Tom Mitchell provides a more modern definition: "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

In supervised learning, we are given a data set and already know what our correct output should look like, having the idea that there is a relationship between the input and the output.

Supervised learning problems are categorized into "regression" and "classification" problems. In a regression problem, we are trying to predict results within a continuous output, meaning that we are trying to map input variables to some continuous function. In a classification problem, we are instead trying to predict results in a discrete output. In other words, we are trying to map input variables into discrete categories.

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function h : X → Y so that h(x) is a “good” predictor for the corresponding value of y. For historical reasons, this function h is called a hypothesis.

We will also use X to denote the space of input values, and Y to denote the space of output values. In this example, X = Y = ℝ.

We can measure the accuracy of our hypothesis function by using a **cost function**. This takes an average difference (actually a fancier version of an average) of all the results of the hypothesis with inputs from x's and the actual output y's



This function is otherwise called the "Squared error function", or "Mean squared error".

So we have our hypothesis function and we have a way of measuring how well it fits into the data. Now we need to estimate the parameters in the hypothesis function. That's where gradient descent comes in.

The way we do this is by taking the derivative (the tangential line to a function) of our cost function. The slope of the tangent is the derivative at that point and it will give us a direction to move towards. We make steps down the cost function in the direction with the steepest descent. The size of each step is determined by the parameter α, which is called the learning rate.

The gradient descent algorithm is:

repeat until convergence:



At each iteration j, one should simultaneously update the parameters *θ*1​,*θ*2​,...,*θn*​. Updating a specific parameter prior to calculating another one on the  *j*(*th*) iteration would yield to a wrong implementation.

线性回归

1. 模型定义

线性回归模型公式：

通过将X替换为输入X的非线性函数，线性回归模型可以模拟非线性关系：

这个成为线性模型的基本表达式。由于对于W来说仍为线性模型，所以仍然叫做线性回归。一个简单的例子为多项式函数，基函数也可以为高斯基函数、sigmoid基函数、tanh基函数、傅里叶基函数。

我们也可以使用多个输入变量，比如E[y|X] = w0 + w1x1 + w2x2

1. 损失函数
2. 过拟合和正则项

逻辑回归

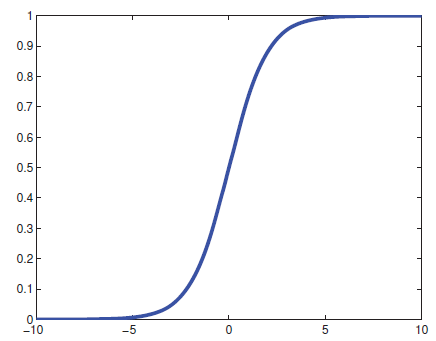
建立概率分类模型的一个方法是创建一个条件概率的联合模型p(y|x)，这个叫做生成方法。另一种方式是直接拟合一个形式为p(y|x)的模型，这种方法叫做判别方法，逻辑回归就是这种方法。实际上，我们假设判别模型的参数是线性的，这件极大地简化模型拟合。

