梯度提升树(GBDT)完整教程 V1.0 发布

决策树 梯度提升 算法 Python 机器学习

作者：Freemanzxp (中科大 在读硕士)
编辑：榛果(Python与算法社区 编辑)

§

写在前面： 去年学习GBDT之初，为了加强对算法的理解，整理了一篇笔记形式的文章，发出去之后发现阅读量越来越多，渐渐也有了评论，评论中大多指出来了笔者理解或者编辑的错误，故重新编辑一版文章，内容更加翔实，并且在GitHub上实现了和本文一致的GBDT简易版（包括回归、二分类、多分类以及可视化），供大家交流探讨。感谢各位的点赞和评论，希望继续指出错误~

Github：https://github.com/Freemanzxp/GBDT_Simple_Tutorial

§

0. 简介

GBDT 的全称是 Gradient Boosting Decision Tree，梯度提升树，在传统机器学习算法中，GBDT算的上TOP3的算法。想要理解GBDT的真正意义，那就必须理解GBDT中的Gradient Boosting 和Decision Tree分别是什么？

1. Decision Tree：CART回归树

首先，GBDT使用的决策树是CART回归树，无论是处理回归问题还是二分类以及多分类，GBDT使用的决策树通通都是都是CART回归树。为什么不用CART分类树呢？因为GBDT每次迭代要拟合的是**梯度值**，是连续值所以要用回归树。

对于回归树算法来说最重要的是寻找最佳的划分点，那么回归树中的可划分点包含了所有特征的所有可取的值。在分类树中最佳划分点的判别标准是熵或者基尼系数，都是用纯度来衡量的，但是在回归树中的样本标签是连续数值，所以再使用熵之类的指标不再合适，取而代之的是平方误差，它能很好的评判拟合程度。

§

回归树生成算法：

输入：训练数据集D:

输出：回归树$f(x)$.

在训练数据集所在的输入空间中，递归的将每个区域划分为两个子区域并决定每个子区域上的输出值，构建二叉决策树：

（1）选择最优切分变量j与切分点s，求解

$$\min_{j,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right]$$

遍历变量j，对固定的切分变量j扫描切分点s，选择使得上式达到最小值的对(j,s).

（2）用选定的对(j,s)划分区域并决定相应的输出值：

$$R_1(j,s) = x|x^{(j)} \leq s, R_2(j,s) = x|x^{(j)} > s$$
$$\hat{c_m} = \frac{1}{N} \sum_{x_1 \in R_m(j,s)} y_i, x \in R_m, m = 1,2$$

（3）继续对两个子区域调用步骤（1）和（2），直至满足停止条件。
（4）将输入空间划分为M个区域 $R_1, R_2, \ldots R_M$，生成决策树：

$$f(x) = \sum_{m=1}^{M} \hat{c}_m I(x \in R_m)$$

---

2. Gradient Boosting：拟合负梯度

梯度提升树（Grandient Boosting）是提升树（Boosting Tree）的一种改进算法，所以在讲梯度提升树之前先来说一下提升树。

先来个通俗理解：假如有个人30岁，我们首先用20岁去拟合，发现损失有10岁，这时我们用6岁去拟合剩下的损失，发现差距还有4岁，第三轮我们用3岁拟合剩下的差距，差距就只有一岁了。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。最后将每次拟合的岁数加起来便是模型输出的结果。

提升树算法：
（1）初始化$f_0(x) = 0$
（2）对$m = 1,2,\ldots,M$
（a）计算残差

$$r_{mi} = y_i - f_{m-1}(x), i = 1,2,\ldots,N$$

（b）拟合残差r_{mi}学习一个回归树，得到$h_m(x)$
（c）更新$f_m(x) = f_{m-1} + h_m(x)$
（3）得到回归问题提升树

$$f_M(x) = \sum_{m=1}^{M} h_m(x)$$

上面伪代码中的**残差**是什么？
在提升树算法中，假设我们前一轮迭代得到的强学习器是

$$f_{t-1}(x)$$

损失函数是

$$L(y, f_{t-1}(x))$$

我们本轮迭代的目标是找到一个弱学习器

$$h_t(x)$$

最小化让本轮的损失

$$L(y, f_t(x)) = L(y, f_{t-1}(x) + h_t(x))$$

当采用平方损失函数时

$$\begin{aligned} & L(y, f_{t-1}(x) + h_t(x)) \\ &= (y - f_{t-1}(x) - h_t(x))^2 \\ &= (r - h_t(x))^2 \end{aligned}$$

这里，

$$r = y - f_{t-1}(x)$$

是当前模型拟合数据的残差（residual）所以，对于提升树来说只需要简单地拟合当前模型的残

差。

回到我们上面讲的那个通俗易懂的例子中，第一次迭代的残差是10岁，第二 次残差4岁.....

