北京师范大学珠海分校

本科生毕业论文

论文题目　Boosting集成学习算法研究

学　　院　应用数学学院

专　　业　数学与应用数学

学　　号　1717010106

学 生 姓 名　章俊鑫

指导教师姓名　李艳

指导教师单位　北师大珠海分校应用数学学院

2021年 03月20日

Boosting 集成学习算法研究

摘要

在机器学习这一领域，Boosting 集成算法一直都是随着时代的发展而发展，不论是在信息，医疗，通信，各行各业都有着急速的发展以及应用，本篇论文首先介绍了 Boosting 集成学习算法的发展历史以及国内研究现状通过对已有的 Boosting 集成学习算法的文献通读再研究，分别详细阐述了集成学习中最基础的 AdaBoost 算法原理，以及算法框架；在日常应用以及竞赛中常用的 GBDT 算法，并详细讲述负梯度拟合，损失函数优化等在 GBDT 算法中的使用及概念；以及着重研究陈天奇老师以 GBDT 算法为基础改进的 XGBoost 算法原理，框架以及损失函数的优化问题，叶子区域的最优解等，这三种算法的原理，以及算法的框架。并且以目前实用性、精确度、效率都较高的 XGBoost 算法做为应用实例讲述算法实际应用，如何调参等。

关键词：机器学习；集成学习；Boosting 集成算法

RESEARCH ON BOOSTING ENSEMBLE LEARNING ALGORITHM

ABSTRACT

In the field of machine learning, boosting ensemble learning algorithm has been developing with the development of the times. It has a rapid development and application in information, medical, communication and all walks of life. This paper first introduces the development history and domestic research status of boosting ensemble learning algorithm, This paper describes the basic AdaBoost algorithm principle and algorithm framework in ensemble learning; gbdt algorithm commonly used in daily application and competition, and describes the use and concept of negative gradient fitting and loss function optimization in gbdt algorithm in detail; and focuses on the principle, framework and loss function of xgboost algorithm improved by Chen Tianqi based on gbdt algorithm Optimization problem, the optimal solution of leaf region, the principle of these three algorithms, and the framework of the algorithm. And taking xgboost algorithm, which has high practicability, accuracy and efficiency, as an application example, it describes the practical application of the algorithm and how to adjust the parameters.

Keywords: machine learning; ensemble learning; boosting ensemble algorithm

目录

1 绪论

1.1 Boosting 集成学习算法发展历史

在人工智能这一领域，自 1956 年提出人工智能这一概念发展至今，机器学习在人工智能这一学科中一直都是发展最快的分支之一，同时也是在人工智能中最能够凸显机器能够具有独立自主学习的能力以及分析思考的智能。[1]伴随着软件、硬件以及需求的发展，尤其在现如今大数据时代的来临，机器学习这一学科的存在对于这一时代的意义显得尤为重要。而在机器学习领域中，自发展至今，如何提高模型训练的精度一直都是人们最为关心的部分。

人工智能中机器学习的研究者们一直都在致力于如何提高通过已知样本已经训练好的模型对新的测试样本得到一个尽可能高的精度估计，然而经过大量的算法研究者的努力以及付出。至今还是认为并没有哪一种算法能够非常好的构造一种模型使得测试精度大大提升，虽然优化后的模型算法精度确实在一定程度上能够提高，但是能够提高的上限非常局限，并且从多类问题中学习对于大多数分类器来说并不是一件容易的任务，因为存在多个问题[2]想要通过目前已有的算法构建出一个高精度模型估计依旧是非常困难的事情。

但是换个角度去思考，以现在的算法构建出多个精度略低的模型却是绰绰有余，如果能够集成多个精度略低的模型来得到一个集成后能够提升精度的模型，提高模型精度的想法在通过集成这一层面来实现，Boosting 算法应运而生。[3]

Boosting 算法是一种通过诸如集成模型和加权系数等算法提高给定学习算法准确性的方法。这一说法源自于 Valiant 在二十世纪初所提到的的 PAC 学习模型。[4]在当时 Valiant 首次提出了弱机器学习和强机器学习的基本概念，他在书中第一次给出了定义在机器学习中识别中准确率高于 50%。也就是说，如果存在一种机器学习算法并且它的准确率能够稍微高于人类行为层面上的随意猜测的统计，比如机器测算抛骰子落地后顶端是哪一面的准确率能够稍高于由人类随意猜测后统计的概论那么在人工智能这一领域中机器学习这一学科中统一认为这种类型的算法是弱机器学习算法反之称之为强机器学习算法。

而与此同时，Valiant 在这一基础上提出了一个更为大胆的想法，即弱机器

学习算法能否通过 PAC 学习模型中的算法实现弱学习算法的模型精度提升。在弱机器随机学习算法中，有些算法的模型准确度要比弱随机机器学习算法稍强一些。如果能够实现，那么就不必费尽心思去构造强机器学习算法，只需要通过集成模型来改进弱机器学习算法的模型精度即可达到最终目的。

1990 年，Schapire 最先构造出一种多项式级的算法，即最初的 Boosting 算法。这种算法可以将弱分类规则转化为强分类规则。[5]

1993 年，Drucker 和 Schapire 第一次以神经网络作为弱学习器，应用 Boosting 算法来解决实际的 ORC 问题。由于早期的 Boosting 算法在解决实际问题时要求事先知道弱学习算法学习正确率的下限，这一步本身就很难实现。[6]

到了二十世纪尾叶，首次提出并且构造了一个多项式级的算法，并且该问题已经被机器学习的研究者们进行了充分的实践证明。这是最早的正真意义上的 Boosting 集成学习算法。

而后又过了一段时间，德国的 Freund 对原始的 Boosting 集成学习算法进行了优化提出了一种更有效的 boosting 集成学习算法。二十世纪末，Boosting 算法再一次进行了算法上额优化以及改进，尤其是提出了 AdaBoost（自适应 boosting）这一划时代的加权算法。该集成学习算法的效能相比最原始的 Boosting 算法提高了一些，但最具意义的是该集成学习算法不需要弱机器学习具备任何先验知识，这使得 Boosting 算法的受众性更强，对于实际的问题更容易应用上去。

之后，随着投票权重这一想法对于 Boosting 这一集成学习的算法进行优化，推出了 AdaBoost M1 和 AdaBoost M2 等算法，使得 Boosting 集成学习算法受众更广，更实用，并且精度也随之上升，到目前为止，boosting 集成学习算法在机器学习领域备受关注。[7]

1.2 Boosting 集成学习算法研究意义

随着时代的发展，机器学习这一领域在信息，医疗，通信，各行各行都有着急速的发展以及使用，各个学科领域对于机器学习算法的需求日益增加，本篇论文通过对已有的 Boosting 集成学习算法的文献再研究希望从中整合或并找出可能被忽略的但是能够在一定程度上提高集成后模型精度的部分，也可在算法上实

现一定程度的优化，也可以通过叠加算法通过迭代使得模型精度上升，又或者在原有基础上实现一定程度的算法整合或是创新以实现最终集成模型精度的再提升，使得 Boosting 的集成学习算法在机器学习这一学科的应用上更上一层楼，用途更加广泛，让数据能够通过不同的聚类、分类、神经网络等等，不同机器学习算法的剖析方式来使得各行各业目前在模型精度的瓶颈能够有一个细微的提升，为社会为国家在信息技术，人工智能等领域做出更多的贡献，为人民的信息生活提供更加优质的服务以及技术。