1. 模型定义

逻辑回归是一个二元分类模型：

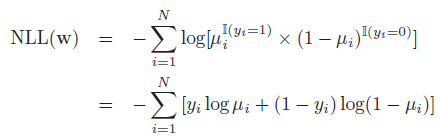


一维逻辑回归如下图所示。



1. 模型拟合
2. MLE

逻辑回归似然性的负对数如下：



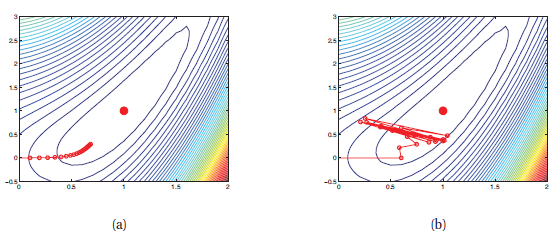
这个也叫作交叉熵误差。

1. 梯度方向

无约束最优化的最简单算法是梯度下降，也叫做最陡下降。梯度下降公式为：



η叫做步长或者学习速率。最主要的问题是如果确定步长η大小。η太小，收敛会很慢，η太大，则会导致振荡不收敛。



我们介绍一种更可靠的方法来选择步长大小，这个方法保证收敛到局部最小值不管起点再什么地方。根据taylor公式



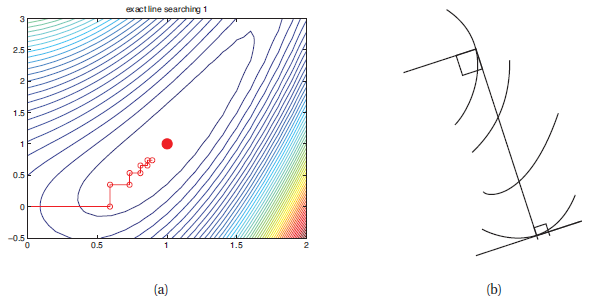
D为梯度方向。如果η足够小，由于梯度方向为f函数值减小方向，所以

如果我们选择的η太小，那么我们将收敛非常慢，于是我们设定η为使得



取得最小值。这叫做线性最小化或线性搜索。

线性搜索的过程为：



可以看到线性搜索的图像为锯齿形，一个简单的启发式的减轻锯齿形的影响的方法是增加一个momentum项（θk – θk-1），如：

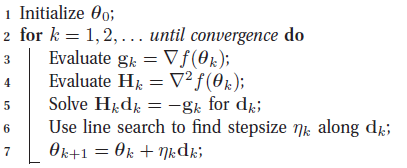


控制着动量项的比重。

另外一种减小锯齿形的方法是conjugate gradients。

1. Newton方法

求严格凸函数最小值的牛顿方法



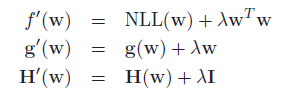
考虑空间的曲率（比如Hessian矩阵），能够得到更快的最优化方法，这个叫做二阶优化方法。牛顿方法就是这样的一个例子。通过迭代计算达到最优值：



1. L2正则

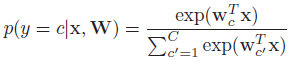
实际上即使我们获得了足够多的数据，正则项对于分类问题来说也是很重要的。假设样本数据线性可分，那么最大似然估计将在||w|| -> ∞时得到，此时sigmoid函数在0点处将有一个无限大的斜率。这将给训练样本赋予最大的概率可能，模型就失去了普遍性。

为了防止模型过拟合，我们可以使用L2正则，新的目标函数，梯度和Hessian矩阵为：



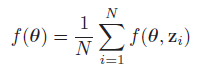
1. 多分类逻辑回归

多分类逻辑回归，有时候又叫做最大熵分类器。这个模型为：



1. 在线学习和随机优化

传统机器学习是离线的，即利用一批数据，来优化目标函数：

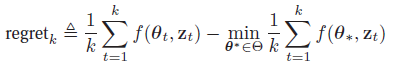


再频率决策理论中，平均损失被称为风险，所以这一类方法又叫做经验风险最小化，或ERM。

但是，如果我们有数据流，我们需要在线学习，所以我们可以对每一条新得到的数据更新我们的估计，而不是等到拿到所有数据。即使我们有一批数据，我们也可以将他们看作数据流来避免数据量过大造成内存不足。

1. 在线学习和regret 最小化

在理论机器学习领域，在线学习使用的目标函数是regret，regret是和事后使用一个固定的参数值获得的最好结果相比，我们的损失：



上式第二项为事后使用一个固定参数值获得的最优结果，regret为在线学习平均损失和最优结果之间的差异。

在线学习的一个简单算法是在线梯度下降，即在每一步k，参数更新公式如下：



是向量V在空间ν的投影，gk为梯度，ηk是步长。下面我们将看到最小化regret和传统目标函数比如MLE的关系。

1. 随机优化和最小化风险

现在假设我们想最小化预期损失，而不是最小化过去的regret，即最小化



目标损失函数中一些变量是随机的优化方法称为随机优化。

假设我们有一个来自同一分布的无限样本流。一种优化随机目标函数的方法是在每一步都使用上式。这叫做随机梯度下降。因为我们经常想做一个单参数的估计，我们可以使用一个平均的方法：

