学号： 2015301000037

密级：

武汉大学本科毕业论文

关于 boosting 算法在分类问题中的比较与应用

院 (系) 名 称： 数学与统计学院

专 业 名 称： 统计学

学 生 姓 名： 许卓岩

指 导 教 师： 邓爱姣 副教授

二〇一九年五月

郑　重　声　明

本人呈交的学位论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料真实可靠。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确的方式标明。本学位论文的知识产权归属于培养单位。

本人签名: _____　　日期: _____

摘　要

本文基于 boosting 的几种算法，深入分析了其提出的理论背景与思路历程，比较这几种算法理论背景与实际表现的异同。由 Boosting 的理论特性与提出动机，将 boosting 归纳为一种非参学习方法。从统计的角度出发，得出 Adaboost 的实质是一种 stagewise forward 可加模型，其使用了指数损失来调优，具体的表现为调节样本权重。从可加模型的角度出发，将 GBM 是一种非参中的函数估计，也即在函数空间上的一种优化。其计算核心为最速下降法，但与传统的梯度优化不同，其并没有将每一步的梯度作为方向，而是用弱分类器（决策树）去拟合每一步的梯度，然后根据可加模型的思想，将弱分类器加入到现在的估计函数中进行优化。XGBoost 在 GBM 的基础上进一步拓展。其计算核心为牛顿方法，所以 XGBoost 的优化有时也被称为 Newton Boosting。XGBoost 用二次梯度来决定优化方向，使用二阶泰勒展开来拟合当前目标损失函数。运用这一思想，可以重新定义一种构建树的算法（与基尼系数，信息熵类似），此算法决定了决策树中怎样选择节点可以达到最大的准确率。在实际数据算法比较中，与我们的理论结果相符，GBM 与 XGBoost 有着最好的预测准确率，也是现在被广泛使用的原因。本文还有一些未解决问题，如 XGBoost 有着更好的收敛率，但实际操作中训练的时间较久，这种弊端是否可以避免，是否与其估计贪婪算法的运用有关。这些问题在之后需要做进一步的探究。

关键词: 机器学习; 数据挖掘; 非参函数估计; 可加模型

ABSTRACT

Based on several algorithms of boosting, this paper deeply analyzes the theoretical background and course of thought, and compares the similarities and differences between the theoretical background and actual performance of these algorithms. Based on the theoretical characteristics and motivation of Boosting, we generalize boosting into a non-parametric learning method. From a statistical point of view, we conclude that the essence of Adaboost is a stagewise forward additive model, which uses exponential loss to tune, the specific performance is to adjust the sample weight. From the perspective of the additive model, we use GBM as a non-meal function estimate, which is an optimization in the function space. Its computational core is the steepest descent method, but unlike traditional gradient optimization, it does not use the gradient of each step as the direction, but uses a weak classifier (decision tree) to fit the gradient of each step, and then according to the idea of additive model, the model adds the weak classifier to the current estimation function for optimization. XGBoost is further extended on the basis of GBM. Its computational core is Newton's method, so XGBoost's optimization is sometimes called Newton Boosting. It uses a quadratic gradient to determine the direction of optimization and a second-order Taylor expansion to fit the current target loss function. Based on this idea, we can redefine an algorithm for building trees (similar to Gini coefficients, information entropy), which determines how nodes can be selected in a decision tree to achieve maximum accuracy. In the comparison of actual data algorithms, in line with our theoretical results, GBM and XGBoost have the best prediction accuracy, which is why it is widely used now. There are still some unresolved issues in this paper. For example, XGBoost has a better convergence rate, but the actual training time is longer. Whether this kind of drawback can be avoided is related to the estimation of the application of greedy algorithm. These issues need to be further explored later.

Key words: Machine learning; Data mining; Non-parametric function approximation; Additive model

Contents

第 1 章　概述

1.1　选题来源、目的及意义

1.1.1　机器学习的背景及现状

机器学习是一种计算机系统改善自己在特定任务上统计算法和统计模型的研究，计算机系统或根据历史数据（经验数据或训练数据）产生模型，再用所得模型去预测，或根据数据在预测的过程中产生模型，是一种具体在经验学习中具体改善算法的性能。机器学习在如今各个方面有着广泛的应用，而统计，数学优化是机器学习算法背后的理论核心。

1.1.2　研究分类问题算法的意义

在机器学习，数据挖掘中，分类问题是其中的一个主要任务之一。分类问题，指对于已知标签（类别）的训练样本，根据其其他属性或特征，结合其标签训练出相应的分类模型，在使用模型根据测试样本的属性去预测其应该属于的类别。分类问题是监督学习中的核心问题，其作为机器学习中的核心问题，在各个方面都有着极其广泛的应用，如自然语言处理，图像识别，自动驾驶，垃圾邮件分类等。

1.1.3　论文的研究意义

集成学习为机器学习中解决分类问题的一类标志性算法，它通过构建多个不同的分类模型（基分类器或弱分类器），再将这些分类器进行某种组合而得到强分类器进行预测。集成学习通过这种组合，可以获得比单一分类器更加稳定和优良的输出结果。根据个体学习器的生成方式和依赖关系，集成学习的算法可分为两大类：1. 个体学习器间存在强依赖关系，学习器串联生成的方法，如 Boosting，2. 个体学习器间不存在强依赖关系，可同时生成的方法，如 bagging 和随机森林。Boosting 作为集成学习的一大代表性算法，有着极大的优势和广泛的应用，除了 boosting 最显著的代表 Adaboost，还现今再各个领域都广为使用的 GBM 和 Xgboost。

从统计的角度来看，boosting 可以看成一种非参数学习算法，此方法不依赖于数据的特征或分布。随着现今数据种类的多样化，数据的分布或特性往往杂乱而不可知，这就加大了基于数据分布的参数推断或参数估计的难度，也提高了 boosting 这类非参算法的有效性。从统计的思路探究 boosting 这类机器学习算法，不仅可

以有助于理解这类算法背后的理论推导，也有助于理解这类算法提出的目的和意义，并且有助于我们比较几种算法的异同，在现有的算法上探究。

1.2 国内外研究现状

Boosting 算法的思想起源于 Kearns 和 Valiant 在 1989 年提出的一个问题：“一系列的弱分类器是否可以创造出一个强分类器？”，弱分类器的定义是分类正确率稍大于 50%（只比随即猜测表现稍好）的分类器，强分类器指有着较好的分类结果的分类器。这个问题在 1990 年 Schapire 的论文中得到了肯定的回答，这篇论文对 boosting 的发展有着重要的意义。1996 年，Freund 和 Schapire 在论文“A decision-theoretic generation of on-line learning and an application to boosting”中首先提出 Adaptive boosting（Adaboost）算法，在机器学习领域中受到极大关注。

在 Adaboost 因其出众的效果被大众所熟知之后，Gradient boosting（GBM）也在 Adaboost 的基础上被提出，虽然都属于集成学习 boosting，但两种方法的思想有些许的区别，GBM 从可加模型，函数估计的角度出发，其被 Breiman 首先提出，他认为 boosting 可被解释为在不同损失函数下的优化方法，可以用优化梯度的思想去解决。之后，显式的算法表达在 1999 年被 Jerome H. Friedman 在论文中完善。

eXtreme Gradient Boosting（XGBoost）最初由华盛顿大学的陈天奇博士提出，是一种可扩展到端到端的 treeBoost 系统，在 kaggle 的竞赛因其出众的效率和准确率被大众所熟知。在 2015 年的 kaggle 竞赛中，29 个挑战成功并在 kaggle 博客上发表的解法中，17 都使用了 XGBoost。在这些解法中，8 个单独的用了 XGBoost，其他一些将 XGBoost 与其他方法结合为一些集成算法。作为比较，第二广泛运用的算法深度神经网络用于其中 11 个解法。并且，胜利的团队还指出，XGBoost 与其他算法组成的集成方法，表现结果只稍稍优于完美配置的 XGBoost 算法。

如今，GBM 和 Xgboost 被广泛使用在机器学习领域的各个方面，在取得良好效果的同时，其中具体的原因还需要我们去进一步了解与探究。

1.3 论文主要内容与研究思路

本文基于 boosting 的几种算法，深入分析了其提出的理论背景与思路历程，比较这几种算法理论背景与实际表现的异同，并尝试在这几种算法的基础上结合并改进。

第一章主要介绍了此算法的背景由来与提出思路，简述了 boosting 算法的广泛性与研究现状，从统计的角度看，这是一种非参学习方法，从统计的内核去理解算法并进行探究是本文的主要目的。第二章本文从损失函数的角度深入分析了 Adaboost，GBM 的理论支撑与提出思路，从统计、优化的角度对这几种算法的思路进行剖析。第三章本文详细介绍了 Xgboost 的理论背景，与 Adaboost 和 GBM 的异同。第四章本文将这几种算法拟合实际数据，比较这几种算法的表现是否符合理论结论，并将这几种算法与其它一些方法相比较。第五章总结了本文所做的工作及研究结论，并进一步提出研究方向。

第 2 章　Adaboost 与 GBM 算法的理论背景

在数据挖掘领域中，所需要的算法不仅要求有较高的预测精度，还需要有较强的解释性，这一点是一些黑箱方法如神经网络所不具备的。所以，在解释性方面，神经网络在数据挖掘中的作用有限。决策树是机器学习中一大传统算法，他有着较强的解释性，对数值变量和分类变量都有着较好的把控，并且，结果不会随着相应变量的一些转化而改变，在数据挖掘中有着较好的应用空间。