1.3 Boosting 集成学习算法在国内外研究现状

1.3.1 近十年国内研究现状

2012 年，高敬阳研究团队对 AdaBoost 算法进行了一定程度的改进，AdaBoost 算法在目前还是处于比较主流的，这也是目前实践中最典型并且也具有一定价值的算法。但是 AdaBoost 算法存在一个误差样本恶性积累的缺点。伴随迭代次数的增长，误差样本的权重将趋于呈现指数增长，这一现象便导致算法随着迭代过程的累计会出现恶性积累，并且这一积累过程还会随着迭代过程持续。为了避免恶性积累以及导致的算法模型过度拟合，高敬阳对算法进行了改进。针对 AdaBoost 算法的样本分布权重更新过分强调困难样本，即过于针对难分类样本，高敬阳提出了基于争议度的权值修改算法 ERstd—AdaBoost 算法；而针对 AdaBoost 算法普遍存在的过拟合情况，高敬阳通过以样本分布 ABSD 的权值调整为基础的修正算法，同时高敬阳深入研究了反向权值分配策略的集成网络、个体分类器和网络差异度的泛化性能，提出了反向权值分配策略的改进算法 IB+。[8]

2014 年，卢婷针对组合分类器中的算法进行了一定程度的研究。卢婷认为，假定现使每个样本都有一个原始权重并以此权重表示样本能够被选入训练子集的基本概率。并且在迭代过程中，倘若这个样本在上一轮迭代被算法归为正确分类，那么这一次归类将会进入归类计数器并且提高这一样本的权重，否则，这一样本的权重将减少。这样，算法就会将重点放在困难分类样本上，卢婷通过对算法迭代过程添加样本权重提高这一方式，大幅提高了困难样本的分类精度。她使用算法对不平衡的数据集进行分类。而通过实验结果表明，通过 C4.5、朴素贝

叶斯、费希尔以及伪逆这几种组合分类器作为AdaBoost算法，结果显示可以提高少量类的分类精度和所有样本的分类性能，并且对高精度模型的获取也获得了有指导性的方向的结论。在此基础上卢婷对算法进行了改进。不是固定训练子集的大小，而是根据每个样本的权重和训练样本集的容量的乘积来确定每个样本被选入新训练子集的次数。[9]如此：

Ⅰ.可以使得训练子集包含所有的样本，并且不会存在关键数据信息的失去，还在一定程度上提高了提高了分类性能。

Ⅱ.在很大程度上避免了训练子集中一些相同类的样本数据特别多从而导致其他类样本少或无样本的情况，有效避免了过拟合和偏差问题。

2016年蔡小龙认为在Boosting算法中由于传统的学习算法多是基于ERM(经验风险最小化）的原则，而ERM原则是不适定的，会发生过拟合现象即用过于复杂的函数去巧合有限样本的情形，推广性较差。为了避免此类问题的发生，蔡小龙研巧的是正则化框架下的Boosting算法。针对独立同分布样本的情形，证明了Boosting算法是一致的。因为无论从理论上，还是实际应用中，独立同分布的条件或假设都是很强的，故我们又针对非独立同分布，即混合序列样本的情形，证明了基于混合序列的正则化Boosting算法是一致的。[10]

2019年，王超提出了基于分位数来分类逻辑回归的逻辑树模型这一思想。该模型通过更好地利用样本权重信息来进行加权样本的分类。随着王超研究的深入在分析现有Boosting算法的基础之上，提出了一种更平滑的损失函数，并且基于更加平滑的损失函数提出了两种新的Boosting算法。王超提出的Boosting算法有效地避免了模型存在过拟合问题，而因为迭代过程错误的样本的权重分布更新程度整体呈线性增长，这与指数形损失函数模型相比使得算法具有更好的健壮性。通过仿真数据和实际数据的实验结论均表明，该算法下的逻辑树模型能较好地适应10-50个协变量的分类问题，王超的AdaBoost.MH想法将这两种Boosting算法推广到多分类问题。[11]

在集成分类中，如何对基分类器实现动态更新和为基分类器分配合适的权值一直是研究的重点。[12]随后就是今年2020年11月份杜诗语团队等人针对以上两点，提出了BIE算法和BIWE算法。BIE算法能够根据新训练的基分类器的准确度，以此确定模型集成是否有必要更改性能较差的基分类器以及需要替换的基分

类器个数，从而实现对集成分类器的实时动态样本权值迭代更新。在此基础上，BIWE 算法更是在迭代基础上添加了一个加权函数，该函数可以针对具有不同参数属性的数据流以此获得此轮迭代下基分类器的能够获得的最佳权重，这一部分与 GBDT 以及 XGBoost 算法中心思想比较相似，并且以此提高集成分类器的整体模型精确度以及性能。通过杜诗语团队的实验研究结果表明，杜诗语团队的算法与常规算法相比，BIE 算法在精度相等或稍高的情况下，可以很大程度上减少生成树的分叉叶数、叶子节点数和树的纵深度；而与常规算法相比，BIWE 算法不仅具有更高的模型精度，同时也可以大大减小生成树的大小。

2 Boosting 中各类算法原理

2.1 AdaBoost 算法阐述

Boosting 算法中最基础并且最著名的属 AdaBoost 算法，为简单阐述原理现给定一个二分类训练数据集：$T=\{(x1,y1),(x2,y2)\ldots(xN,yN)\}$ 首先在选择好的训练集中先赋予初始的权重:

$$D_1=(w_{11},w_{12}\ldots w_{1i}\ldots,w_{1N}),w_{1i}=\frac{1}{N},i=1,2,\ldots,N$$

并且以这个权重开始第一次训练，得到模型 1:

$$G_m(x):\chi\to\{-1,+1\}$$，m 表示迭代的第几轮

然后根据模型 1 中的学习误差率来更新训练样本的权重，计算 Gm(x)在训练数据集上的分类误差率：

$$e_m=P(G_m(x_i)\neq y_i)=\sum_{i=1}^{N}w_{mi}I(G_m(x_i)\neq y_i)$$

通过上述式子 Gm（x）明显就是在训练数据集上的误差率 em 就是被 Gm（x）误分类样本的权值之和。接着使得模型 1 中的经过学习的训练集的误差率高的样本点的权重变高由 am 表示 Gm(x)在最终分类器中的重要程度：

$$a_m=\frac{1}{2}\log\frac{1-e_m}{e_m}$$

明显的当 em <= 1/2 时，am >= 0,并且 am 显然是随着 em 的增大而减小，这

说明分类误差越大基分类器在最终模型中的权重占比更小。那么在模型 2 中这些权重增高的样本对于模型所能得到的最终精度所占的比重更高，即样本分布权重更新：

$$D_{m+1} = (w_{m+1}, w_{m+1,2}, ...w_{m+1,i}..., w_{m+1,N})$$

$$w_{m+1,i} = \frac{w_{mi}}{Z_m} \exp(-a_m y_i G_m(x_i)), i = 1,2,..., N$$

其中 Zm 是规范化因子，使得 Dm+1 成为一个概率分布：

$$Z_m = \sum_{i=1}^{N} w_{mi} \exp(-a_m y_i G_m(x_i))$$

并且经过如此重复的迭代过程，即迭代 M 次，使得模型直接得到 M 个基学习器。最终将这 M 个基学习器通过集合策略进行最终的样本分布权重更新后重新整合，得到最终的强学习器，即最终模型：