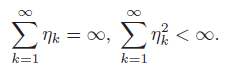


这叫做Polyak-Rupper平均，可以通过递归计算实现：



1. 步长大小

我们现在讨论一些关于确保SGD收敛的学习速率需要满足的充分条件：



学习速率η的集合称为学习速率表，各种公式用来计算η，比如ηk = 1/k；或者

，

会减缓算法早期的迭代，控制旧值的遗忘速度。

1. 每个参数的步长

SGD的一个缺点是对所有参数使用相同的步长。我们简单介绍一个叫做adagrad的算法，它本质上和对角Hessian矩阵类似。特别的，如果是第i个参数在第k次的值，是这个值的梯度，然后我们的更新公式如下：



1. SGD和批训练对比

如果我们没有无限输入数据流，我们可以使用训练样本模拟一个。

1. 在离线场景下，最好采用小批量数据B计算梯度，B=1时，为标准SGD，B=N时，为标准最陡下降，一般B选100左右。

虽然SGD是简单的一次方法，但是它在一些问题上表现非常好，特别是对大的数据集。原因是可以从少部分样本得到比较好的梯度估计。在大量数据上精心计算精确的梯度是很浪费时间的，因为算法将在下一步重新重新计算梯度。对计算时间比较好的利用是估计噪音并且快速遍历参数空间。在极端情况下，如果我们把训练样本复制一份，数量变成2倍，那么批量方法将使用2倍的时间，而在线方法不受影响，因为梯度的方向没有变化。

除了提高速度，SGD不太容易收敛到局部最优，因为加入了一些噪声。因此它在机器学习界非常流行，用于拟合具有非凸目标的模型，比如神经网络和深度置信网络。

1. LMS算法

我们以SGD为例，考虑如何计算在线线性回归的MLE。我们已经推导出批量梯度的公式为：



在线梯度第k次迭代的梯度为：

可以看到第k次迭代的梯度为向量Xk乘以我们的预测值和真实值之间的误差，所以这个梯度像一个错误信号。

计算出梯度后，参数更新如下：



这不需要一个投影，因为这个问题是无约束问题。这个算法叫做least mean squares或者LMS算法，也叫delta rule，或者windrow-hoff rule。

1. Perceptron 算法

现在让我们考虑如何在线拟合一个二分类逻辑回归。在线拟合参数更新公式为：





我们看到这和LMS算法具有一样的形式。实际上所有广义线性模型都是这样的。

我们现在考虑一下这个算法得近似。特别的，让



表示样本xi最有可能是的分类。我们再梯度表达式中使用sigmod函数来代替y为1的概率，即，那么梯度近似为：



如果我们假设y为1或-1，来代替0或1，那表达式将更简单，y的预测值为



如果，那我们预测错误，如果，我们预测正确。

再每一步，我们通过增加梯度来更新权重向量。如果我们正确地预测了y，那么梯度为0，我们不需要改变θ，如果我们预测错误，那么负的梯度为，结果为

2Xi或者-2Xi。我们把2提取到步长η中，那么当我们预测错误时，θ的更新为：



这种方法被称为感知机算法，当样本线性可分时，算法将收敛。然而如果数据线性不可分，算法将不会收敛，或者即使收敛，也将使用很长时间。

1. 生成式和判别式分类器

当我们拟合一个判别式模型，我们通常最大化条件log似然

当我们拟合一个生成式模型，我们通常最搭话联合log似然

当GDA做出的高斯分布假设是正确的，那么GDA与逻辑回归相比，达到同样的性能水平需要更少的样本，但是如果高斯分布假设是不正确的，逻辑回归将表现更好。

这是因为判别式模型不需要对特征的分布进行建模。分类条件密度相当复杂，特别的

是一个多重分布，或者很难估计，然而分类后验概率，是一个简单的sigmoid函数，阈值为0.55.所以通常判别式方法更准确，因为他们的任务简单一些。然而精度不是选择方法唯一的重要因素，下面将讨论两种方法的优劣：

1. 是否容易拟合？

生成式分类器很容易训练，比如我们可以通过简单的计数和平均来拟合一个朴素贝叶斯模型和一个LDA模型，但是逻辑回归需要解决一个凸优化问题，训练更慢。

1. 分别拟合每个分类?