然而，在预测中，决策树往往没有较高的精度，boosting 算法通过牺牲一些学习速度，解释性来改善这一情况。Boosting 是近二十年来最有效的统计学习算法之一，在最初它被提出用来解决分类问题，但现在同样可以解决回归问题（这两类问题有着紧密联系）。boosting 问题的主要思想是将许多弱分类器结创造一个有效的判断法则，这使得 boosting 与 bagging，stacking 等集成学习算法有许多相似之处，但随着探究的深入，我们会发现这些方法之间是有本质差异的。

2.1　Adaboost 主要内容及提出思路

我们从 boosting 最典型，最具代表性的 adaboost 开始探究，其中运用最为广泛的版本就是 “Discrete Adaboost” 或称 “Adaboost.M1.” [Freund and Schapire 1996b]. 考虑二分类问题，假设我们有 n 个训练样本 $(x_1, y_1), \ldots, (x_n, y_n)$, 其中 x_i 是表示一系列特征（属性）的向量，y_i 是每个样本的标签，取值为 1 或-1. 我们定义

$$F(x) = \sum_{i=1}^{M} c_m g_m(x)$$

其中 $g_m(x)$ 指每一次计算的弱分类器，其输入每一个样本的属性可预测其标签，预测结果只比随即猜测稍好一些 ($error \leq 50\%$) , 其中 c_m 是每一个弱分类器所占的权重。Adaboost 在每一次训练中将不同权重的样本训练弱分类器，一共进行 M 次训练，在第一次训练赋予每个样本相同权重，在每一次训练后，赋予错判样本更高的权重，赋予对判样本更少的权重，再将修改权重的样本投入到下一个弱分类器中去训练。最终的强分类器是 M 个弱分类器的线性组合（最终强分类器的结果是 M 个弱分类器结果的加权求和），最终结果为 $sign(F(x)) = sign(\sum_{i=1}^{M} c_m g_m(x))$.

许多统计学家探索了以决策树为弱分类器的 adaboost 的表现，并且发现它对于单颗决策树而言有着更好的预测精度。事实上, Breiman(NIPS Workshop,1996) 将

Adaboost 比作“世界上可以直接使用的最好的现成的分类器”。许多例子都证明了随着弱分类器的增加，Adaboost 的误差在持续减小而不是上升。

算法 1 展示了 Adaboost.M1 在二分类问题上的详细步骤 (Discrete Adaboost):

Algorithm 1 *Adaboost.M1*.

1: Initialize the observation weights $w_i = 1/N, i = 1, \ldots, N$
2: **for** $m = 1, 2, \ldots, M$ **do**
3: (a) Fit the weak classifier $g_m(x) \in \{-1, 1\}$ on the training data with weights w_i.
4: (b) Compute error:

$$err_m = E_w[1_{(y \neq g_m(x))}] = \frac{\sum_{i=1}^{N} w_i 1(y_i \neq g_m(x_i))}{\sum_{i=1}^{N} w_i}.$$

5: (c) Compute $c_m = log((1 - err_m)/err_m)$.
6: (d) Set $w_i \leftarrow w_i exp[c_m 1(y_i \neq g_m(x_i))], i = 1, 2, \ldots, N$, renormalize weights so that $\sum_i w_i = 1$.
7: Output the classifier $sign[\sum_{i=1}^{M} c_m g_m(x)]$

2.2 可加模型

在统计中，可加模型是一种非参回归方法，可加模型将多种基函数或简单光滑器结合，组成一个复杂的回归模型，达到预测的目的。接下来从几步，详细引入可加模型的概念。

2.2.1 可加回归模型与 backfitting 算法

我们首先考虑一个非参回归问题，响应值 y 为连续变量，解释变量 X 与 Y 存在某种联合分布，我们想要探究 E(Y|X) = F(X) 中 F(X) 的具体形式，其中 $F(x) = \sum_{j=1}^{p} f_j(x_j)$. 其中 $f_j(x_j)$ 指对 p 个变量中其中一个变量 x_j 的作用函数，或者更广义地说，每一个 f_j 都是对输入变量 X 的其中一个子集的作用函数。Backfitting algorithm 由 Leo Breiman 和 Jerome Friedman 于 1985 年提出:

$$f_j(x_j) \leftarrow Smooth\{E[y - \sum_{k \neq j} f_k(x_k)|x_j]\} \ \ for j = 1, 2, \ldots, p$$

通常来说，Smooth 使用三次样条光滑器，或局部多项式回归或核回归方法。

对于应用更广泛的情况，我们可以将基函数的显示表达写的更具体一些，通常情况下，基函数 $f_m(x)$ 有着以下的形式:

$$f_m(x) = \beta_m h(x, a_m) \tag{2.1}$$

其中 β_m 是函数乘子，$h(x, a_m)$ 是带有参数 a 的对于 x 的简单函数。举几个简单的例子：在单隐藏层神经网络中，$b(x, a) = \sigma(a^T x)$, 其中 $\sigma(\cdot)$ 是 sigmoid 函数，a 是输入变量 x 的线性组合或仿射变换；在多元自适应回归样条中，$h(x, a)$ 是被截的指数基，a 是变量参数，如节点数等。在 Adaboost 中，$h(x, a)$ 通常是决策树，a 是内部结点的分类法则和叶节点的预测值等。

在分类问题中, $E(1[y = j]|x) = P(y = j|x)$, 考虑而分类问题, 我们引入 logistic 回归的思想，可加 logistic 回归模型的形式如下：

$$log(\frac{P(y = 1|x)}{P(y = -1|x)}) = \sum_{i=1}^{M} f_m(x)$$

同样的我们有 $p(x) = P(y = 1|x) = \frac{e^{F(x)}}{1+e^{F(x)}}$。

2.2.2 Stagewise 可加模型

在这里，我们引入 stagewise 概念。首先我们回顾一下线性回归中熟知的 stepwise（逐步回归），在多元线性回归中，stepwise 每次添加或减少一个变量，将整个变量添加到原有的变量集中，从几何角度思考，若将变量 x_k 添加到 $y = \beta_0 + \beta_1 x_1 + \ldots + \beta_m x_m$ 中，则 x_k 的系数等于 y 投影到 x_k 空间上的值，在很多时候直接删除或引入一个变量容易造成过拟合或欠拟合，stagewise 方法将 x_k 前面的系数减小, 将一大步变成一小步。Stagewise 在解 LASSO 问题的最小角回归中有着充分的体现，在这里我们不做过多说明，只需了解此方法走一小步的基本思想。在 Boosting 方法中，主要用到的是 Forward stagewise, 在每次添加基函数或称弱分类器（前面所说的新变量）时不修改之前基函数的形式与系数。下面是 Forward Stagewise 可加模型的详细算法：

Algorithm 2 *Forward stagewise additive modeling*

1: Initialize $F_0(x) = 0$.
2: **for** $m = 1, 2, \ldots, M$ **do**
3: (a) Compute

$$(\beta_m, a_m) = \arg\min_{\beta, a} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i, a))$$

4: (b) Set$F_m(x) = F_{m-1}(x) + \beta_m h(x, a_m)$

如算法 2 所示，在每一次迭代中，都寻找最优的基函数 $b(x, a_m)$ 和函数乘子 β_m，前面 m-1 步的基函数和乘子不再改变。

对于平方损失（L2 损失）函数来说：$L(y, F(x)) = (y - F(x))^2$, 则我们有：

$$\begin{aligned} L(y_i, F_{m-1}(x) + \beta h(x_i, a)) =& (y_i - F_{m-1}(x_i) - \beta h(x_i, a))^2 \\ =& (r_{im} - \beta h(x_i, a))^2 \end{aligned}$$

其中 $r_{im} = y_i - F_{m-1}(x_i)$ 是再第 m 次训练中当前模型对于第 i 个样本回判的残差。可以得出，在 L2 损失时，新加入的基函数 $h(x, a_m)$ 主要特征是拟合当前残差。这个思想是 Gradient Boosting 中使用 L2 损失的主要理论背景，我们会在之后的章节中详细探究。在分类问题中，L2 损失并不是一个非常好的选择，在下一章节，我们将探究 Adaboost 和可加模型的关系。

2.3 指数损失与 Adaboost

引入了 Adaboost 的主要内容和可加模型的基本概念后，在这一章节将证明 Adaboost.M1. 就是一种 Forward stagewise 可加模型。这个可加模型运用指数损失作为损失函数:

$$L(y, F(x)) = e^{-yF(x)}$$

要证明运用此损失函数的可加模型就是 Adaboost.M1., 首先我们需要引入一个引理：

Lemma.1. $E(e^{-yF(x)})$ is minimized at:

$$F(x) = \frac{1}{2} log \frac{P(y=1|x)}{P(y=-1|x)} \tag{2.2}$$

$$Hence: P((y=1|x) = \frac{e^{2F(x)}}{1+2F(x)} \tag{2.3}$$

Proof. 由于我们需要利用 x 去预测相应变量 y, 此问题可转化为求 $E(e^{-yF(x)}|x)$ 的最小值，由于：

$$E(e^{-yF(x)}|x) = P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)} \tag{2.4}$$

$$\frac{\partial E(e^{-yF(x)}|x)}{\partial F(x)} = -P(y=1|x)e^{-F(x)} + P(y=-1|x)e^{F(x)} \tag{2.5}$$