$$G(x) = sign(f(x)) = sign\left(\sum_{m=1}^{M} a_m G_m(x)\right)$$

其中 f(x)为组合各个分类器：

$$f(x) = \sum_{m=1}^{M} a_m G_m(x)$$

算法流程原理如图 1（图 1 参照刘建平老师所做流程图制作）所示：

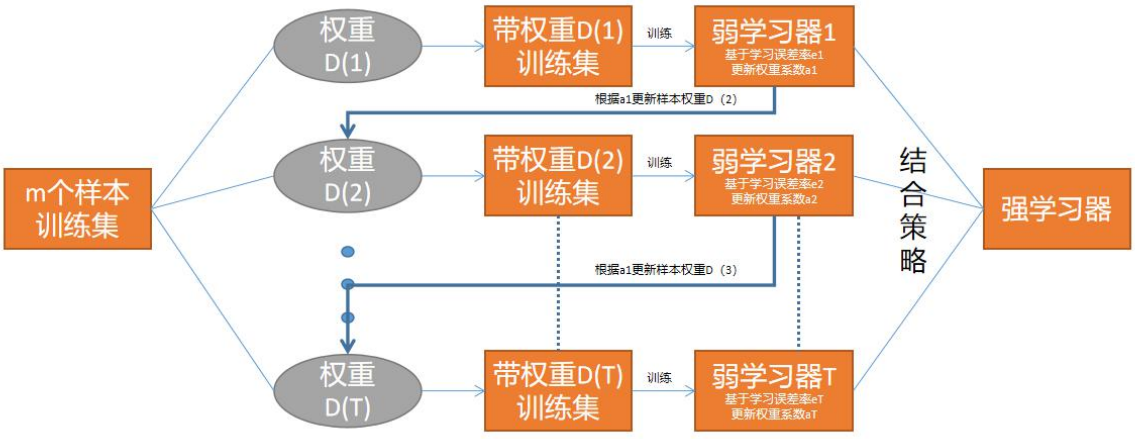

图 1　AdaBoost 算法流程图

对于日常学习及教学 AdaBoost 算法已经属于泛化性能比较优秀的算法，但是 AdaBoost 算法随着 M 值增大即迭代次数的上涨，AdaBoost 算法也存在明显的缺陷，由于初始训练集所赋予的 1/N 权值随着算法迭代会逐渐转化为非等的权

值，如果训练集中出现极难分类的样本，这会导致这一类样本的权重增涨随着迭代次数增多而显著升高，甚至会出现线性至指数增长的权重占比情况。这样就导致训练偏向这类困难样本，从而降低集成网络的泛化性能。[13]AdaBoost 算法提高被分类错误的样本权重[14]，本质上是需要使得单个基学习器之间存在差异最终整合后使得最终模型具备较高的精度。不过在基学习器之间差异上升的同时误差也随之增大，这就导致最终模型的准确性必然出现下降，反之同理最终模型准确性上升也必然会出现基学习器之间差异的下降。

所以，若是能够在迭代开始前就对样本分布权重更新的时候就定义一个约束条件来限制住模型在对困难样本出现权重占比过高的情况，理想状态下是应该在基学习器训练后差异度不变的情况下使得个体模型的准确性升高，并且使得最终的集成模型泛化性能上升。

2.2 Gradient Boosting(GBDT)算法阐述

2.2.1 GBDT 算法原理

目前在 Boosting 算法中更加常用的，在相关竞赛中出现比较多的就是 GBDT 算法，GBDT 全称 Gradient Boosting Decison Tree，这一算法与 AdaBoost 算法最大的差别就在于 AdaBoost 是通过上一轮的基学习器的误差率来进行样本分布权重更新，并且以同样的方式进行 M 次迭代；而 Gradient Boosting Decison Tree 虽然也是通过迭代，并且也同样使用了前向分布算法，而这里 Gradient Boosting Decison Tree 算法中有一条非常重要的限制，即基学习器被限定了只允许采用 CART 回归树模型，并且迭代的方式和 AdaBoost 也有较大的区别。

对于 AdaBoost 算法简单来说就是：

Boosting 框架+任意基学习器算法+指数损失函数

而 Gradient Boosting Decison Tree 简单来说就是：

Boosting 框架+CART 回归树模型+任意损失函数

详细来说，现假设有一训练集$\{x_i, y_i\}, i = 1,2,...,m,...,n$。先假设第 m-1 轮得到的强学习器为$F_{m-1}(x)$，而可以通过 Gradient Boosting Decison Tree 算法的递推公式:

$$F_m(x)=F_{m-1}(x)+\arg\min_{h\in H}\sum_{i=1}^{n}Loss(y_i,F_{m-1}(x_i)+h(x_i))$$

从而得到下一轮的弱基学习器 h(x)，其中 h(x)定义为函数空间 H 上的一个最小化损失函数，但显然希望通过实际算法最小化是比较难以实现的，但是换个角度若是通过高精确度地去拟合训练数据，即可以将损失函数 $\sum_{i=1}^{n}Loss(y_i,F_{m-1}(x_i)+h(x_i))$ 看作向量 $(F(x_1),F(x_2),...,F(x_n))$ 上的函数。这样在第 m-1 轮迭代之后，向量位于 $(F_{m-1}(x_1),F_{m-1}(x_2),...,F_{m-1}(x_n))$，为了能够更进一步地减小损失函数，就需要考虑向量移动的方向。根据 Freidman 的想法将向量移动方向定义为损失函数的负梯度方向，即对损失函数求偏导（一阶泰勒展开）：

$$V=-(\frac{\partial loss(y_1,F_{m-1}(x_1))}{\partial F_{m-1}(x_1)},\frac{\partial loss(y_2,F_{m-1}(x_2))}{\partial F_{m-1}(x_2)},...,\frac{\partial loss(y_n,F_{m-1}(x_n))}{\partial F_{m-1}(x_n)})$$

这样通过 $\left\{x_i,-\frac{\partial loss(y_i,F_{m-1}(x_i))}{\partial F_{m-1}(x_i)}\right\}_{i=1}^{n}$ 去进行 CART 树的训练，根据实验结果模型的损失函数明显是减小的。

在 GBDT 算法中损失函数的主体思想可以用一个较为容易理解的的例子去诠释，假定四十岁这一年龄为初始样本，首先用二十五岁去进行第一轮拟合，发现样本损失有十五岁，等到第二轮拟合再用九岁去拟合样本剩下的损失，结果发现样本差距还有六岁，第三轮我们用五岁拟合剩下的差距，如此一来样本拟合差距就只有一岁。如果我们的迭代轮数还没有完，可以继续迭代下面，每一轮迭代，拟合的岁数误差都会减小。[15]这一部分即是 GBDT 算法中损失函数的主体思想。