在生成式分类器，我们单独估计每一个分类的条件概率密度，所以当我们增加更多分类时不需要重新训练模型。相比较而言，生成式模型，所有参数都是相关的，所以当我们增加一个分类时需要重新训练整个模型。

1. 训练样本特征缺失？

有些时候输入样本的某些特征没有观察到。生成式分类器处理这类情况比较简单，但是对于判别式分类器，这类问题没有标准答案，因为模型假定输入x是可以任意调整的。

1. 输入和输出对称？

我们可以逆向运行一个生成式模型，从输出推断出输入。判别式模型不可能这么做。

1. 处理无标签数据？

在半监督学习中，使用没有标签的数据来帮助解决一个监督学习任务。生成式模型很容易完成这样的任务，判别式相对很难。

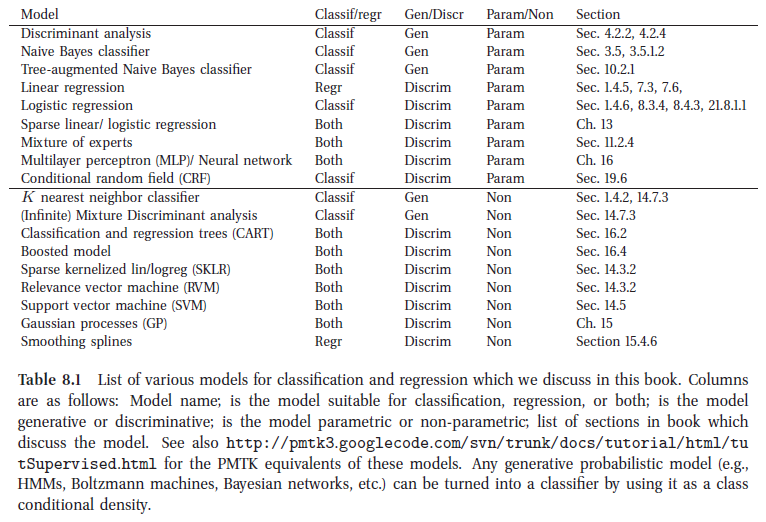
1. 能否对数据进行预处理？

判别式方法的一个很好的优点是我们可以对输入数据做任何的预处理，比如我们可以使用代替x。在预处理数据上很难推断一个生成式模型，因为新的特征往往存在复杂的相关性。

1. 概率校准？

一些生成式模型比如朴素贝叶斯，做了一个强独立假设，这个假设实际上不常见。这将导致一个非常极端的后验分类概率(非常接近0或1)。判别式模型，比如逻辑回归，通常对得到的结果做了更好的校准

有很多关于这两种方法优劣的争论。但是这两种方法都应该在您的工具箱中。



1. 处理缺失数据

有时候输入X存在数据缺失，生成式模型能够处理这样的数据。

我们可以增加一个二元变量ri∈{0，1}，来标明数据x是否缺失。联合概率分布为



这里是控制数据是否缺失得参数。如果假设，我们认为数据是完全随机丢失，如果假设，是xi得部分观察值，我们认为数据是随机丢失。如果这一项没有定义，那么我们认为数据不是随机丢失。

当数据存在丢失时，需要区分丢失数据是测试数据还是训练数据。我们将讨论这两种情况。

测试数据丢失：

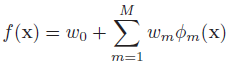
训练数据丢失：

自适应基函数模型

在14、15章我们讨论了核函数。但是选择好的核函数是非常困难的，比如我们如何定义两张图片之间的相似度，像素级别的强度比较（高斯核）效果不好。虽然可以为特定任务定制核，但是如果我们能够学习到这个核将更有趣。