求相应值使得一阶导等与 0, 且二阶导恒正，lemma 结论得证。我们可以看出，如果忽略常数 2，使得损失函数最小的 F(x) 是 P(y=1|x) 的 logit 变换。这也解释了为什么 Adaboost 最后使用 sign(F(x)) 作为判断输出的结果。

对于 Adaboost 来说，基函数或弱分类器用 $g_m(x) \in \{-1, 1\}$ 指代。使用指数损失时，我们需要解决

$$(\beta_m, g_m) = \arg\min_{\beta,g} \sum_{i=1}^{N} \exp\{-y_i(F_{m-1} + \beta g(x_i))\}$$

由于在 forward stagewise 中，第 m 步添加的分类器和系数 β_m 对前面不造成改变，则可化为下式：

$$(\beta_m, g_m) = \arg\min_{\beta,g} \sum_{i=1}^{N} w_i^{(m)} \exp\{-\beta y_i g(x_i)\} \tag{2.6}$$

其中 $w_i^{(m)} = \exp(-y_i F_{m-1}(x_i))$. 由于每一步的 $w_i^{(m)}$ 与 β 与 $g(x)$ 都无关，其可被看成作用于每一个样本上的权重，此权重与 $F_{m-1}(x_i)$ 有关，所以样本的权重在每一次迭代后都会发生更新。公式 [2.6] 的优化可分为基函数和系数 β 两部分。首先，对于任何的 β 大于 0，我们有

$$g_m = \arg\min_{g} \sum_{i=1}^{N} w_i^{(m)} 1[y_i \neq g(x_i)] \tag{2.7}$$

基分类器可由带有权重的样本训练得到，寻找步长 β 就变成了我们的重点。令 Q 指代公式 [2.6] 右边的损失部分，即 $Q = \sum_{i=1}^{N} w_i^{(m)} \exp\{-\beta y_i g(x_i)\}$，由于 $y_i, g(x_i) \in \{-1, 1\}$ 我们有：

$$Q = e^{\beta} \sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq g(x_i)) + e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} 1(y_i = g(x_i)) \tag{2.8}$$

为了寻找 $\beta = \arg\min_{\beta} Q$, 我们有：

$$\begin{aligned}
0 = \frac{\partial Q}{\partial \beta} =& e^{\beta} \sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq g(x_i)) - e^{-\beta} \sum_{i=1}^{N} w_i^{(m)} 1(y_i = g(x_i)) \qquad (2.9)\\
=& e^{\beta} \sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq g(x_i)) - e^{-\beta} (\sum_{i=1}^{N} w_i^{(m)} - \sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq g(x_i)))\\
e^{2\beta} =& \frac{\sum_{i=1}^{N} w_i^{(m)} - \sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq g(x_i))}{\sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq g(x_i))} \qquad (2.10)\\
=& \frac{1 - err_m}{err_m}\\
where \quad err_m =& \frac{\sum_{i=1}^{N} w_i^{(m)} 1(y_i \neq g(x_i))}{\sum_{i=1}^{N} w_i^{(m)}} \qquad (2.11)
\end{aligned}$$

则我们有 $\beta_m = \frac{1}{2} log \frac{1-err_m}{err_m}$, 每一步强分类器的更新为：$F_m(x) = F_{m-1}(x) + \beta_m g_m(x)$, 在 Adaboost 情况下则为样本权重的更新:

$$w_i^{(m+1)} = \exp(-y_i F_m(x_i)) = w_i^{(m)} \cdot e^{-\beta_m y_i g_m(x)}$$

由于 $-y_i g_m(x_i) = 2 \cdot 1(y_i \neq g_m(x_i)) - 1$, 上式也可写成:

$$\begin{aligned} w_i^{(m+1)} &= w_i^{(m)} \cdot e^{\beta_m(2 \cdot 1(y_i \neq g_m(x_i))-1)} \\ &= w_i^{(m)} \cdot e^{2\beta_m 1(y_i \neq g_m(x_i))} e^{-\beta_m} \\ &= w_i^{(m)} \cdot \exp[log(\frac{1-err_m}{err_m}) 1(y_i \neq g_m(x_i))] \cdot e^{-\beta_m} \end{aligned} \tag{2.12}$$

因在权重迭代中需重新标准化使其和为 1, 则可忽略项 $e^{-\beta_m}$, 可以看出, 公式 [2.12] 与 算法 1:Adaboost.M1. 中权重迭代的公式相同。

由此，我们得出 Adaboost.M1. 是一种运用指数损失的 forward stagewise 可加模型。

2.4 损失函数

可加模型使用不同的损失函数, 会产生不同的算法, 当使用指数损失时, 产生的算法为 Adaboost.M1. 在这一章节中，我们将简要了解几种不同的损失函数，以及他们的作用。

2.4.1 分类问题

在分类问题中, 我们的目的是使得错判样本尽可能少, 以二分类问题为例（相应变量为-1/1），我们首先引入 “margin” 的概念,$margin = yF(x)$, 在分类问题中, 我们的目的是构造 F(x) 使得 margin 越多越好，margin 在分类问题中的作用就类似于 $y - F(x)$ 在回归问题中的作用，我们的损失函数要随着 margin 的增大而减小。在这一章节中，我们将仔细探究指数损失函数的一些性质，以及其与一些分类问题常见的损失函数（如交叉熵）的关系。且通过这些探究，我们也能深入理解为何 Adaboost 选取指数损失作为损失函数。

Adaboost.M1. 算法最开始的提出是并不是基于指数损失函数，在 Adaboost 产生五年后，它与运用指数损失的 forward stagewise 可加模型的等价性才被提出。指数损失最主要的优点之一是它的算法步骤简便性：其使得 Adaboost 通过简单的重赋予样本权重来实现算法。但是，通过探究指数损失的统计内涵，以及在指数损失下最优解的寻找，我们可以理解用其估计的准确性。

回想我们在章节 2.3 中，我们将第 m 步添加的分类器及其系数写成广义形式 $f_m(x) = \beta_m g_m(x)$. 且通过 forward stagewise 模型每一步对前面不做改变的性质, 我

们有:

$$f_m = \arg\min_f \sum_{i=1}^{N} \exp\{-y_i(F_{m-1} + f(x_i))\} = \arg\min_f \sum_{i=1}^{N} \exp\{-y_i f(x_i)\}$$

由 lemma 1, 我们有:

$$f_m(x) = \frac{1}{2} log \frac{P(y=1|x)}{P(y=-1|x)}$$
$$Hence: P((y=1|x) = \frac{e^{2f(x)}}{1+2f(x)}$$

除了我们刚才介绍的指数损失外，还有广义线性回归中经常使用的二项对数似然和 deviance(或称交叉熵) 可达到这一目的。且与上述的指数损失在同样的条件下取得最小值 (均为 lemma 1 的结果)。下面我们将推导这两种损失的等价性。

将 F 看作 $p(x) = P(y = 1|x)$ 的 logit 变换，我们有:$p(x) = \frac{1}{1+e^{-2f(x)}}$, 定义 $y' = (y+1)/2$ 为转化后的 0/1 响应变量，则二项对数似然函数的形式为

$$-l(y, p(x)) = y' log(p(x)) + (1-y') log(1-p(x))$$

其中 l 指 loss function。与广义线性回归思想相同，通常需要找到 p(x) 最大化上式，常用迭代法求解。由上述关系，我们推导出交叉熵损失为:

$$\begin{aligned}
l(y, p(x)) =& -y' log(p(x)) - (1-y') log(1-p(x)) \\
=& -y' log(\frac{p(x)}{1-p(x)}) - log(1-p(x)) \\
=& -(y+1) f(x) + log(1+e^{2f(x)}) \\
=& log\{e^{-yf(x)}(e^{-f(x)} + e^{f(x)}\} \\
=& \begin{cases} log(1+e^{-2f(x)}) & y=1 \\ log(1+e^{2f(x)}) & y=-1 \end{cases} \\
=& log(1+e^{-2yf(x)}) \\
=& l(y, f(x))
\end{aligned}$$

可等价化为 deviance。

则我们有 deviance$E_{y|x}[-l(y, f(x))]$ 的最优解与指数损失 $E_{y|x}[e^{-yf(x)}]$ 相同，使用其中任一种准则可得到相同的结果。值得一提的是，e^{-yf} 本身并不是任何一种 log-likelihood，因其不是任何一种伯努利分布变量似然函数的 log 值。

2.4.2 回归问题

在回归问题中，最常用到的两种损失函数是平方误差 $L(y, f(x)) = (y - f(x))^2$ 和绝对值误差 $L(y, f(x)) = |y - f(x)|$. 平方误差更着重于有较大残差的样本，当训练样本中有异常点时模型稳定性会受到较大影响，绝对值误差在这方面有着较好的表现。在很多统计文献中，提出了很多回归损失函数，这些函数对异常点有较强的抵抗性，并且在计算误差时，与高斯分布下的平方误差一样有效。其中一个代表就是用于 M 回归的 Huber 损失函数 (Huber,1964) 运用此损失函数的 boosting 问题我们将在后面的章节做详细探究。