而经过 GBDT 算法所得到的最终的 CART 提升树模型显然是这 m 颗树的模型组合，如图 2 所示。

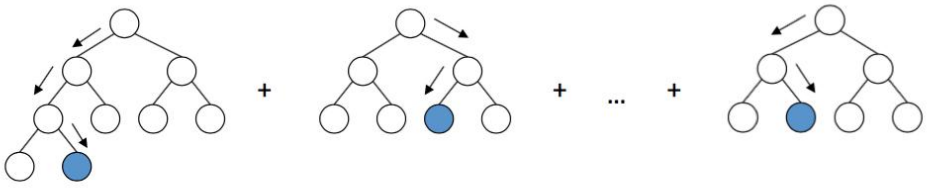

图 2　m 颗 CART 树组合

然而有一条非常重要的思想使得 GBDT 算法在实际应用中实现在每一轮迭代过程的损失最小，这一思想就是 Freidman 提出的负梯度拟合。

2.2.2 负梯度拟合

现假定在 GBDT 算法的迭代过程中，假定在第 t 轮迭代，上一轮的迭代所得到的较强的基学习器为$F_{t-1}(x)$，而损失函数为$Loss(y_i,F_{t-1}(x))$，此时第 t 轮迭代的目的是希望找到 CART 回归树模型的弱学习器$h_t(x)$，使得第 t 轮的损失量

$$Loss(y,F_t(x)+F_{t-1}(x))$$

为了使得上式值最低，Freidman 则提出通过负梯度来拟合本轮的损失量，进而逆向拟合出一个 CART 回归树。[16]

现将第 t 轮的第 i 个样本的损失函数对其求偏导，即负梯度（一阶泰勒展开）表示为：

$$r_{ti}=-\frac{\partial Loss(y_i,F_{t-1}(x_i))}{\partial F_{t-1}(x_i)}$$

因为$(x_i,r_{ti}),i=1,2,...,m$，由此可以拟合出 CART 回归树，进而得到第 t 轮的回归树，其对应的叶结点区域$R_{tj},j=1,2,...,J$，其中 J 为叶子节点个数。针对每一个叶子结点里的样本，我们求出使得损失函数值最小，也就是拟合叶子结点最好的输出值C_{tj}：

$$\omega_{tj}=\arg\min_c\sum_{x_i\in R_{tj}}Loss(y_i,F_{t-1}(x_i)+c)$$

这样就得到了第 t 轮的决策树拟合函数：

$$F_t(x)=\sum_{j=1}^{J}\omega_{tj}I(x\in R_{tj})$$

进而得出第 t 轮的最终基学习器，并且表达式如下：

$$F_t(x)=F_{t-1}(x)+\sum_{j=1}^{J}\omega_{tj}I(x\in R_{tj})$$

通过损失函数的负梯度来拟合，可以找到一种通用的拟合损失误差的办法，通过其损失函数的负梯度拟合这一办法无论是分类问题还是回归问题，都可以使用 GBDT 算法来解决这一类分类回归问题。而这个办法最大的区别仅仅在与损失

函数不同所导致的负梯度不同。并不会出现 AdaBoost 算法中出现难分类而导致误差率上升的情况。

2.2.3 GBDT 分类算法框架

GBDT 算法应用在回归问题中理解起来非常简易，因为样本输出的是连续的值而不同于分类算法样本输出的是离散的值，从而导致无法直接从输出的类别去拟合类别的误差，通常为了解决分类算法中样本的离散属性通过较常采用的是通过指数损失函数，此时 GBDT 变相的逆向退化为基础的 Adaboost 算法。而另一种方法是用类似于 logistic 回归的对数似然损失函数的方法。换个方式理解，即使用的是通过类别的预测概率值和样本的真实概率值的差来拟合损失。为了简化 GBDT 算法说明，这里仅仅探讨用对数似然损失函数的 GBDT 分类。而对于对数似然损失函数，又可以分为二元分类和多元分类的区别。

并且对于回归问题来说，损失函数通常选取平方差损失函数，这样残差刚好等值于预测值，实际值之间的差值。这样能够使得每一次迭代进行 CART 树的拟合中残差随着迭代次数的上升而下降。与 logistic 回归和 FM 模型相似，主要采用线性模型或带交叉项的非线性模型来拟合所谓的对数概率 $\ln\frac{p}{1-p}$。

为使得上述过程更直观易懂。现通过一个简单的二分类问题详细说明 GBDT 算法是如何生产 CART 树的。对于 GBDT 算法，通过负梯度拟合得到的 CART 树后再去拟合上述对数概率，而通过迭代拟合后得到的是大量 CART 回归树，其分类模型设为：

$$P(y=1|x)=\frac{1}{1+e^{-\sum_{m=0}^{M}h_m(x)}}$$

$h_m(x)$ 即为迭代学习后的 CART 树。现将单个样本 (x_i,y_i) 的损失函数表达为交叉熵：

$$Loss(x_i,y_i)=-y_i\log\hat{y}_i-(1-y_i)\log(1-\hat{y}_i)$$

那么当学习器迭代到第 t 轮，基学习器为 $F(x)=\sum_{m=0}^{t}h_m(x)$，将 $\hat{y}_i$ 回代得到损失函数：

$$Loss(x_i, y_i \mid F(x)) = y_i \log(1+e^{-F(x_i)}) + (1-y_i)[F(x_i) + \log(1+e^{-F(x_i)})]$$

然后再求损失函数的偏导得到负梯度（一阶泰勒展开）：

$$-\frac{\partial Loss}{\partial F(x)}|_{x_i, y_i} = y_i - \frac{1}{1+e^{-F(x_i)}} = y_i - \hat{y}_i$$

显然下一轮训练的 CART 树为：$\{x_i, y_i - \hat{y}_i\}_{i=1}^n$，而为了使得拟合出的残差为样本标签与预测概率的差，通过 GBDT 应用于二分类问题的算法，下面完整描述 GBDT 算法运用于二分类问题的伪代码：

①初始化基学习器；

现设定初始学习器 $F_0(x) = h_0(x) = \log\frac{p_1}{1-p_1}$，并且 p_1 为训练集中 y=1 的比例，而后通过先验信息初始化基学习器。

②For m=1,2,...,m,...,M do；

Ⅰ.计算损失函数的偏导得到负梯度 $y_i - \hat{y}_i$，并使用训练集 $\{(x_i, y_i - \hat{y}_i)\}_{i=1}^n$ 拟合 CART 树 $t_m(x)$，并且 $\hat{y}_i = \frac{1}{1+e^{-F_{m-1}(x)}}$。

Ⅱ. 进而得到第 m 轮的回归树，其对应的叶结点区域 $R_{tj}, j = 1,2,...,J$，其中 J 为叶子节点个数。

Ⅲ. 通过一维最小化损失函数得到 CART 树的最优权值：

$$\omega_{mj} = \arg\min_{\omega} \sum_{x_i \in R_{mj}} \text{Loss}(y_i, f_{m-1}(x_i) + c)$$