一种方法是学习核函数的参数，这种方法依赖于一个好的基函数。

另一种方法是从数据直接学习有用的特征函数。我们称这种方法为自适应基函数模型ABM，模型公式为：



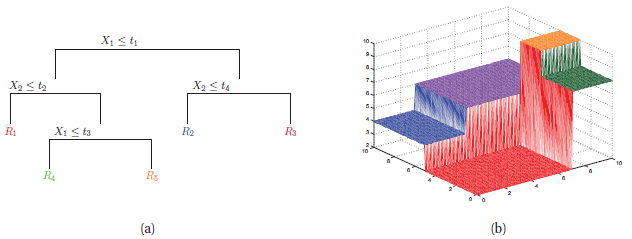
是第m个基函数。我们可以定义Vm为基函数的参数，于是目标函数中的θ可以写成。最终对于参数来说，模型不再是线性的，所以我们将只能计算θ的局部最优MLE或者MAP。这样的模型通常明显优于线性模型。

1. 分类和回归树

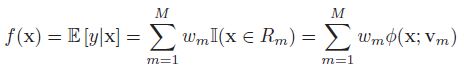
分类和回归树或者CART模型，也叫做决策树(不要与决策理论中的决策树混淆)。决策树通过递归地划分输入空间，并在局部输入空间定义一个本地模型。这个可以用一个带叶子的树来表示。

1. 基本概念

解释CART树方法，考虑下图。

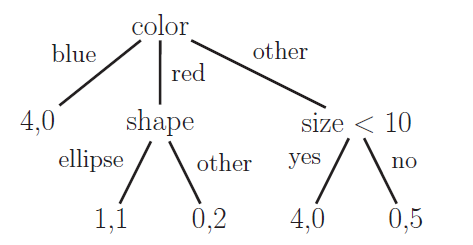


我们可以用以下方式来表达这个模型：



Rm为第m个区域，wm为m区域的平均响应值，vm指定了参数的划分以及阈值。一个CART模型就是一个自适应基函数，这个基函数定义了这些区域，权重定义了每个区域的响应。下面我们讨论如何找到这些基函数。

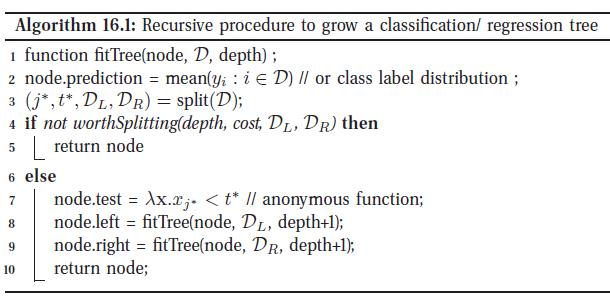
我们也可以把这方法用于分类，通过将叶子节点上的平均响应值替换成分类标签的分布，如下图所示。



比如我们首先检查对象的颜色，如果是蓝色的，那么我们选择左边的分支，该分支下有4个正样本和0个负样本。于是当x为blue时，我们预测，如果是红色，我们在检查形状，如果是椭圆形，我们达到节点(1,1)，所以我们预测这个概率只是满足从根节点到叶子节点路径上所有特征值的正样本的经验分布。