前面我们提到，在平方损失中，基函数主要拟合当前残差。在指数损失中，基函数主要拟和调整过后的加权样本。运用回归问题中这几种回归损失函数的可加模型在 Gradient Boost 中有着广泛的运用。

2.5 GBM：一种函数估计

函数估计指一种在函数空间上而不是参数空间上的数值优化。GBM 全称 Gradient Boosting Machine, 是一种运用可加模型和结合不同损失函数的广义梯度下降函数算法。其中特别的算法有回归中的平方损失，绝对值损失和 Huber M 损失，和分类中的对数似然。当基分类器为回归树时，GBM 对于回归问题和分类问题都有着较强的预测精度，解释性和稳定性。GBM 与 Adaboost 的主要区别在于损失函数的不同，且由于直接优化目标函数的困难性，GBM 使用梯度下降法作为算法核心。

在这一章节，我们将推广可加模型到一种函数估计，引入梯度下降的思想，探究 GBM 的构造原理。

在函数估计中，我们的最终目的是寻找优化函数：

$$F = \arg\min_{F} E_{y,x} L(y, F(x)) = \arg\min_{F} E_x [E_y L(y, F(x))|x]$$

这是一种函数空间上的非参方法，在实际应用中，当 (y,x) 的联合分布的信息蕴含在有限的样本 $\{y_i, x_i\}_1^N$ 中时：

$$F = \arg\min_{F} \sum_{i=1}^{N} L(y_i, F(x_i)) \tag{2.13}$$

在 Boosting 算法中，$F_m(x) = \sum_{m=0}^{M} f_m(x)$, 其中 $F_m(x) = f_m(x)$ 是迭代函数初值，之后每一步迭代都更新一次 f(x) 并添加到现有的 F(x) 中。在 Adaboost 中更新可被理解为改变样本权重，在 GBM 中，它有着不同的含义。

2.5.1 梯度下降法

在探究 GBM 的原理前，我们先引入一个优化方法：梯度下降法。这个算法是很多机器学习算法的优化方法，也是 GBM 使用的优化方法。它定义了 GBM 每一步优化的具体步骤，在第 m 步时：$f_m = -\rho_m g_m$, 其中 ρ_m 为标量代表每一步步长，$g_m = (g_{1m}, \ldots, g_{Nm})$ 为损失函数在 m-1 步的梯度，表示为：

$$g_{im} = g_m(x_i) = \left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{m-1}(x_i))} \tag{2.14}$$

步长 ρ_m 由下式得出：

$$\rho_m = \arg\min_{\rho} E_{y,x}(L(y, F_{m-1} - \rho g_m)) = \arg\min_{\rho} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) - \rho g_{im}(x_i)))$$

其中 N 为样本个数。

由此，我们得到第 m 步模型的更新为 $F_m = F_{m-1} - \rho g_m$.

2.5.2 GBM 与梯度下降法

最速下降法原本是对于某个函数进行的参数优化或数值优化，而在 GBM 中，我们需要解决的是函数优化，函数估计的问题，回顾式 [2.13], 在每一步迭代中都将新的基函数加入原有函数中，我们仍用式 [2.1] 的思想来表示基分类器 $f_m(x) = \beta_m h(x, a_m)$，其中 h 为基分类器，a 为分类器参数，$\beta$ 为分类器权重，在这里可以理解为步长:$F_m = F_{m-1} + \beta h_m(x, a_m)$. 考虑到 forward stagewise 可加模型，我们在 m 步需要解决

$$(\beta_m, a_m) = \arg\min_{\beta, a} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i, a)) \tag{2.15}$$

直接解出往往比较困难，我们在这里借用最速下降的思想，每一次迭代向负梯度方向跨出一步，我们得到每一步的梯度如式 [2.14] 所示。

然而，式 [2.14] 只在训练样本的数据点上有定义，即考虑最速下降法中模型的更新 $F_m = F_{m-1} - \rho g_m$，估计函数 F(x) 只在 x 等与样本点的数值时对函数有更新，在 x 等与其他值时，估计函数 F(x) 的形式保持不变。如果我们的目标是使其训练误差越小越好，使每一步的改变 $f_m = g_m$，梯度下降法便可满足需求，而我们的目标是将它应用于新数据中。

一个有效的方案是在每一步引入一个弱分类器，使其尽可能去拟合这一步得出的负梯度。即在现有数据上，使得每一步的弱分类器与当前的梯度（函数前进

方向）尽可能“相近（或高度相关）”，在“相近”的定义上，我们运用了平方损失法则。则在每一次训练中，可化为如下两步：

1. 训练基分类器使得与当前负梯度最相近：

$$a_m = \arg\min_{a_m,\beta} \sum_{i=1}^{N} [-g_m(x_i) - \beta h(x_i, a)]^2$$

其中步长 β 为固定值。

2. 寻找步长使得其达到式 [2.15] 标准（对于不同损失函数）：

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i, a_m))$$

这样，就将复杂的式 [2.15] 双参数优化化为了单参数优化，通过引入最速下降法和平方损失衡量“相近”这一思想，且当损失函数在大部分定义域可微时，将 GBM 表示为易解决的 forward stagewise 可加模型。我们用 $\{\widetilde{y}_i = -g_m(x_i)\}_{i=1}^{N}$ 表示每一步弱分类器拟合的负梯度，将其称为伪响应变量 (pseudo response)，该算法如下：

Algorithm 3 *Gradient Boosting*

1: Initialize $F_0(x) = \arg\min_{\beta} \sum_{i=1}^{N} L(y_i, \beta)$.
2: **for** $m = 1, 2, \ldots, M$ **do**
3: (a)

$$\widetilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{m-1}(x_i))}, i = 1, \ldots, N$$

4: (b) $a_m = \arg\min_{a,\beta} \sum_{i=1}^{N} [\widetilde{y}_i - \beta h(x_i, a)]^2$
5: (c) $\beta_m = \arg\min_{\beta,a} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i, a_m))$
6: (d) $F_m(x) = F_{m-1}(x) + \beta_m h(x, a_m)$

平方损失 LS 回归 当使用平方损失时，$L(y,F)=(y-F)^2/2$, 初始 $F_0(x)=y$，$\widetilde{y}_i=y_i-F_{m-1}(x_i)$, 算法 3 中第 3 行的伪响应变量为当前计算的残差，那么 GBM 在平方损失上的算法就变成了不断拟合当前的残差，算法 3 中 4,5 行可表示为:

$$(\beta_m,a_m)=\arg\min_{\beta,a}\sum_{i=1}^{N}(\widetilde{y}_i-\beta h(x_i,a))^2 \tag{2.16}$$

绝对值损失 LAD 回归 当使用绝对值损失时，$L(y,F)=|y-F|$, 初始 $F_0(x)=y$，$\widetilde{y}_i=sign(y_i-F_{m-1}(x_i))$，则算法变为不断拟合当前残差的符号，算法 3 中第 4 行不变，第 5 行表示为:

$$\begin{aligned}\beta_m=&\arg\min_{\beta}\sum_{i=1}^{N}|y_i-F_{m-1}(x_i)-\beta h(x_i,a_m)| \\ =&\arg\min_{\beta}\sum_{i=1}^{N}|h(x_i,a_m)|\cdot|\frac{y_i-F_{m-1}(x_i)}{h(x_i,a_m)}-\beta| \\ =&median_w\left\{\frac{y_i-F_{m-1}(x_i)}{h(x_i,a_m)}-\beta\right\}_1^N,\ \ w_i=h(x_i,a_m)\end{aligned} \tag{2.17}$$

其中 $median_w$ 表示以 w 为权重的加权中位数。

2.6 回归树

前面提到，在 GBM 中，用决策树（尤其是回归树）作为基分类器时有着较好的预测精度。在这一章节，我们讨论决策树作为基分类器时这一特殊情况。此时，算法有着一些变动。

回顾决策树,其将样本通过叶节点分为了不同互不相交的区域 $R_j,j=1,2,\ldots,J$, 每一个区域都有对应的预测的标签，其预测法则可写为:$x\in R_j\Rightarrow f(x)=b_j$。则每一棵树可被写为 $h(x;\{b_j,R_j\})=\sum_{i=1}^{J}b_j1(x\in R_j)$，将其带入算法 3，则算法 3 的第 6 行变为

$$F_m(x)=F_{m-1}(x)+\beta_m\sum_{j=1}^{J}b_{jm}1(x\in R_{jm}) \tag{2.18}$$

其中 $\{R_{jm}\}_{j=1}^{J}$ 指第 m 次迭代被叶节点分为的区域。当我们使用决策树时，b_{jm} 的寻找往往比较简单。如使用回归树时，每棵树内的决策法则为平方和损失，则有 $b_{jm}=ave(\widetilde{y}_i|x\in R_{jm})$。

β_m 的选择与算法 3 中第 5 行不变。则式 [2.18] 变为

$$F_m(x)=F_{m-1}(x)+\sum_{j=1}^{J}\gamma_{jm}1(x\in R_{jm})$$

其中 $\gamma_{jm} = \beta_m b_{jm}$。上式可理解为在第 m 步，添加 J 个独立的基函数到 F(x) 而不是 1 个，我们可将算法 3 的第 5 行化为：

$$\{\gamma_{jm}\}_1^J = \arg\min_{\{\gamma_j\}_1^J} = \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \sum_{j=1}^{J} \gamma_j 1(x \in R_{jm}))$$