当损失函数是平方损失和指数损失函数时，梯度提升树每一步优化是很简单的，但是对于一般损失函数而言，往往每一步优化起来不那么容易，针对这一问题，Friedman 提出了梯度提升树算法，这是利用最速下降的近似方法，**其关键是利用损失函数的负梯度作为提升树算法中的残差的近似值。**

那么负梯度长什么样呢?

第 t 轮的第 i 个样本的损失函数的负梯度为:

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

此时不同的损失函数将会得到不同的负梯度，如果选择平方损失

$$L(y, f(x_i)) = \frac{1}{2}(y - f(x_i))^2$$

负梯度为

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

此时我们发现GBDT的**负梯度就是残差**，所以说对于回归问题，我们要拟合的就是残差。

那么对于分类问题呢？二分类和多分类的损失函数都是$logloss$，**本文以回归问题为例进行讲解**。

---

3. GBDT算法原理

上面两节分别将Decision Tree和Gradient Boosting介绍完了，下面将这两部分组合在一起就是我们的GBDT了。

GBDT算法：

（1）初始化弱学习器

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

（2）对$m = 1, 2, \ldots, M$有:

（a）对每个样本$i = 1, 2, \ldots, N$，计算负梯度，即残差

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

（b）将上步得到的残差作为样本新的真实值，并将数据$(x_i, r_{im}),\ i = 1, 2, .. N$作为下棵树的训练数据，得到一颗新的回归树$f_m(x)$其对应的叶子节点区域为$R_{jm}, j = 1, 2, \ldots, J$。其中$J$为回归树 t 的叶子节点的个数。

（c）对叶子区域$j = 1, 2, .. J$计算最佳拟合值

$$\Upsilon_{jm} = \underbrace{arg\ min}_{\Upsilon} \sum_{x_i \in R_{jm}} L(y_i, f_{m-1}(x_i) + \Upsilon)$$

（d）更新强学习器

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

（3）得到最终学习器

$$f(x) = f_M(x) = f_0(x) + \sum_{m=1}^{M} \sum_{j=1}^{J} \Upsilon_{jm} I(x \in R_{jm})$$

---

4. 实例详解

本人用python以及pandas库实现GBDT的简易版本，在下面的例子中用到的数据都在github可以找到，大家可以结合代码和下面的例子进行理解，欢迎star~

Github：https://github.com/Freemanzxp/GBDT_Simple_Tutorial

数据介绍：

如下表所示：一组数据，特征为年龄、体重，身高为标签值。共有5条数据，前四条为训练样本，最后一条为要预测的样本。

编号	年龄(岁)	体重（kg）	身高(m)(标签值)
0	5	20	1.1
1	7	30	1.3
2	21	70	1.7
3	30	60	1.8
4(要预测的)	25	65	？

训练阶段：

参数设置：

- 学习率：learning_rate=0.1
- 迭代次数：n_trees=5
- 树的深度：max_depth=3

1.初始化弱学习器：

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

损失函数为平方损失，因为平方损失函数是一个凸函数，直接求导，导数等于零，得到c。

$$\sum_{i=1}^{N} \frac{\partial L(y_i, c))}{\partial c} = \sum_{i=1}^{N} \frac{\partial (\frac{1}{2}(y_i - c)^2)}{\partial c} = \sum_{i=1}^{N} c - y_i$$