1. 树成长

寻找样本的局部最优是一个NP完全问题，所以通常使用贪婪算法来计算局部最优MLE。CART，C4.5和ID3使用这样的方法。



分支选择最好的特征，并且特征最好的阈值：



确定一个节点是否值得继续分支可以使用几种启发式剪枝规则：

* 代价函数是否减少的太小？通常我们定义使用一个特征的增益来作为代价函数减少的标准化测量：



* 是否达到树的最大深度
* 节点左右分支的样本是否足够同种类的，比如左右分支的标签都是一样的
* 左右分支的样本数是不是太少

剩下的就是指定用于评估提议的拆分质量的代价函数。这和我们的目标是回归还是分类有关，我们将讨论这两种情形。

回归代价函数

我们定义回归代价函数为：



是指定数据集响应值的平均。或者我们可以对每一个叶子节点拟合一个线性回归模型，使用从根节点到叶子节点路径上的特征为输入，然后计算残差。

分类代价函数

有几种方法来测量分类的分支质量。

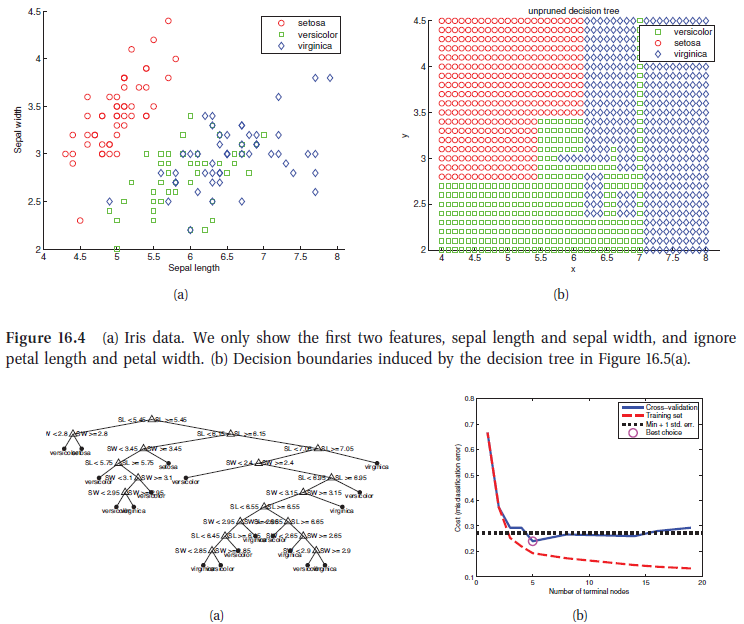
错分率

熵或者偏差

基尼指数

例子

参考下图，



图中的树交叉验证比训练集上的误差大很多，说明过拟合。下面我们讨论树的剪枝。

1. 剪枝

为防止过拟合，当增加一个子树，误差没有明显减少时，我们可以停止树的生长。然而，这种方式是局限的。比如，在XOR数据中，由于每一个特征都几乎没有预测能力，所以树不会增长。

标准方法是先生长一颗完成的树，然后再进行剪枝。剪枝的标准是剪掉那些误差减小最少的分支。

我们可以评估每一个子树的交叉验证误差，然后选择cv误差在1个标准差之内的树。

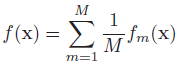
1. 树模型的利弊

CART模型的优点：易解释，能够方便地处理离散和连续混合输入，对输入的单调变换不敏感，能够自动选择特征，对离群点相对稳健，能够处理大规模数据集，能够处理缺失输入

缺点：最主要的缺点是对比其他模型，他们预测的准确度不够高。一个原因是树构建算法的贪婪性质，另一个原因是树模型不够稳健，树顶端的误差会影响下级树。用频率用语来说，CART树方差较大。

1. 随机森林

减小方差的一种方法是对多个树的估计进行平均。比如，我们可以在数据不同自己上训练M个不同的树，放回抽样，并合并计算：



Fm是第m个树。这个技术叫做bagging，表示bootstrap aggregating。

不幸的是，在数据不同子集上运行相同的算法会产生高度相关的预测，这影响了偏差减少的可能。随机森林技术试图通过训练基于随机选择的输入变量子集和随机选择的数据子集的树来解除这些树之间的相关性。这些模型有很高的预测准确度，并在很多应用中使用，比如微软kinect的身体姿态识别。