算法 3 的第四行 a_m 变为决策树不同的划分区域。由于拟合当前残差的法则为平方损失法则，我们选取回归树作为基分类器。

由于决策树的不同区域互不相交，上式可化为:

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \tag{2.19}$$

则算法 3 可被写成：

Algorithm 4 *Gradient Boosting with regression tree(TreeBoost)*

1: Initialize $F_0(x) = \arg\min_\beta \sum_{i=1}^{N} L(y_i, \beta)$.
2: **for** $m = 1, 2, \ldots, M$ **do**
3: (a)
$$\widetilde{y}_i = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{m-1}(x_i))}, i = 1, \ldots, N$$
4: (b) $\{R_{jm}\}_1^J$ = J terminal node tree($\{\widetilde{y}_i, x_i\}_1^N$),
5: i.e. fit a regression tree to ($\{\widetilde{y}_i, x_i\}_1^N$) to determine $\{R_{jm}\}_1^J$
6: (c) $\gamma_{jm} = \arg\min_\gamma \sum_{x \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$
7: (d) $F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} 1(x \in R_{jm})$

平方损失 LS TreeBoost　当使用平方损失时,$\widetilde{y}_i = y_i - F_{m-1}(x_i)$，树的构建为不断拟合当前残差，算法第 6 行中 γ_{jm} 的选取为:

$$\gamma_m = mean_{x_i \in R_{jm}}\{y_i - F_{m-1}(x_i)\} \tag{2.20}$$

绝对值损失 LAD TreeBoost　当使用绝对值损失时，$\widetilde{y}_i = sign(y_i - F_{m-1}(x_i))$, 树的构建为不断拟合当前残差的符号，算法第 6 行中 γ_{jm} 的选取为:

$$\gamma_m = median_{x_i \in R_{jm}}\{y_i - F_{m-1}(x_i)\} \tag{2.21}$$

使用绝对值损失的 TreeBoost 算法具有高度稳定性（鲁棒性）。树的结点根据中位数进行更新。另一种使用绝对值损失的算法抛弃了平方损失法则作为拟合当前残差损失函数的思想而直接构造树来最小化绝对值损失函数：

$$tree_m(x) = \arg\min_{J\ node\ tree} \sum_{i=1}^{N} |y_i - F_{m-1}(x_i) - tree(x_i)|$$

然而，使用绝对值损失的算法 4 具有更快的计算速度，因其使用平方损失作为拟合残差的损失函数，在开始树的构建，结点的选区方面，直接构造回归树比根据绝对值选取结点更快。

2.7 M 回归

对于正太误差项的样本，M 回归在长尾误差分布和离群点的影响下仍有较好的表现和较高的效率。M 回归主要运用 Huber 损失函数，其定义如下:

$$L(y,F(x)) = \begin{cases} \frac{1}{2}(y-F(x))^2 & for \ \ |y-F(x)| \le \delta, \\ \delta|y-F(x)| - \frac{1}{2}\delta^2 & otherwise \end{cases}$$

则有伪响应变量 $\widetilde{y}_i$:

$$\begin{aligned} \widetilde{y}_i &= -\left[\frac{\partial L(y_i,F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{m-1}(x_i))} \\ &= \begin{cases} y_i - F_{m-1}(x_i) & for \ \ |y_i - F_{m-1}(x_i)| \le \delta, \\ \delta(y_i - F_{m-1}(x_i)) & otherwise \end{cases} \quad i = 1,\ldots,N \end{aligned}$$

同理，步长的寻找变为

$$\beta_m = \arg\min_{\beta} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i, a_m))$$

上述式子使用标准迭代方法求解，解法在 [Huber 1964] 中有详细阐述。在这里，我们更多考虑到将优化解法与此问题相结合。

转折点 δ 的值定义了哪些点会被当做离群点，离群点使用绝对值损失而不是平方损失。一个好的 δ 取值往往取决于 $y - F^*(x)$ 的分布，其中 $F^*(x)$ 为我们想要估计的真实目标函数。一个较好的经验性结论往往取 δ 的值为 $|y - F^*(x)|$ 分布的上 α 分为点。其中 $(1-\alpha)$ 控制着整个过程样本中的故障点。故障点指在模型作用过程中，可被任意修改却不影响模型的整体表现。由于在实际估计中 $F^*(x)$ 未知，我们在第 m 次迭代中使用 $F_{m-1}(x)$ 作为 $F^*(x)$ 的估计，则有:

$$\delta_m = quantile_{\alpha}\{|y_i - F_{m-1}(x_i)|\}_1^N$$

由于回归树的特性，同理我们将迭代分配到每一个区域 R_{jm} 上。对于 Huber 损失函数，使用从残差中位数开始的迭代方法。令第 m 步迭代前的残差为

$$r_{m-1}(x_i) = y_i - F_{m-1}(x_i)\ , \ \ i = 1,\ldots N$$

定义中位数为

$$\widetilde{r}_{jm} = median_{x_i \in R_{jm}}\{r_{m-1}(x_i)\}$$

则在第 m 步的函数估计为:

$$\gamma_{jm} = \widetilde{r}_{jm} + \frac{1}{N_{jm}} \sum_{x_i \in R_{jm}} sign(r_{m-1}(x_i) - \widetilde{r}_{jm}) \dot{\min}(\delta_m, abs(r_{m-1}(x_i) - \widetilde{r}_{jm}))$$

其中 N_{jm} 是第 j 个结点的样本数，基于 Huber 损失函数的 M 回归的详细算法如下:

Algorithm 5 *M TreeBoost*

1: Initialize $F_0(x) = median\{y_i\}_1^N$.
2: **for** $m = 1, 2, \ldots, M$ **do**
3: (a)
$$r_{m-1}(x_i) = y_i - F_{m-1}(x_i)\ ,\ \ i = 1, \ldots N, i = 1, \ldots, N$$
4: (b) $\delta_m = quantile_\alpha\{|r_{m-1}(x_i)|\}_1^N$
5: (c) $\widetilde{y}_i = \begin{cases} y_i - F_{m-1}(x_i) & for\ \ |y_i - F_{m-1}(x_i)| \le \delta, \\ \delta(y_i - F_{m-1}(x_i)) & otherwise \end{cases} \quad i = 1, \ldots, N$
6: (d) $\{R_{jm}\}_1^J =$ J terminal node tree$(\{\widetilde{y}_i, x_i\}_1^N)$
7: (e) $\widetilde{r}_{jm} = median_{x_i \in R_{jm}}\{r_{m-1}(x_i)\},\ j = 1, \ldots, J$
8: (f)
$$\gamma_{jm} = \widetilde{r}_{jm} + \frac{1}{N_{jm}} \sum_{x_i \in R_{jm}} sign(r_{m-1}(x_i) - \widetilde{r}_{jm}) \dot{\min}(\delta_m, abs(r_{m-1}(x_i) - \widetilde{r}_{jm}))$$
9: (g) $F_m(x) = F_{m-1}(x) + \sum_{j=1}^J \gamma_{jm} 1(x \in R_{jm})$

在误差正态分布时，M 回归有着与 LS 相近的良好性质，在误差为极度长尾分布时，M 回归有着与 LAD 相近的良好性质。在数据分布为中长尾分布时，M 回归有着优于两者的表现。

2.8 logistic 下的 GBM

logistic 情况下，我们使用的是准则为似然函数（或称交叉熵损失）。在章节 2.4.1 中损失函数在分类问题的讨论中，我们验证了交叉熵损失和 deviance 的等价性，即

$$\begin{aligned} l(y, p(x)) =& -y' log(p(x)) - (1 - y') log(1 - p(x)) \\ =& log(1 + e^{-2yf(x)}) \\ =& l(y, f(x)) \end{aligned}$$

则在 logistic 中，我们需要最小化:

$$l(y, F) = log(1 + e^{-2yF(x)}),\ \ y \in \{-1, 1\}$$

其中:

$$F(x) = \frac{1}{2} log[\frac{p(y=1|x)}{1-p(y=1|x)}] \tag{2.22}$$

则经计算伪响应变量为 $\widetilde{y}_i$:

$$\begin{aligned}\widetilde{y}_i &= -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x_i)=F_{m-1}(x_i))} \\ &= \frac{2y_i}{1+exp(y_i F_{m-1}(x_i))}\end{aligned}$$

同理，步长的寻找变为

$$\begin{aligned}\beta_m &= \arg\min_{\beta} \sum_{i=1}^{N} L(y_i, F_{m-1}(x_i) + \beta h(x_i, a_m)) \\ &= \arg\min_{\beta} \sum_{i=1}^{N} log(1 + exp(-2y_i(F_{m-1}(x_i) + \beta h(x_i, a_m))))\end{aligned}$$

同理，考虑到我们在章节 2.6 提到的回归树的性质，样本通过叶节点划分为互不相交的区域 R_{jm}，在每一个区域的更新可写为:

$$\begin{aligned}\gamma_{jm} &= \arg\min_{\gamma} \sum_{x \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \\ &= \arg\min_{\beta} \sum_{x \in R_{jm}} log(1 + exp(-2y_i(F_{m-1}(x_i) + \gamma)))\end{aligned}$$