令导数等于0

$$\sum_{i=1}^{N} c - y_i = 0$$

$$c = (\sum_{i=1}^{N} y_i)/N$$

所以初始化时，c取值为所有训练样本标签值的均值。$c = (1.1 + 1.3 + 1.7 + 1.8)/4 = 1.475$，此时得到初始学习器$f_0(x)$

$$f_0(x) = c = 1.475$$

2.对迭代轮数m=1，2,...,M：

由于我们设置了迭代次数：n_trees=5，这里的$M = 5$。

计算负梯度，根据上文损失函数为平方损失时，负梯度就是残差残差，再直白一点就是y与上一

轮得到的学习器f_{m-1}的差值

$$r_{i1} = -\left[\frac{\partial L(y_i, f(x_i)))}{\partial f(x_i)}\right]_{f(x)=f_0(x)}$$

残差在下表列出：

编号	真实值	f_0	残差
0	1.1	1.475	-0.375
1	1.3	1.475	-0.175
2	1.7	1.475	0.225
3	1.8	1.475	0.325

此时将残差作为样本的真实值来训练弱学习器$f_1(x)$，即下表数据

编号	年龄(岁)	体重（kg）	标签值
0	5	20	-0.375
1	7	30	-0.175
2	21	70	0.225
3	30	60	0.325

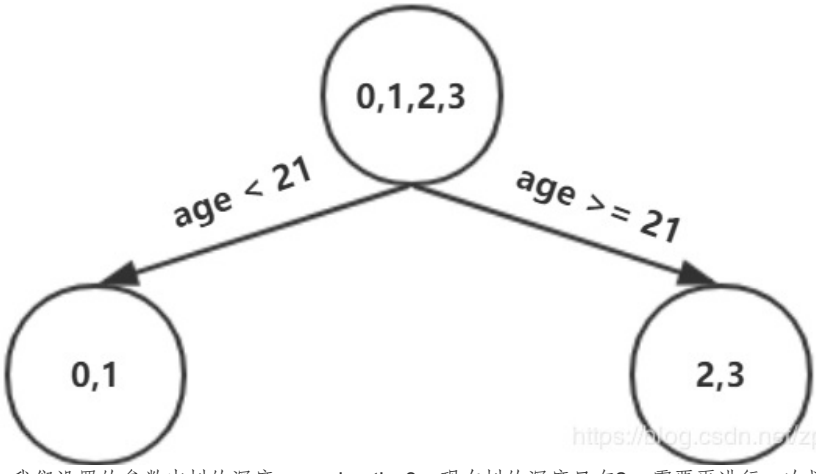
接着，寻找回归树的最佳划分节点，遍历每个特征的每个可能取值。从年龄特征的5开始，到体重特征的70结束，分别计算分裂后两组数据的平方损失（Square Error），SE_l左节点平方损失，SE_r右节点平方损失，找到使平方损失和$SE_{sum} = SE_l + SE_r$最小的那个划分节点，即为最佳划分节点。

例如：以年龄7为划分节点，将小于7的样本划分为到左节点，大于等于7的样本划分为右节点。左节点包括x_0，右节点包括样本x_1, x_2, x_3，$SE_l = 0, SE_r = 0.047, SE_{sum} = 0.047$，所有可能划分情况如下表所示：

划分点	小于划分点的样本	大于等于划分点的样本	SE_l	SE_r	SE_{sum}
年龄5	/	0，1，2，3	0	0.327	**0.327**
年龄7	0	1，2，3	0	0.140	**0.140**
年龄21	0，1	2，3	0.020	0.005	**0.025**
年龄30	0，1，2	3	0.187	0	**0.187**
体重20	/	0，1，2，3	0	0.327	**0.327**
体重30	0	1，2，3	0	0.140	**0.140**
体重60	0，1	2，3	0.020	0.005	**0.025**
体重70	0，1，3	2	0.260	0	**0.260**

