有的意义， 可以参考的度量标准有：1> 混淆矩阵 CM 2>精度 3>召回率 4>F1 分数（权衡精度和召回率）；5.Kappa 6，ROC 曲线

算法集成

上述部分涉及通过重采样原始数据提供平衡类来处理不平衡数据，在本节中，我们将研究一种替代方法：修改现有的分类算法，使其适用于不平衡数据集。集成方法的主要目的是提高单个分类器的性能。该方法从原始数据中构建几个两级分类器，然后整合它们的预测。主要有基于 bagging 的方法，基于 boosting 的方法等。可参考 http://www.dlworld.cn/YeJieDongTai/3611.html

惩罚模型

意思就是添加新的惩罚项到 cost 函数中，以使得小样本的类别被判断错误的 cost 更大，迫使模型重视小样本的数据。比如：带惩罚项的 SVM

LSI 隐含语义索引

定义

潜在语义索引(LSI)，又称为潜在语义分析(LSA)，是在信息检索领域提出来的一个概念。主要是在解决两类问题，一类是一词多义，如“bank”一词，可以指银行，也可以指河岸；另一类是一义多词，即同义词问题，如“car”和“automobile”具有相同的含义，如果在检索的过程中，在计算这两类问题的相似性时，依靠余弦相似性的方法将不能很好的处理这样的问题。所以提出了潜在语义索引的方法，利用 SVD 降维的方法将词项和文本映射到一个新的空间。LSA 的基本假设是，如果两个词多次出现在同一文档中，则这两个词在语义上具有相似性。LSA 使用大量的文本上构建一个矩阵，这个矩阵的一行代表一个词，一列代表一个文档，矩阵元素代表该词在该文档中出现的次数，然后再此矩阵上使用奇异值分解（SVD）来保留列信息的情况下减少矩阵行数，之后每两个词语的相似性则可以通过其行向量的 cos 值（或者归一化之后使用向量点乘）来进行标示，此值越接近于 1 则说明两个词语越相似，越接近于 0 则说明越不相似。

过程

词-文档矩阵（Occurences Matrix)

LSA 使用词-文档矩阵来描述一个词语是否在一篇文档中。词-文档矩阵式一个稀疏矩阵，其行代表词语，其列代表文档。一般情况下，词-文档矩阵的元素是该词在文档中的出现次数，也可以是是该词语的 TF-IDF (term frequency–inverse document frequency)。
词-文档矩阵和传统的语义模型相比并没有实质上的区别，只是因为传统的语义模型并不是使用“矩阵”这种数学语言来进行描述。

降维

在构建好词-文档矩阵之后，LSA 将对该矩阵进行降维，来找到词-文档矩阵的一个低阶近似。降维的原因有以下几点：

- 原始的词-文档矩阵太大导致计算机无法处理，从此角度来看，降维后的新矩阵式原有矩阵的一个近似。

- 原始的词-文档矩阵中有噪音，从此角度来看，降维后的新矩阵式原矩阵的一个去噪矩阵。
- 原始的词-文档矩阵过于稀疏。原始的词-文档矩阵精确的反映了每个词是否“出现”于某篇文档的情况，然而我们往往对某篇文档“相关”的所有词更感兴趣，因此我们需要发掘一个词的各种同义词的情况。

降维的结果是不同的词或因为其语义的相关性导致合并，如：

{(car), (truck), (flower)} --> {(1.3452 * car + 0.2828 * truck), (flower)}

降维可以解决一部分同义词的问题，也能解决一部分二义性问题。具体来说，原始词-文档矩阵经过降维处理后，原有词向量对应的二义部分会加到和其语义相似的词上，而剩余部分则减少对应的二义分量。

推导

假设 X 是词-文档矩阵，其元素（i,j）代表词语 i 在文档 j 中的出现次数，则 X 矩阵看上去是如下的样子：

$$\begin{matrix} & & \mathbf{d}_j \\ & & \downarrow \\ \mathbf{t}_i^T \rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} \end{matrix}$$

可以看到，每一行代表一个词的向量，该向量描述了该词和所有文档的关系。

$$\mathbf{t}_i^T = \begin{bmatrix} x_{i,1} & \dots & x_{i,n} \end{bmatrix}$$

相似的，一列代表一个文档向量，该向量描述了该文档与所有词的关系。

$$\mathbf{d}_j = \begin{bmatrix} x_{1,j} \\ \vdots \\ x_{m,j} \end{bmatrix}$$

词向量$\mathbf{t}_i^T \mathbf{t}_p$的点乘可以表示这两个单词在文档集合中的相似性。矩阵 XX^T 包含所有词向量点乘的结果，元素（i,p）和元素（p,i）具有相同的值，代表词 p 和词 i 的相似度。类似的，矩阵 X^T X 包含所有文档向量点乘的结果，也就包含了所有文档那个的相似度。

现在假设存在矩阵 X 的一个分解，即矩阵 X 可分解成正交矩阵 U 和 V，和对角矩阵 $\sum$ 的乘

积。这种分解叫做奇异值分解（SVD），即：

$$X = U\Sigma V^T$$

因此，词与文本的相关性矩阵可以表示为：

$$\begin{aligned} XX^T &= (U\Sigma V^T)(U\Sigma V^T)^T = (U\Sigma V^T)(V^{T^T}\Sigma^T U^T) = U\Sigma V^T V\Sigma^T U^T = U\Sigma\Sigma^T U^T \\ X^T X &= (U\Sigma V^T)^T(U\Sigma V^T) = (V^{T^T}\Sigma^T U^T)(U\Sigma V^T) = V\Sigma^T U^T U\Sigma V^T = V\Sigma^T\Sigma V^T \end{aligned}$$