对上式运用一步 Newton-Raphson 迭代逼近，我们有:

$$\gamma_{jm} = \sum_{x \in R_{jm}} \frac{\widetilde{y}_i}{\sum_{x \in R_{jm}} |\widetilde{y}_i(2 - \widetilde{y}_i)|}$$

由此，我们有 logistic GBM 的算法如下:

Algorithm 6 *M TreeBoost*

1: Initialize $F_0(x) = \frac{1}{2} log \frac{1+y}{1-y}$.
2: **for** $m = 1, 2, \ldots, M$ **do**
3: (a)

$$\widetilde{y}_i = \frac{2y_i}{1+exp(y_i F_{m-1}(x_i))}$$

4: (b) $\{R_{jm}\}_1^J$ = J terminal node tree$(\{\widetilde{y}_i, x_i\}_1^N)$
5: (c) $\gamma_{jm} = \sum_{x \in R_{jm}} \frac{\widetilde{y}_i}{\sum_{x \in R_{jm}} |\widetilde{y}_i(2-\widetilde{y}_i)|}$, $j = 1, \ldots, J$
6: (d) $F_m(x) = F_{m-1}(x) + \sum_{j=1}^{J} \gamma_{jm} 1(x \in R_{jm})$

最终的估计得到函数 $F_M(x)$，我们可由式 [2.22] 去估计真正的概率:

$$\hat{p}(x) = \frac{1}{1 + e^{-2F_M(x)}}$$

最终预测的结果为:

$$\hat{y} = 2 \cdot 1[\hat{p}(x) > 1 - \hat{p}(x)] - 1$$

2.9 过拟合与正则化

在训练数据时，过多的拟合训练样本会使得模型时去泛化性，无法在预测中得到比较好的结果。而在 boosting 中，基分类器的叠加，基分类器之间相互的联系很容易造成过拟合。过拟合的概率往往随着迭代次数（或称基分类器的个数，在这里称为 M）的增长而增长。一个较好的选取 M 的方法是使用交叉验证，计算不同 M 在验证集上的表现，选取表现最好的 M。这与神经网络中的 early stopping 思想相近。

2.9.1 Shrinkage

控制 M 的数值不是正则化的唯一方法。与岭回归中同理，我们可以用 shrinkage 的思想。最简单的 shrinkage 通过因子 $\nu \in [0, 1]$ 来缩小基分类器对于估计的改变。当基分类器为回归树时（算法 4，5，6），算法的最后一行估计函数的更新变为:

$$f_m(x) = f_{m-1}(x) + \nu \sum_{j=1}^{J} \gamma_{jm} 1(x \in R_{jm})$$

参数 ν 控制着 boosting 的学习率（或步长），与随机梯度下降（SGD）的 learning rate decay 同理。然而，参数 ν 的选择并不是独立的，在保证同样的训练优度的情况下，更小的 ν 会造成更大的 M。一些经验性结论显示，最好的方法是选择更小的 $\nu(\nu < 0.1)$ 然后选择合适的 M 来 early stopping。这类方法的结果在回归问题和概率估计中比起普通的方法有很大的提升，但与之相应的代价是计算复杂度的提升。计算复杂度与 M 成正比，而 M 会随着 ν 的减小而提升。但在实际大规模的数据的应用中，此类方法的计算复杂度是可以接受的，这其中一部分原因是因为，在 boosting 中，基分类器大多数为单节点决策树 (stumps)，并且不需要修枝。

2.9.2 Subsampling

另一个防止过拟合的方法是 subsampling，这在训练模型时有着广泛的运用。

在这里我们引入随机梯度 boosting 的思想 (stochastic gradient boosting,Friendman, 1999), 在每一次迭代我们抽取样本的一部分（不放回抽取），并且用这一部分样本去训练回归树（同算法 4）。一个比较常见的取法是每次取总体样本的 50%。随着 N

的变大，抽取比例会随着降低以保证计算复杂度。当然，与 SGD 相同的是，随着抽取比例的降低，基分类器的方差会随之增大。此类方法可以看成样本矩阵 ($n \times p$, 其中 n 为样本数，p 为特征数) 的行（样本）subsampling。

与之改进的相应的方法是列（特征）subsampling，此类方法广泛运用于随机森林。根据使用的结果来看，列 subsampling 比传统的行 subsampling 在防止过拟合中有着更好的效果，列 subsampling 也能更好地加速计算进程。

在模拟中，我们可以看到，subsampling 在与 shrinkage 同时使用时有着更好的表现：

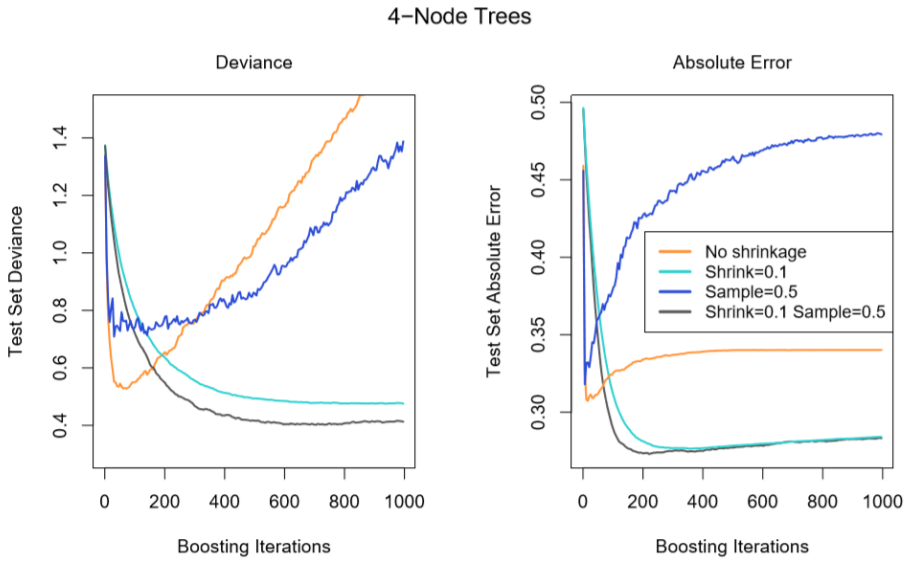

Figure 2.1　source:*The element of statistical learning*

如图所示，当基分类器为 4 结点决策树时，随着迭代次数的增加，shrinkage 有着比普通训练更好的表现，shrinkage 与 subsampling 同时使用时有着最好的表现。无论是在分类问题的 deviance(左) 和回归问题的平方损失 (右) 上都有着更快的收敛速度和更低的收敛值。

如今在正则化的过程中，我们需要决定的参数有 J,M,ν,η。在一些先前的工作中，J,ν,η 已被探索出一些合适的值作为经验性结论，而 M 需要我们在实际应用过程中自己探索。

第 3 章　XGBoost 的提出与理论背景

Boosting 是一种高效且广泛运用的机器学习方法。XGBoost 是在 boosting 的基础上进一步发展得到。XGBoost 最成功的因素就是它在很多场景的可延展性，这些好的性质源于它几个重要系统和算法的优化，包括：新颖的用于处理稀疏数据的数学习算法，一种被理论验证的分位加权树估计算法，以及使得运算更快的平行分布式计算。在这里，我们仅对其基本思想做探究。

XGBoost 更好的运用了二次梯度来优化函数，此思想被 Friedman[*a statistical view of boosting,1999*] 首次提出，XGBoost 在正则化条件下，对二次梯度有着更好的应用。

3.1　正则化下的目标函数

回顾我们在上一章节提到的可加模型：

$$F(x) = \sum_{j=1}^{p} f_j(x)$$

考虑到基分类器为回归树的特殊形式，我们有：

$$f_m(x) = \sum_{j=1}^{J} \gamma_{jm} 1(x \in R_{jm})$$

假设数据集有 n 个样本，p 个特征，我们在这里，将基分类器（回归树）的一般形式写为：

$$f(x) = \sum_{j=1}^{J} \gamma_j 1(x \in R_j) = w_{q(x)}$$
$$q : \mathbb{R}^p \longrightarrow J,\ w \in \mathbb{R}^J, w_j = \gamma_j$$

其中 q 代表把样本映射到叶编号的每一棵树的结构，每一棵树的深度相同，叶数量也相同。J 代表每一棵树叶的数量，即最后划分区域的数量。每一个 f_m 都有着其唯一的树结构和叶的预测值 w(或称 η)。与决策树不同，每一个回归树的叶预测值都为连续值，我们用 w_i(即 η_i) 代表第 i 个区域的预测值 (其中 $i = 1, \ldots, J$)。对于每一个需要预测的个体，我们将其在每一棵树对应的叶预测值 w 相加，得到最终的预测值。

回顾上一章节提到的正则化，我们除了使用 shrinkage 和 subsampling 两种方法，还可将正则化项直接加入优化目标函数中:

$$\mathcal{L}(\phi) = \sum_i l\left(y_i, F(x_i)\right) + \sum_m \Omega\left(f_m\right)$$
$$\text{where } \Omega(f) = \gamma J + \frac{1}{2}\lambda\|w\|^2$$

其中 l 为可微的凸损失函数，$\Omega\left(f_m\right)$ 衡量了基分类器的复杂度，其中 γ, λ 为正则化中罚项的系数。直观上来说，此目标函数会倾向选择预测精确，结构简单（不容易过拟合）的基分类器。当正则化中罚项系数设为 0 时，目标函数变为传统的 GBM。