以上划分点是的总平方损失最小为**0.025**有两个划分点：年龄21和体重60，所以随机选一个作为划分点，这里我们选 **年龄21**

现在我们的第一棵树长这个样子：



我们设置的参数中树的深度max_depth=3，现在树的深度只有2，需要再进行一次划分，这次划分要对左右两个节点分别进行划分：

---------- § ----------

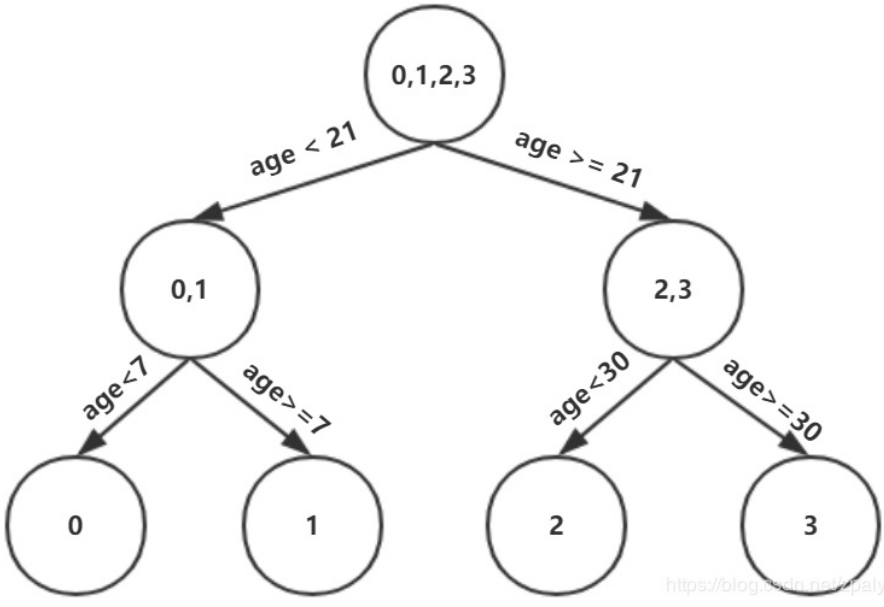
对于**左节点**，只含有0,1两个样本，根据下表我们选择**年龄7**划分

划分点	小于划分点的样本	大于等于划分点的样本	SE_l	SE_r	SE_{sum}
年龄5	/	0，1	0	0.020	**0.020**
年龄7	0	1	0	0	**0**
体重20	/	0，1	0	0.020	**0.020**
体重30	0	1	0	0	**0**

对于**右节点**，只含有2,3两个样本，根据下表我们选择**年龄30**划分（也可以选**体重70**）

划分点	小于划分点的样本	大于等于划分点的样本	SE_l	SE_r	SE_{sum}
年龄21	/	2，3	0	0.005	**0.005**
年龄30	2	3	0	0	**0**
体重60	/	2，3	0	0.005	**0.005**
体重70	3	2	0	0	**0**

现在我们的第一棵树长这个样子：



此时我们的树深度满足了设置，还需要做一件事情，给这每个叶子节点分别赋一个参数Υ，来拟合残差。

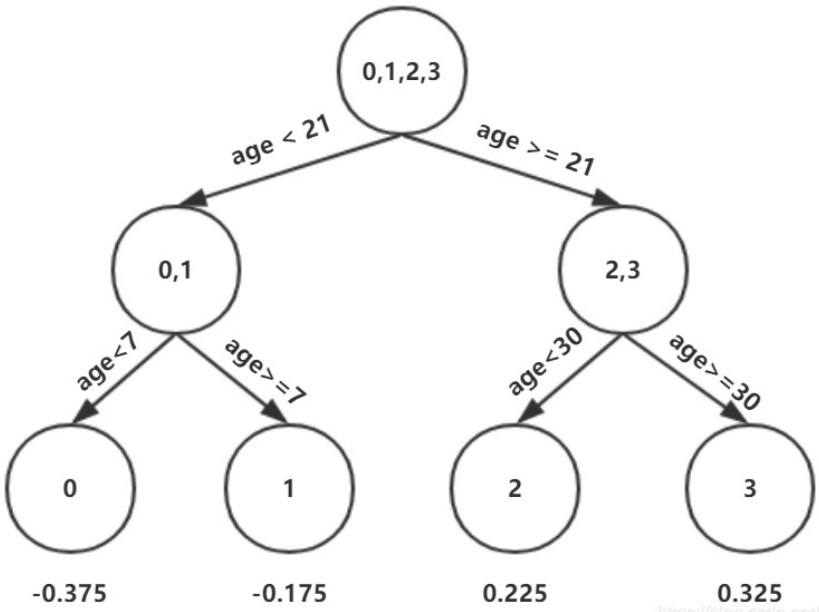
$$\Upsilon_{j1} = \underbrace{arg\,min}_{\Upsilon} \sum_{x_i \in R_{j1}} L(y_i, f_0(x_i) + \Upsilon)$$

这里其实和上面初始化学习器是一个道理，平方损失，求导，令导数等于零，化简之后得到每个叶子节点的参数Υ，其实就是标签值的均值。这个地方的标签值不是原始的 y，而是本轮要拟合的标残差 $y - f_0(x)$.