③考虑到函数的收缩，可以得到第 t 轮迭代后的学习器：

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

至此 GBDT 算法简单阐述完毕。

2.3 XGBoost 算法阐述

2.3.1 XGBoost 算法原理

XGBoost 算法是 16 年华盛顿大学的陈天奇老师基于 GBDT（Gradient Boosting Decison Tree）改进而来，XGBoost 属于算力上限较高的算法，相较于 GBDT 算法中的限制，即 GBDT 算法的基学习器被限定了只允许采用 CART 回归树模型，而 XGBoost 算法并没有这一限制，算法本身除了迭代过程的损失（损失函数）还在算法中添加了正则化部分并且在 XGBoost 算法的优化上对比 GBDT 算法只对误差函数求一阶偏导，即求函数的一阶泰勒展开（负梯度），XGBoost 对于损失函数选择进行二阶偏导，这一想法使得 XGBoost 算法本身具有更高的准确性。同时对于每一个弱学习器选择在 CART 树建立同时做对特征值有排序的多线程并行选择，换句话说即通过算法找出合适的 CART 子树分裂的一个特征向量。并且陈天奇对算法的健壮性进行了一定程度的提升，对训练数据进行预处理的时候对于缺失值的特征，通过穷举所有缺失值来决定 CART 树分叉是走向左子树或者右子树来决定缺失值的处理方式，并且算法本身添加了正则化部分，使得算法避免出现过拟合现象，大幅提高算法的泛化能力。

2.3.2 XGBoost 算法损失函数

因为 XGBoost 算法是基于 GBDT 算法改进而成的算法，所以下面通过简单阐述 GBDT 算法中损失函数的部分来引出 XGBoost 是如何对 GBDT 算法损失函数进行改进的。

①计算损失函数的偏导得到负梯度 $y_i - \hat{y}_i = -[\frac{\partial Loss(y_i, f(x_i))}{\partial f(x_i)}]_{f(x)=f_{t-1}(x)}$。

②进而得到第 m 轮的回归树，其对应的叶结点区域 $R_{tj}, j = 1,2,...,J$，其中 J 为叶子节点个数。

③通过一维最小化损失函数得到 CART 树的最优权值：

$$\omega_{tj} = \arg\min_c \sum_{x_i \in R_{tj}} \mathrm{Loss}(y_i, f_{t-1}(x_i) + c)$$

④更新强学习器，可以得到第 t 轮迭代后的学习器：

$$f_t(x) = f_{t-1}(x) + \sum_{j=1}^{J} \omega_{tj} I(x \in R_{tj})$$

其中 a 为学习率，并且随之一直迭代 M 次得到最终的强学习器模型。

上述第一步是为了得到负梯度即泰勒展开的一阶偏导；第二步是对第一步进行了优化求解，即基于样本标签与预测概率的差来拟合出一个 CART 树，这样得到 J 个叶子节点；然后第三步是对第二步进行优化求解，即在对第二步优化求解的结果上，对每个叶子节点再进行一次线性搜索，并且取每个叶子节点的最优取值；最终得到第 t 轮的强学习器。

GBDT 算法通过两步分别实现最优节点区域选择，然后再针对这些节点区域进行最优权值计算，而陈天奇在探究算法时更期望能够通过结合 GBDT 算法这两步来实现同步上述目的。首先对损失函数 $Loss(y, f_{t-1}(x) + h_t(x))$ 加入正则化项：

$$\Omega(h_t) = \gamma J + \frac{\lambda}{2} \sum_{j=1}^{J} \omega_{tj}^2$$

所以 XGBoost 损失函数表示为：

$$\sum_{i=1}^{m} Loss(y_i, f_{t-1}(x_i) + h_t(x_i)) + \gamma J + \frac{\lambda}{2} \sum_{j=1}^{J} \omega_{tj}^2$$

为了极小化损失函数，现对损失函数进行求二次偏导即二阶泰勒展开：

$$\sum_{i=1}^{m} (Loss(y_i, f_{t-1}(x_i)) + g_{ti} h_t(x_i) + \frac{1}{2} h_{ti} h_t^2(x_i)) + \gamma J + \frac{\lambda}{2} \sum_{j=1}^{J} \omega_{tj}^2$$

$$\text{其中} g_{ti} = \frac{\partial Loss(y_i, f_{t-1}(x_i))}{\partial f_{t-1}(x_i)}, h_{ti} = \frac{\partial^2 Loss(y_i, f_{t-1}(x_i))}{\partial f_{t-1}^2(x_i)}$$

因为 $Loss(y_i, f_{t-1}(x_i))$ 是常数，并不影响损失函数的极小化，并且不论迭代至第几轮决策树的第 J 个叶子节点始终为 ω_{tj}，所以损失函数可以极化简为：

$$\sum_{j=1}^{J} (\sum_{x_i \in R_{ij}} g_{ti} \omega_{tj} + \frac{1}{2} \sum_{x_i \in R_{ij}} h_{ti} \omega_{tj}^2) + \gamma J + \frac{\lambda}{2} \sum_{j=1}^{J} \omega_{tj}^2$$

$$= \sum_{j=1}^{J} [(\sum_{x_i \in R_{tj}} g_{ti}) \omega_{tj} + \frac{1}{2} (\sum_{x_i \in R_{tj}} h_{ti} + \lambda) \omega_{tj}^2] + \gamma J$$

$$\text{令} G_{tj} = \sum_{x \in R_{tj}} g_{ti}, H_{tj} = \sum_{x \in R_{tj}} h_{ti}$$

$$= \sum_{j=1}^{J} [(G_{tj} \omega_{tj} + \frac{1}{2} (H_{tj} + \lambda) \omega_{tj}^2] + \gamma J$$

2.3.3 XGBoost 损失函数的最优求解

在得到了极简化的损失函数后，现在的目标非常明确就是对损失函数进行优化求解，现优先求出每个叶子节点的最优解ω_{tj}，显然损失函数直接对ω_{tj}求导并且令导数为0，易得出ω_{tj}最优解的表达式：

$$\omega_{tj} = -\frac{G_{tj}}{H_{tj}+\lambda}$$

而接下来需要对如何选择特征以及特征值来使得损失函数最小，在GBDT算法中，是直接拟合出CART回归树，通过均方误差来对树节点进行分叉。而陈天奇老师在这一步骤不用均方误差，而是采用贪婪算法，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，算法得到的是在某种意义上的局部最优解。即每次的树节点分裂都期望最小化损失函数的误差。而在取ω_{tj}最优时损失函数应为：

$$-\frac{1}{2}\sum_{j=1}^{J}\frac{G_{tj}^2}{H_{tj}+\lambda}+\gamma J$$

而通过贪婪算法，在每一次树节点分裂过程都在最大程度上降低损失函数的损失，所以在决策树分裂时不再通过CART回归树的均方误差，现假设左右子树一阶，二阶导数和为G_L, H_L, G_R, H_R，则现使下式最大化并且在决策树分裂时取代CART回归树的均方误差：

$$-\frac{1}{2}\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}+\gamma J-(-\frac{1}{2}\frac{G_L^2}{H_L+\lambda}-\frac{1}{2}\frac{G_R^2}{H_R+\lambda}+\gamma(J+1))$$

$$\Rightarrow \max\frac{1}{2}\frac{G_L^2}{H_L+\lambda}+\frac{1}{2}\frac{G_R^2}{H_R+\lambda}-\frac{1}{2}\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}-\gamma$$