3.2 Newton-boosting

Gradient Boosting Machine 运用了损失函数的一阶导数来决定优化梯度，在这里我们回顾 newton 优化方法的迭代步骤，用二次梯度来决定优化梯度。同理，我们以可加模型的角度取看待目标函数，用 $F_m(x)$ 表示在第 m 次迭代的值，则我们有:

$$\mathcal{L}^{(m)} = \sum_i l\left(y_i, F_{m-1}(x_i) + f_m(x_i)\right) + \sum_m \Omega\left(f_m\right)$$

我们需要找到每一步最适合的 f_m 来最好的拟合当前模型，最小化损失函数。

在这里，我们引入二次梯度的思想，通过直接对 $l\left(y_i, F_{m-1}(x_i) + f_m(x_i)\right)$ 在 $l\left(y_i, F_{m-1}(x_i)\right)$ 附近做二阶泰勒展开，我们有:

$$l\left(y, F^{(m-1)}(x) + f_m(x)\right) \approx l\left(y, F^{(m-1)}(x)\right) + g_m(x) f_m(x) + \frac{1}{2} h_m(x) f_m(x)^2$$

其中:

$$g_m(x) = \left[\frac{\partial L\left(y, F\right)}{\partial F}\right]_{F=F_{m-1}(x)}$$
$$h_m(x) = \left[\frac{\partial^2 L\left(y, F\right)}{\partial F^2}\right]_{F=F_{m-1}(x)}$$

则优化函数可写为:

$$\mathcal{L}^{(m)} \simeq \sum_{i=1}^{n}\left[l\left(y_i, F_{m-1}\right) + g_{mi} f_m(x_i) + \frac{1}{2} h_{mi} f_m^2(x_i)\right] + \Omega\left(f_m\right)$$

去除常数项我们有 newton-boosting 中决定迭代步骤的优化目标函数为为:

$$\tilde{\mathcal{L}}^{(m)} = \sum_{i=1}^{n}\left[g_{mi} f_m(x_i) + \frac{1}{2} h_{mi} f_m^2(x_i)\right] + \Omega\left(f_m\right)$$

回顾 R_j 为树的第 j 块划分区域，且带入 $\Omega(f_m)$ 的定义，我们有:

$$\begin{aligned}\tilde{\mathcal{L}}^{(m)} &= \sum_{i=1}^{n}\left[g_{mi}f_m(x_i)+\frac{1}{2}h_{mi}f_m^2(x_i)\right]+\gamma J+\frac{1}{2}\lambda\sum_{j=1}^{J}w_j^2 \\ &= \sum_{j=1}^{J}\left[\left(\sum_{i\in R_j}g_i\right)w_j+\frac{1}{2}\left(\sum_{i\in R_j}h_i+\lambda\right)w_j^2\right]+\gamma J\end{aligned}$$

对于一个固定的结构的树，我们只需找到最优的预测值 w_j^*:

$$w_j^* = -\frac{\sum_{i\in R_j}g_i}{\sum_{i\in I_j}h_i+\lambda},\ \ j=1,\ldots,J$$

经计算得目标函数最优值为:

$$\tilde{\mathcal{L}}^{(m)}(q) = -\frac{1}{2}\sum_{j=1}^{J}\frac{\left(\sum_{i\in R_j}g_i\right)^2}{\sum_{i\in R_j}h_i+\lambda}+\gamma J \tag{3.1}$$

式 [3.1] 可被当作一种用于衡量一个基函数是否优秀的得分函数。然而通常来说，遍历所有的树结构来找到最优解是不可能的。一个精确算法是从单节点树开始，不断添加 split 来构建树，看得分的变化，用 R_L, R_R 来表示一个区域被分开为两个区域后两个子区域的数据集，且 $R = R_L \cup R_R$, 则每一次的信息熵损失函数可被表示为

$$\mathcal{L}_{split} = \frac{1}{2}\left[\frac{\left(\sum_{i\in R_L}g_i\right)^2}{\sum_{i\in R_L}h_i+\lambda}+\frac{\left(\sum_{i\in R_R}g_i\right)^2}{\sum_{i\in R_R}h_i+\lambda}-\frac{\left(\sum_{i\in R}g_i\right)^2}{\sum_{i\in R}h_i+\lambda}\right]-\gamma \tag{3.2}$$

3.3 树构建算法

接下来，我们的任务就是怎样通过式 [3.2] 提出的熵增法则去构建树。精确的树构建算法为在所有特征上遍历所有可能的结点，此算法被称为精确贪婪算法，算法详细步骤如下:

Algorithm 7 *Exact Greedy Algorithm for Split Finding*

Input R, instance(obs) set of current node
Input d, dimension feature
gain $\longleftarrow 0$, $G \longleftarrow \sum_{i \in R} g_i$, $H \longleftarrow \sum_{i \in R} h_i$
for $k = 1, 2, \ldots, d$ **do**
    $G_L \longleftarrow 0, H_L \longleftarrow 0$
    for $j \ in \ sorted(R, by \ x_{jk})$ **do**
        $G_L \longleftarrow G_L + g_j, H_L \longleftarrow H_L + h_j$
        $G_R \longleftarrow G - G_L, H_R \longleftarrow H - H_L$
        $Score \longleftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}))$
Output:Split with max score

精确贪婪算法要求遍历连续特征上所有的结点，为了更有效率，算法必须首先观察所有数据并且根据特征值对数据排序，然后依照式 [3.2] 中梯度统计量的累加去一次访问数据。

当数据量过大时，精确贪婪算法往往变得困难，在这个基础上又提出了估计贪婪算法，此算法先根据特征分布选出适合的结点，再在这些节点中依次寻找最优结点。在这里我们不做过多描述。

第 4 章　统计模拟

4.1　数据集

为了检验我们几种算法的效果，我们将模型应用于实际数据上，对比它们的效果。我们使用的数据为 MNIST database。

MNIST 数据的全称是 Modified National Institude of Srandards and Technology, 是一个关于手写数字的大数据集，在图形处理和机器学习中都有着广泛的应用。MNIST 数据集一共有 60000 张训练图片，10000 张测试图片。每一张图片都被标准化为尺寸为 28 乘 28 的黑白图片。如下图所示：

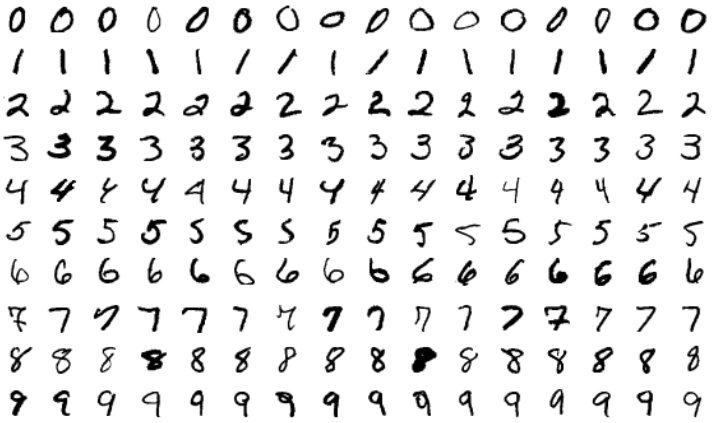

Figure 4.1　MNIST database

在此模型中，我们只用其中 5000 张来作为训练集与测试集。我们将每一张图片的像素值拉成一个向量，作为此样本的输入值，其长度为 $28 \times 28 = 756$。则每一个样本的预测变量（特征）为长度为 756 的向量，响应变量为 0 至 9 中的一位。我们将此数据投入先前讨论的模型中训练。

4.2　训练结果

首先，我们将数据集随机分为训练集与测试集。5000 张数据中，一共有 0 至 9 十类，每一类都有 500 张图片。我们取其中每一类的五分之一作为测试集，一共有 1000 张图片作为测试集，4000 张图片作为训练集。

我们使用训练集训练数据，测试集测试训练效果，在 GBM 使用 deviance loss

时，预测函数所谓分类器。分类器的精确度计算方式为测试集预测正确的响应变量在测试集中占的比例 在未调任何参数时，分类器得到的精确度如下：

Adaboost	0.626
GBM(ls loss)	0.348
GBM(lad loss)	0.389
GBM(huber loss)	0.334
GBM(deviance)	0.912
XGboost	0.921

其中 Adaboost 与 GBM 的基分类器数目均为 100，分类器为决策树。可以看出，GBM 运用 deviance 作为损失函数，即 logistic GBM，时的效果明显好于其他损失函数，也优于分类的 Adaboost。XGboost 的效果在所有方法中最优。

4.3 交叉验证

为了确定此结果是否具有代表性，还是在特殊情况下的结果，我们使用交叉验证，将数据集随机分为 5 份，每一次用其中一份作为验证集，剩下 4 份作为训练集，观察训练效果。

考虑到 GBM 中损失函数为 LS，LAD, Huber 损失时，基分类器为回归树，最终预测结果是回归预测的连续值，使用上述计算精确度的方法略有不妥当，我们使用决定系数 R^2 来刻画。回归器的预测结果如下：

GBM(ls loss)	GBM(lad loss)	GBM(huber loss)
0.726	0.665	0.713
0.736	0.678	0.735
0.731	0.670	0.712
0.742	0.688	0.739
0.724	0.654	0.713