根据上述划分结果，为了方便表示，规定从左到右为第$1, 2, 3, 4$个叶子结点

$$(x_0 \in R_{11}),\ \Upsilon_{11} = -0.375$$
$$(x_1 \in R_{21}),\ \Upsilon_{21} = -0.175$$
$$(x_2 \in R_{31}),\ \Upsilon_{31} = 0.225$$
$$(x_3 \in R_{41}),\ \Upsilon_{41} = 0.325$$

此时的树长这个样子：



此时可更新强学习器，需要用到参数学习率：learning_rate=0.1，用lr表示。

$$f_1(x) = f_0(x) + lr * \sum_{j=1}^{4} \Upsilon_{j1} I(x \in R_{j1})$$

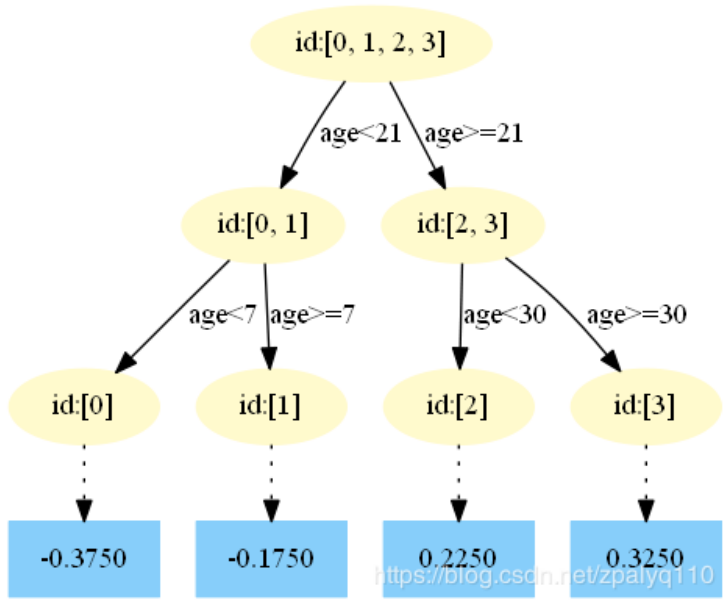
为什么要用学习率呢？这是**Shrinkage**的思想，如果每次都全部加上（学习率为1）很容易一步学到位导致过拟合。

重复此步骤，直到$m > 5$ 结束，最后生成5棵树。

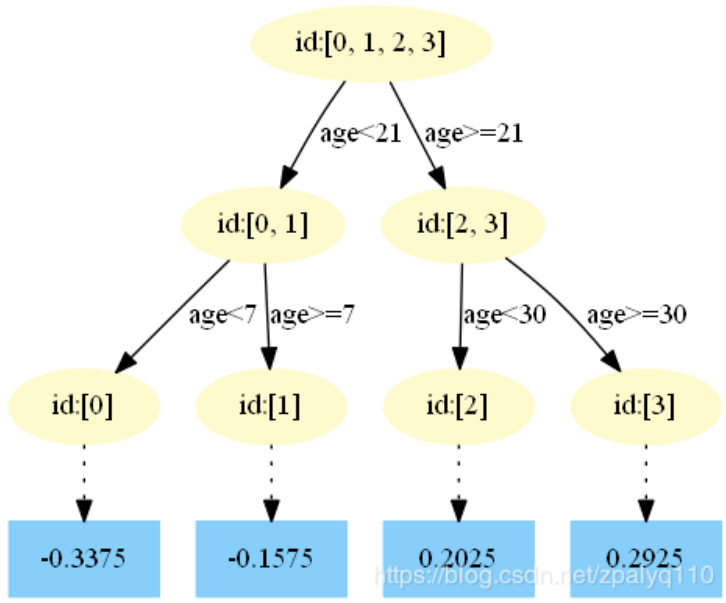
下面将展示每棵树最终的结构，这些图都是GitHub上的代码生成的，感兴趣的同学可以去一探究竟