2.3.4 XGBoost 算法框架及优势

为简单阐述算法框架现假设有二分类训练集$\mathrm{I}=\{(x_1,y_1),(x_2,y_2),...,(x_m,y_m)\}$，设定迭代T轮，损失函数Loss，正则化系数$\gamma,\lambda$。

For t = 1,2,...,T do

①对第 i 个样本在第 t 轮的损失函数 Loss 求 $f_{t-1}(x_i)$ 的一阶导数 g_{ti} 和二阶导数 h_{ti}。并且令 $G_t=\sum_{i=1}^{m} g_{ti}; H_t=\sum_{i=1}^{m} h_{ti}$ 。

②对特征序号 $k=1,2,...,k$ 分别进行：

Ⅰ.令 $G_L=0, H_L=0$，即初始化。

Ⅱ.将样本随特征序号 k 按照由小到大排列，并且依特征序号 k 取出第 i 个样本，计算后放入左子树，并且左右子树的一阶二阶导数为：$\begin{aligned} G_L&=G_L+g_{ti}, G_R=G-G_L \\ H_L&=H_L+h_{ti}, H_R=H-H_L \end{aligned}$ 。

Ⅲ. 接着算法尝试更新最高分数：

$$Score=\max(score,\frac{1}{2}\frac{G_L^2}{H_L+\lambda}+\frac{1}{2}\frac{G_R^2}{H_R+\lambda}-\frac{1}{2}\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}-\gamma)$$

③基于最大 score 对应的划分特征和特征值分裂子树；

IF 最大 score==0;

⇒计算所有区域的 ω_{tj},

⇒得到弱学习器 $h_t(x)$,

⇒更新强学习器 $f_t(x)$,

⇒t+1,即进入下一轮迭代

ELSE

回到第②步在此尝试分裂决策树。

至此 XGBoost 算法具体框架阐述完毕，相较于 GBDT 算法Ⅰ. 通过损失函数对误差做二阶泰勒展开，更加精确 Ⅱ. 优先对特征值重要性进行排序，使得效率更高。

3 Boosting 算法应用实例

3.1 XGBoost 算法预测明天是否下雨

3.1.1 数据说明及预处理

数据来源：阿里_天池大数据众智平台_学习赛数据

竞赛题简述：现有气象站提供的每日降雨数据，需要根据历史降雨数据通过学习算法来预测明天是否会下雨。

数据简要说明见表 1 数据属性简要说明表：

特征名称	意义	数据类型
Date	日期	Str
Location	气象站地址	Str
MinTemp	最低温度	Int
MaxTemp	最高温度	Int
Rainfall	降雨量	Int
Evaporation	蒸发量	Int
Sunshine	日照时间	Int
WindGustDir	最强风向	Str
WindGustSpeed	最高风速	Int
WindDir9am	上午 9 点风向	Str
WindDir3pm	下午 3 点风向	Str
WindSpeed9am	上午 9 点风速	Int

WindSpeed3pm	下午 3 点风速	Int
Humidity9am	上午 9 点湿度	Int
Humidity3pm	下午 3 点湿度	Int
Pressure9am	上午 9 点大气压	Int
Pressure3pm	下午 3 点大气压	Int
Cloud9am	上午 9 点云浓度	Int
Cloud3pm	下午 3 点云浓度	Int
Temp9am	上午 9 点温度	Int
Temp3pm	下午 3 点温度	Int
RainToday	今天是否下雨	No, Yes
RainTomorrow	明天是否下雨	No, Yes

表 1　数据属性简要说明表

导入基本函数库后，读取要训练的数据，首先先对数据进行整体情况的查看，详情见图 3。

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 106644 entries, 0 to 106643
Data columns (total 23 columns):
Date             106644 non-null object
Location         106644 non-null object
MinTemp          106183 non-null float64
MaxTemp          106413 non-null float64
Rainfall         105610 non-null float64
Evaporation      60974 non-null float64
Sunshine         55718 non-null float64
WindGustDir      99660 non-null object
WindGustSpeed    99702 non-null float64
WindDir9am       99166 non-null object
WindDir3pm       103788 non-null object
WindSpeed9am     105643 non-null float64
WindSpeed3pm     104653 non-null float64
Humidity9am      105327 non-null float64
Humidity3pm      103932 non-null float64
Pressure9am      96107 non-null float64
Pressure3pm      96123 non-null float64
Cloud9am         66303 non-null float64
Cloud3pm         63691 non-null float64
Temp9am          105983 non-null float64
Temp3pm          104599 non-null float64
RainToday        105610 non-null object
RainTomorrow     106644 non-null object
dtypes: float64(16), object(7)
memory usage: 18.7+ MB
```

图 3　数据信息

再对头部尾部数据进行查看，通过观测数据的整体情况发现数据集中一些属性存在 NaN 值即缺失值，原因大概率是在数据采集或者数据在服务器进行预处理的时候所造成的一些数据缺失。为简化实验目的现将缺失值以“-1”为值进行填补。

接下来通过 value_counts 函数查看训练集标签的占数情况如图 4 所示：

```
No      82786
Yes     23858
Name: RainTomorrow, dtype: int64
```

图 4　数据标签占数情况

通过上图发现数据集中的“明天不下雨”样本数量大约是“明天下雨”样本数量的三到四倍，样本数据处于分类不平衡状态，但是这里并不影响 XGBoost 算法进行模型训练，故不作特殊处理，

接下来由于 XGBoost 算法是不能够处理 Str 数据，所以需要对离散型数据进行编码处理，这里为了简化预处理过程将采取对所有的相同类别的特征编码成同一个值，编码见图 5。

In [15]:
```
## 把所有的相同类别的特征编码为同一个值
def get_mapfunction(x):
    mapp = dict(zip(x.unique().tolist(),
         range(len(x.unique().tolist()))))
    def mapfunction(y):
        if y in mapp:
            return mapp[y]
        else:
            return -1
    return mapfunction
for i in category_features:
    data[i] = data[i].apply(get_mapfunction(data[i]))
```

In [16]:
```
## 编码后的字符串特征变成了数字

data['Location'].unique()
```

Out[16]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48],
      dtype=int64)
```

图 5　离散型数据编码

至此数据预处理完成。

3.1.2 数据属性与标签相关性可视化描述

为了确定数据集中是否存在与模型训练无关的属性，故需要对数据进行一定的模型训练相关性可视化分析。

首先选取数据集中意义与标签“明天是否下雨”看上去关系不大的降雨量 Rainfall，蒸发量 Evaporation，日照时间 Sunshine 与标签作散点可视化，如图 6 所示。