1. CART和HME对比
2. 广义可加模型

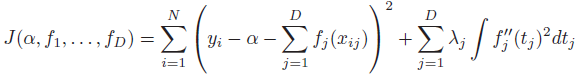
创建一个多输入非线性模型的简单方法是使用广义可加模型，形式为：



如果我们使用回归样条来拟合fi，那么每一个fi都可以写成，整个模型可以写成：



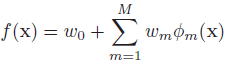
然而，使用平滑样条来拟合fi更常见，在这种情况下，目标函数为：



Λj表示fj正则强度。

1. 回火
2. 计算复杂度
3. 多元自适应回归样条
4. Boosting

Boosting是一种拟合自适应基函数模型的贪婪算法，如16.3式



式中代表弱学习器或者基学习器。这个算法通过将数据以不同权重依次应用于弱学习器序列来实现，被前面的弱学习器错误分类的数据被赋予更多权重。

这个弱学习器可以式分类或者回归算法，通常用于CART模型。2009年，一个针对10个不同分类器的广泛的经验比较，boosting decision trees是在错误率和概率上表现最好的分类器。

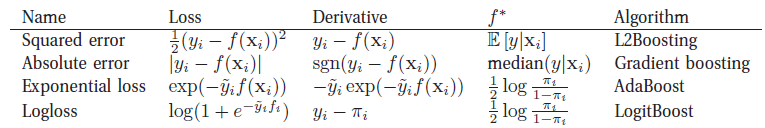
鉴于boosting方法的成功，统计学家开始对这种方法产生兴趣。Breiman指出boosting可以被解释成函数空间内的梯度下降形式。这一部分，我们将介绍boosting的统计学解释。

1. 正向阶段性叠加建模

Boosting的目的是解决下面的最优化问题：



L是损失函数，f是式16.3中的自适应基模型。一些常用的损失函数如下表：



表中的一些常见损失函数，包括梯度、f的最小值和一些最小化损失的算法。对于二元分类问题，我们假设，，。对于回归问题，我们假设。

如果我们使用平方误差损失，最优估计为：



实际上这个无法计算，因为需要知道条件分布p(y|x)，所以这个有时候又被叫做人口最小化，因为这个期望值是从频率的角度来解释的。下面我们将使用boosting来估计这个条件期望。

对于二元分类，最显而易见的损失是0-1损失，但是这个损失是不可微分的。通常使用logloss来代替0-1损失，logloss是0-1损失的凸上界。在这种情况下，损失函数最小值为：





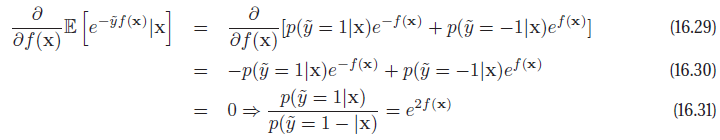
另外一种凸上界是指数损失，定义为：



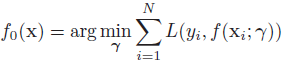
指数损失和logloss相比有一些计算上的优点。指数损失的最优值也是：



我们可以将期望损失在每一个x的梯度设置为0：



因为找到最优解f很困难，我们可以分阶段的来实现。我们定义初始值为:

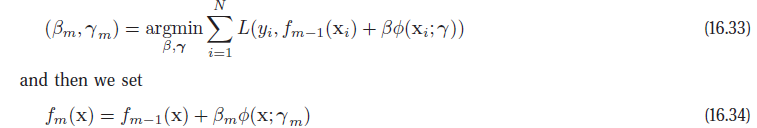


比如，如果我们使用平方误差，可以设，如果使用logloss或者指数损失，我们可以设



我们也可以使用一个更强大的模型作为我们的基准，比如GLM。

然后做一个迭代m，



关键一点是我们不在回头更改之前的参数，这就是为什么叫做向前阶段叠加模型。

我们持续这个迭代m次，实际上m是这种方法的一个主要调整参数。我们经常在一个分离的验证集上监控模型的表现，如果发现性能开始下降，那么我们就停止迭代，这叫做early stopping。

实际上，通过部分升级可以在测试集上得到更好的表现：



通常v取一个小的值，比如0.1，这叫做**Shrinkage。**

下面我们讨论如何求解方程16.33中的问题。这将取决于损失函数的形式。然而，它和f采用什么形式没有关系。

1. L2boosting
2. Adaboost
3. LogitBoost
4. Boosting as functional gradient descent
5. Sparse boosting
6. Multivariate adaptive regression trees
7. Why does boosting work so well
8. A Bayesian view