Github：https://github.com/Freemanzxp/GBDT_Simple_Tutorial

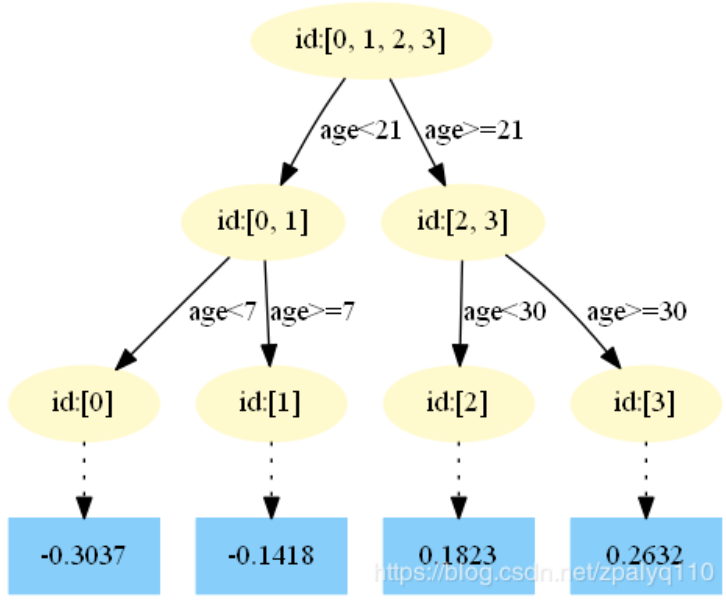
第一棵树：



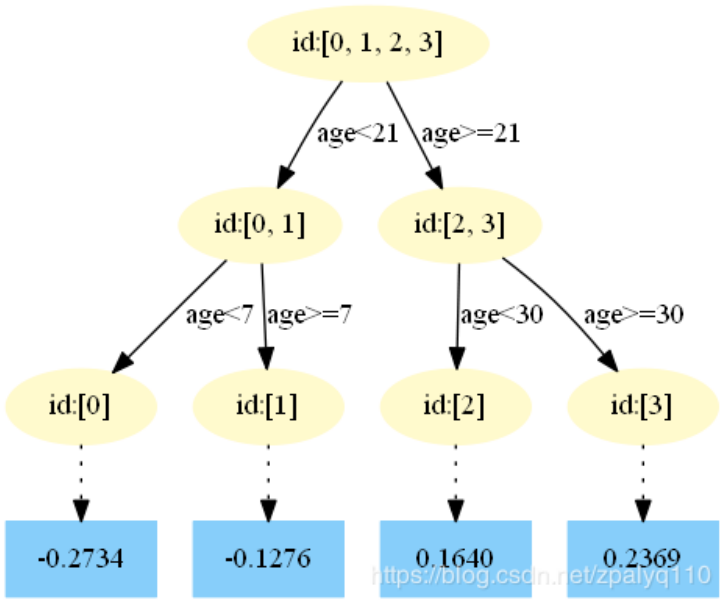
第二棵树：



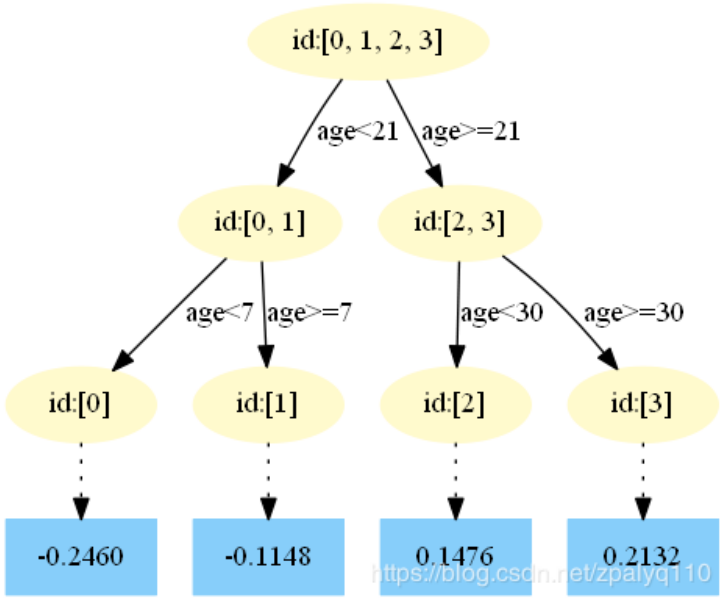
第三棵树：



第四棵树：



第五棵树：



4.得到最后的强学习器：

$$f(x) = f_5(x) = f_0(x) + \sum_{m=1}^{5}\sum_{j=1}^{4} \Upsilon_{jm} I(x \in R_{jm})$$