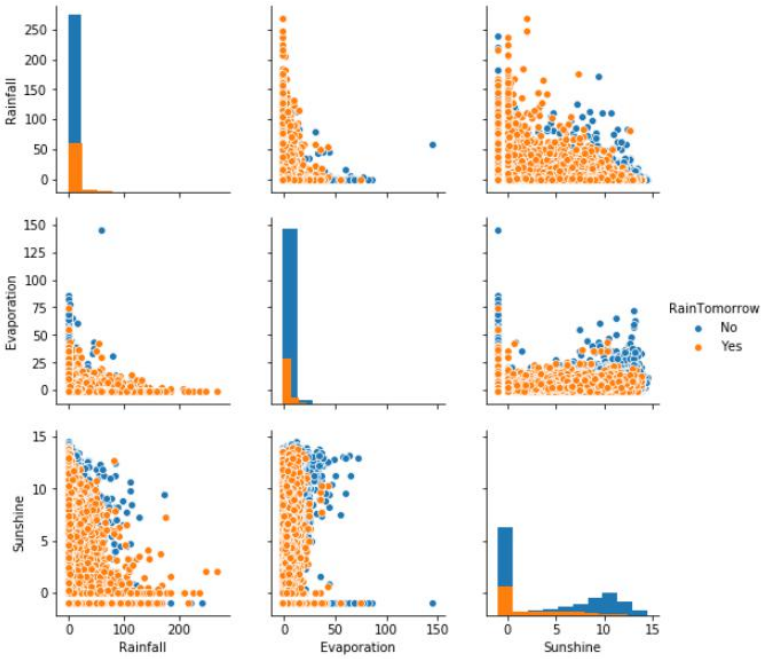

图 6　降雨量、蒸发量、日照时间与标签散点可视图

通过在平面散点图下不同的特征组合对于“明天是否下雨”的散点分布观察，相对其他两种属性日照时间 Sunshine 与其他特征的组合的散点图更具有区分度。

接着我们再通过基于随机森林的算法来度量每个变量对于标签“明天是否下雨”的重要程度，对变量重要性画图，并依次降序排列，但是由于数据量较大，这里只取前 1/10 的数据作为判断依据，见图 7。

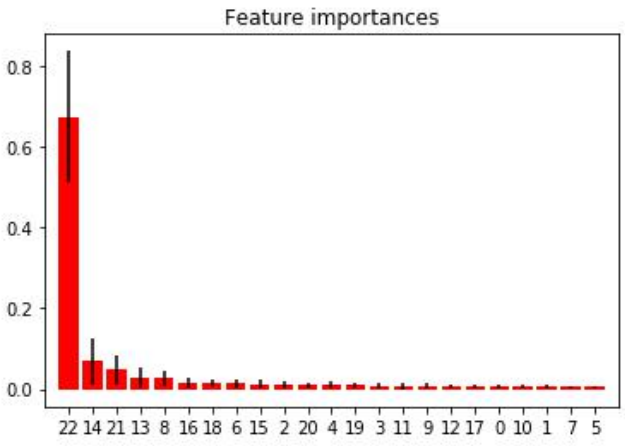

图 7　各属性对于标签分类重要程度（2）

通过观察上图发现“今天是否下雨”这一属性对标签“明天是否下雨”的分类有极高的相关性，但是图中单一属性存在如此高的相关性从而导致其他属性相关性无法判断这显然是不合理的，所以采取箱线图来逐一属性判断与标签之间的相关性，但由于属性较多这里只截取部分，如图 8。

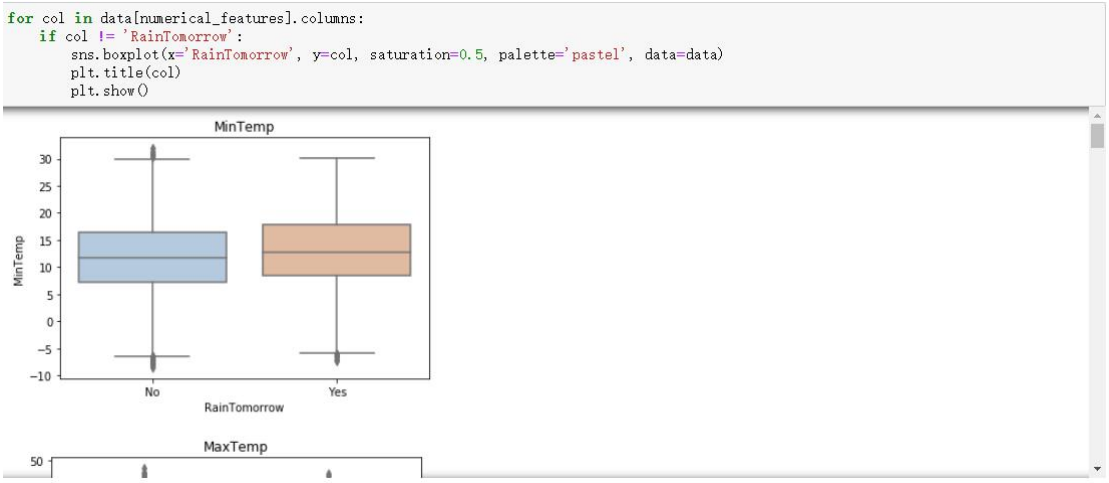

图 8　全属性与标签相关性箱线图（部分）

利用箱型图我们也可以得到不同类别在不同特征上的分布差异情况。我们可以发现 Sunshine,Humidity3pm,Cloud9am,Cloud3pm 的区分能力较强，但是其他属性也具有一定分类重要性，故全部属性都具有模型训练价值。

3.1.3 XGBoost 算法初训练与精度

现已数据集的 80%为训练集 20%为测试集划分数据，分好训练集以及测试集，

通过直接调用 XGBoost 算法获得训练模型，接着以模型精度，混淆矩阵，以及通过热力图对训练结果进行可视化，分析训练结果，见图 9。

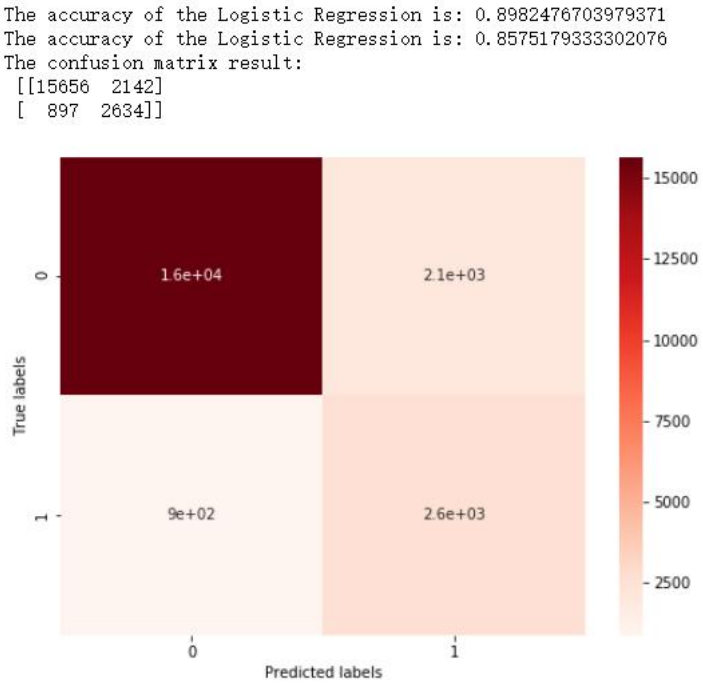

图 9　模型精度，混淆矩阵，模型热力图

通过上图得出，XGBoost 算法在未调参初次训练下算法准确率已经接近 90%，并且通过混淆矩阵数据表明共有 15656 + 2142 个样本预测正确，2634 + 897 个样本预测错误。这里已经在一定程度上说明 XGBoost 算法本身的高精度，高时效能力，而 XGBoost 的特征选择属于特征选择中的嵌入式方法，在 XGboost 中可以用 feature_importances_去查看属性的重要度，这样相较于上文提到的随机森林算法分析属性对于标签的重要程度效果明显，详情见图 10。