由此结果看出，回归器的预测结果并不非常好，但也在接受范围内，与精确度的评判结果相符合。

分类器的交叉验证结果如下： 由此看出，第一次训练的结果具有一定代表性，交叉验证显示的精确度与之前的结果相符。XGBoost 仍然有最好的效果，而 GBM 稍次之，Adaboost 的结果并不理想。

Adaboost	GBM(deviance)	XGboost
0.578	0.913	0.928
0.507	0.919	0.914
0.494	0.895	0.913
0.385	0.917	0.921
0.490	0.917	0.918

4.4 参数调节

在这一章节，我们使用 Greedy search 的方法，找到对于每一个分类器最优的参数，如基分类器的个数，每一个决策树的深度，优化步长等等。将每一个参数带入分类器进行训练，选取结果最好的参数结果。为了方便比较，我们只选取分类器的结果进行比对，最终的使用结果如下：

Adaboost Adaboost 使用最优参数 (47 个基分类器，优化步长 0.2 等) 得到的最好预测结果为 69.6%. 其中，混淆矩阵的 heatmap 为：

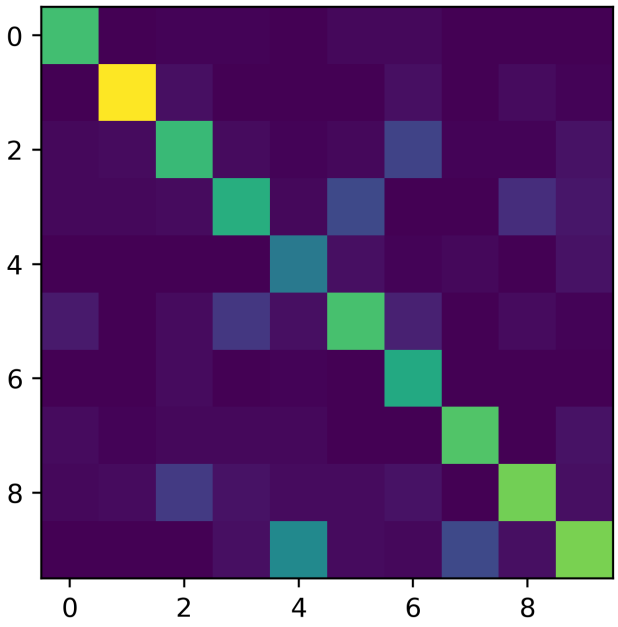

Figure 4.2 Adaboost heatmap

GBM with deviance　GBM 使用最优参数 (200 个基分类器，优化步长 0.2 等) 得到的最好预测结果为 93.2%. 其中，混淆矩阵的 heatmap 为：

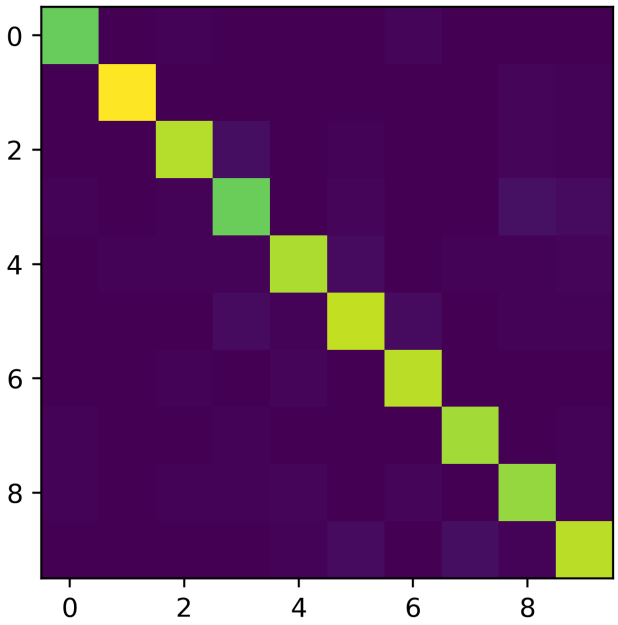

Figure 4.3　GBM heatmap

XGBoost　XGBoost 使用最优参数 (200 个基分类器，优化步长 0.25 等) 得到的最好预测结果为 94.7%. 其中，混淆矩阵的 heatmap 为：

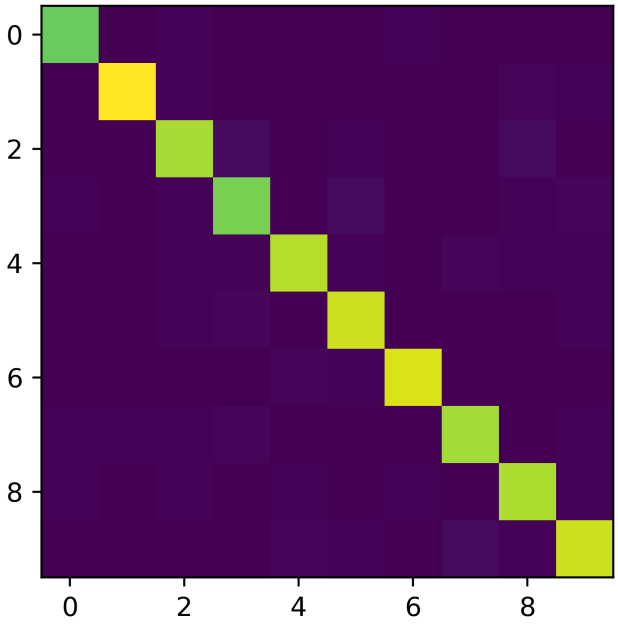

Figure 4.4　XGBoost heatmap

4.4.1　结论

由以上结果可知，Adaboost 的结果较好，在可接受范围。相比之下，GBM 与 XGBoost 有着更好的预测结果，也是它们现在被广泛使用的原因，当 GBM 的损失函数为 deviance 时，对于分类问题有着很高的预测精度。XGBoost 因使用二次梯度来优化函数吗，有着更高的收敛率。但实际收敛的时间并不占优，也许与计算二次梯度的计算复杂度有关。

第 5 章　结语

本文基于 boosting 的几种算法，深入分析了其提出的理论背景与思路历程，比较这几种算法理论背景与实际表现的异同。

由 Boosting 的理论特性与提出动机，将 boosting 归纳为一种非参学习方法。从统计的角度出发，得出 Adaboost 的实质是一种 stagewise forward 可加模型，其使用了指数损失来调优，具体的表现为调节样本权重。并且，我们推断得出 deviance，交叉熵损失和指数损失有着相同的结果，但在实际的优化中，由于优化函数不同，可能根据数值计算的差异（如有时出现的数值不稳定性），最终收敛到不同的结果。

从可加模型的角度探索，GBM 是一种非参中的函数估计，也即在函数空间上的一种优化。GBM 的计算核心为最速下降法，但与传统的梯度优化不同，其并没有将每一步的梯度作为方向，而是用弱分类器（决策树）去拟合每一步的梯度，根据可加模型的思想，将弱分类器加入到现在的估计函数中进行优化。其常用的损失函数为 LS 损失，绝对值损失，huber 损失和交叉熵损失。其中，用于分类问题的交叉熵损失有着对于分类问题有着最好的表现。这一点在我们的例子中也有证明。

XGBoost 在 GBM 的基础上进一步延申。它的计算核心为牛顿方法，所以 XGBoost 的优化有时也被称为 Newton Boosting。其用二次梯度来决定优化方向，使用二阶泰勒展开来拟合当前目标损失函数。使用这一思想，可以重新定义一种构建树的算法（与基尼系数，信息熵类似），此算法决定了决策树中怎样选择节点可以达到最大的准确率。

在实际数据算法比较中，与我们的理论结果相符，GBM 与 XGBoost 有着最好的预测准确率，也是现在被广泛使用的原因。

本文还有一些缺点与不足，如 GBM 基分类器为回归树时，GBM 回归器有着较好的性质，但在分类问题中却没有突出的表现，是否可以由某种方法将回归预测的离散结果更好的运用于离散响应变量预测上。以及 XGBoost 有着更好的收敛率，但实际操作中训练的时间较久，这种弊端是否可以避免，是否与其估计贪婪算法的运用有关。在进一步的研究中，我会对这些问题做进一步探究。

Bibliography

[1] Friedman, Jerome, et al. "Special Invited Paper. Additive Logistic Regression: A Statistical View of Boosting." The Annals of Statistics, vol. 28, no. 2, 2000, pp. 337–374. JSTOR, www.jstor.org/stable/2674028.

[2] Friedman, Jerome H. "Greedy function approximation: a gradient boosting machine." Annals of statistics (2001): 1189-1232.

[3] Efron, Bradley, et al. "Least angle regression." The Annals of statistics 32.2 (2004): 407-499.

[4] Friedman, Jerome, Trevor Hastie, and Robert Tibshirani. The elements of statistical learning. Vol. 1. No. 10. New York: Springer series in statistics, 2001.

[5] Chen, Tianqi, and Carlos Guestrin. "Xgboost: A scalable tree boosting system." Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. ACM, 2016.

[6] Nielsen, Didrik. Tree Boosting With XGBoost-Why Does XGBoost Win" Every" Machine Learning Competition?. MS thesis. NTNU, 2016.

致谢

感谢邓爱姣老师在毕业论文中给我的指导与鼓励!