因为$\Sigma\Sigma^T$与$\Sigma^T\Sigma$是对角矩阵，因此U肯定是由XX^T的特征向量组成的矩阵，同理V是X^TX特征向量组成的矩阵。这些特征向量对应的特征值即为$\Sigma\Sigma^T$中的元素。综上所述，这个分解看起来是如下的样子：

$$\begin{matrix} & X & & U & & \Sigma & & V^T \\ & (\mathbf{d}_j) & & & & & & (\hat{\mathbf{d}}_j) \\ & \downarrow & & & & & & \downarrow \\ (\mathbf{t}_i^T)\rightarrow & \begin{bmatrix} x_{1,1} & \dots & x_{1,n} \\ \vdots & \ddots & \vdots \\ x_{m,1} & \dots & x_{m,n} \end{bmatrix} & = (\hat{\mathbf{t}}_i^T)\rightarrow & \begin{bmatrix} \begin{bmatrix} \\ \mathbf{u}_1 \\ \\ \end{bmatrix} & \dots & \begin{bmatrix} \\ \mathbf{u}_l \\ \\ \end{bmatrix} \end{bmatrix} & \cdot \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_l \end{bmatrix} & \cdot \begin{bmatrix} [& &] \\ & \mathbf{v}_1 & \\ [& &] \\ & \vdots & \\ [& &] \\ & \mathbf{v}_l & \\ [& &] \end{bmatrix} \end{matrix}$$

$\sigma_1, \dots, \sigma_l$ 被称作是奇异值，而 $u_1, \dots, u_l$ 和$v_1, \dots, v_l$则叫做左奇异向量和右奇异向量。通过矩阵分解可以看出，原始矩阵中的$\mathbf{t}_i$ 只与 U 矩阵的第 i 行有关，我们则称第 i 行为 $\hat{\mathbf{t}}_i$。同理，原始矩阵中的$\hat{\mathbf{d}}_j$只与V^T中的第 j 列有关，我们称这一列为$\hat{\mathbf{d}}_j$。$\mathbf{t}_i$与$\hat{\mathbf{d}}_j$并非特征值，但是其由矩阵所有的特征值所决定。

当我们选择 k 个最大的奇异值，和它们对应的 U 与 V 中的向量相乘，则能得到一个 X 矩阵的 k 阶近似，此时该矩阵和 X 矩阵相比有着最小误差（即残差矩阵的 Frobenius 范数）。但更有意义的是这么做可以将词向量和文档向量映射到语义空间。向量$\hat{\mathbf{t}}_i$与含有 k 个奇异值的矩阵相乘，实质是从高维空间到低维空间的一个变换，可以理解为是一个高维空间到低维空间的近似。同理，向量 $\hat{\mathbf{d}}_j$也存在这样一个从高维空间到低维空间的变化。这种变换用公式总结出来就是这个样子：

$$X_k = U_k\Sigma_k V_k^T$$

有了这个变换，则可以做以下事情：

- 判断文档 j 与 q 在低维空间的相似度。比较向量 $\Sigma_k\hat{\mathbf{d}}_j$ 与向量$\Sigma_k\hat{\mathbf{d}}_q$ (比如使用余弦夹角)即可得出。
- 通过比较$\Sigma_k\hat{\mathbf{t}}_i^T$ 与 $\Sigma_k\hat{\mathbf{t}}_p^T$ 可以判断词i和词p的相似度。
- 有了相似度则可以对文本和文档进行聚类。

- 给定一个查询字符串，算其在语义空间内和已有文档的相似性。
- 要比较查询字符串与已有文档的相似性，需要把文档和查询字符串都映射到语义空间，对于原始文档，由以下公式可以进行映射：

$$\hat{\mathbf{d}}_j = \Sigma_k^{-1} U_k^T \mathbf{d}_j$$

其中对角矩阵Σ_k 的逆矩阵可以通过求其中非零元素的倒数来简单的得到。

同理，对于查询字符串，得到其对应词的向量后，根据公式 $\hat{\mathbf{q}} = \Sigma_k^{-1} U_k^T \mathbf{q}$将其映射到语义空间，再与文档进行比较。

应用

低维的语义空间可以用于以下几个方面:

- 在低维语义空间可对文档进行比较，进而可用于文档聚类和文档分类。
- 在翻译好的文档上进行训练，可以发现不同语言的相似文档，可用于跨语言检索。
- 发现词与词之间的关系，可用于同义词、歧义词检测。.
- 通过查询映射到语义空间，可进行信息检索。
- 从语义的角度发现词语的相关性，可用于“选择题回答模型”（multi choice qustions answering model）。

（http://blog.csdn.net/roger__wong/article/details/41175967）

问答

1. 解释欠拟合，过拟合？

答案

1. 考虑模型的简单和复杂会导致欠拟合和过拟合。过拟合的解决方法有：

   1）early stopping

   2）数据集扩增

   3）正则化

   4）dropout

   5）验证数据：交叉验证

   6）减少特征

   7）权值衰减。主要用于神经网络中