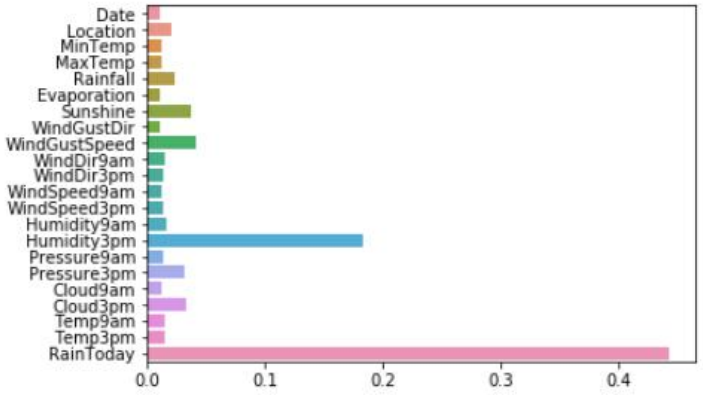

图 10　属性相较于标签重要程度（2）

3.1.4 XGBoost 算法调参及二次训练比较

在本文描述过陈天奇老师在 XGBoost 算法通过贪婪算法在每一次树节点分裂过程都在最大程度上降低损失函数的损失，而进行调节模型参数的方法有贪婪算法、均差调参、网格调参、贝叶斯调参等等。本实验通过网格调参，即在所有候选的参数选择中，通过循环遍历，尝试每一种可能性，表现最好的参数就是最终的结果，选择网格调参除了为算法调参，其次目的是为了通过穷举搜索来验证算法本身是否存在泛化能力差或者出现过拟合现象，具体见图 11。

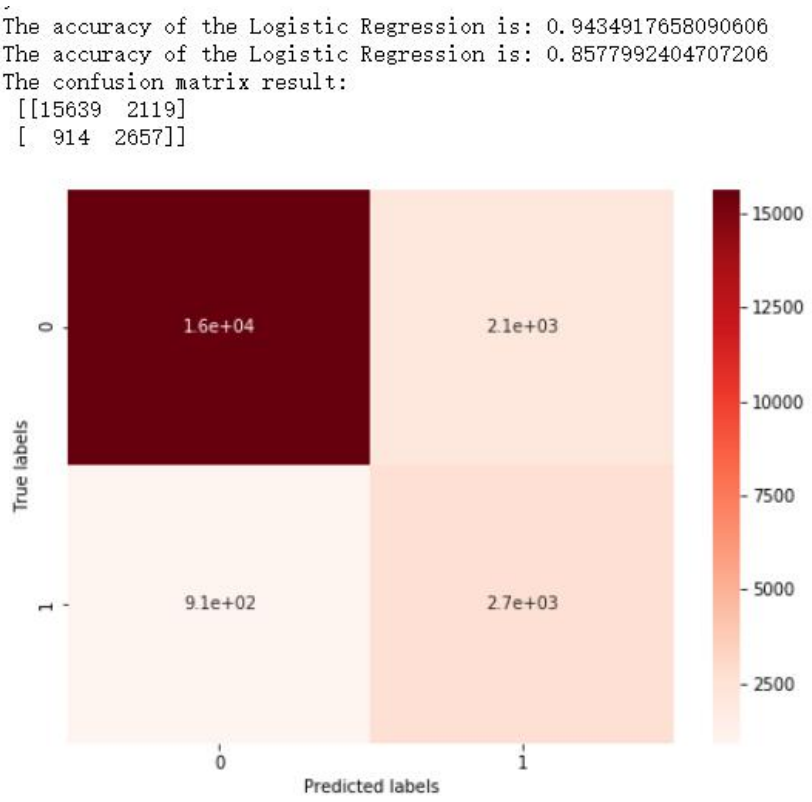

图 11　调参后算法精度，混淆矩阵，模型热力图

经过调参后，算法精度从原本初次训练的 89.8%提升至 94.3%，混淆矩阵数据也表明经过调参后相较于初次训练的 15656 + 2142 个样本预测正确，2634 + 897 个样本预测错误，调参后共有 15639 + 2119 个样本预测正确，2657 + 914 个样本预测错误。精度上升，而数值下降结合本文对属性与标签之间的相关度分析代表调参后的平均是这一结果，高的只是个例。而实验经调参后的准确度已经十分可观。XGBoost 算法实验实例至此结束。

参考文献

[1] 周志华. 机器学习[M].北京：清华大学出版社,2016.

[2] Mehmood Zafar and Asghar Sohail. Customizing SVM as a base learner with AdaBoost ensemble to learn from multi-class problems: A hybrid approach AdaBoost-MSVM[J]. Knowledge-Based Systems, 2021, 217

[3] 于玲,吴铁军. 集成学习:Boosting 算法综述[J].模式识别与人工智能,2004,17(01):52-59.

[4] Valiant L.G. A Theory of the learnable[P].Communications of the ACM, 1984,27(11):1134-1142.

[5] Schapire R.E. The Strength of Weak Learnability[J].Machine Learning,1990,5(2):197-227

[6] Drucker H,Schapire R.E,Simard P. Boosting Performance in Neural Networks[J]. International Journal of Pattern Recognition and Artificial Intelligence,1993,7(4):705-719

[7] 董乐红,耿国华,高原. Boosting 算法综述[J].计算机应用与软件,2006(08):27-29.

[8] 高敬阳. 神经网络集成 BOOSTING 类算法研究[D].北京化工大学,2012.

[9] 卢婷. 基于 AdaBoost 的分类器学习算法比较研究[D].华东理工大学,2014.

[10] 蔡小龙. 正则化 Boosting 算法的一致性[D].湖北大学,2016.

[11] 王超. 加权样本分类算法设计和基于加法逻辑回归模型的 Boosting 算法设计[D].华中师范大学,2019.

[12] 杜诗语,韩萌,申明尧,张春砚,孙蕊.基于 Boosting 的迭代加权集成分类算法[J/OL].计算机应用研究:1-7[2020-12-13].https://doi.org/10.19734/j.issn.1001-3695.2020.04.0101.

[13] Torres-Sospedra J , Hernandez-Espinosa C , Fernandez-Redondo M . Mixing Aveboost and Conserboost to Improve Boosting Methods[C].Proceeding of International Joint Conference on Neural Networks. IEEE, 2007.

[14] Roli F , Raudys S , Marcialis G L . An Experimental Comparison of Fixed and Trained Fusion Rules for Crisp Classifier Outputs[C].Multiple Classifier Systems, Third International Workshop, MCS 2002, Cagliari, Italy, June 24-26, 2002, Proceedings. 2002.

[15] arkirameiao. 带你搞懂算法原理[EB/OL].https://blog.csdn.net/akirameiao/article/details/80009155,2018-04-19.

[16] 李航. 统计学习方法—2 版[M].北京：清华大学出版社，2019.