5.预测样本5：

$f_0(x) = 1.475$

在$f_1(x)$中，样本4的年龄为25，大于划分节点21岁，又小于30岁，所以被预测为**0.2250**。

在$f_2(x)$中，样本4的...此处省略...所以被预测为**0.2025**

为什么是0.2025？这是根据第二颗树得到的，可以GitHub简单运行一下代码

在$f_3(x)$中，样本4的...此处省略...所以被预测为**0.1823**

在$f_4(x)$中，样本4的...此处省略...所以被预测为**0.1640**

在$f_5(x)$中，样本4的...此处省略...所以被预测为**0.1476**

最终预测结果：

$$f(x) = 1.475 + 0.1 * (0.225 + 0.2025 + 0.1823 + 0.164 + 0.1476) = 1.56714$$

---

5. 完整源码

1 依赖环境
操作系统：Windows/Linux

编程语言：Python3

Python库：pandas、PIL、pydotplus,
其中pydotplus库会自动调用Graphviz，所以需要去Graphviz官网下载graphviz的-2.38.msi，先安装，再将安装目录下的 bin 添加到系统环境变量，此时如果再报错可以重启计算机。详细过程不再描述，网上很多解答。

2 文件结构
example.py 回归/二分类/多分类测试文件

GBDT 主模块文件夹

gbdt.py 梯度提升算法主框架

decision_tree.py 单颗树生成，包括节点划分和叶子结点生成

loss_function.py 损失函数

tree_plot.py 树的可视化

3 手写源码

example.py

```
import os
import shutil
import logging
import argparse
import pandas as pd
from GBDT.gbdt import GradientBoostingRegressor
from GBDT.gbdt import GradientBoostingBinaryClassifier
from GBDT.gbdt import GradientBoostingMultiClassifier


logging.basicConfig(level=logging.INFO)
logger = logging.getLogger()
logger.removeHandler(logger.handlers[0])
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
logger.addHandler(ch)


def get_data(model):
    dic = {}
    dic['regression'] = [pd.DataFrame(data=[[1, 5, 20, 1.1],
                                            [2, 7, 30, 1.3],
                                            [3, 21, 70, 1.7],
                                            [4, 30, 60, 1.8],
                                            ], columns=['id', 'age', 'weight', 'label']),
                         pd.DataFrame(data=[[5, 25, 65]], columns=['id', 'age', 'weight'])]

    dic['binary_cf'] = [pd.DataFrame(data=[[1, 5, 20, 0],
                        [2, 7, 30, 0],
                        [3, 21, 70, 1],
                        [4, 30, 60, 1],
                        ], columns=['id', 'age', 'weight', 'label']),
                        pd.DataFrame(data=[[5, 25, 65]], columns=['id', 'age', 'weight'])]

    dic['multi_cf'] = [pd.DataFrame(data=[[1, 5, 20, 0],
                        [2, 7, 30, 0],
                        [3, 21, 70, 1],
                        [4, 30, 60, 1],
                        [5, 30, 60, 2],
                        [6, 30, 70, 2],
                        ], columns=['id', 'age', 'weight', 'label']),
                       pd.DataFrame(data=[[5, 25, 65]], columns=['id', 'age', 'weight'])]

    return dic[model]


def run(args):
    model = None
    # 获取训练和测试数据
    data = get_data(args.model)[0]
    test_data = get_data(args.model)[1]
    # 创建模型结果的目录
    if not os.path.exists('results'):
        os.makedirs('results')
    if len(os.listdir('results')) > 0:
        shutil.rmtree('results')
        os.makedirs('results')
    # 初始化模型
```

```
    if args.model == 'regression':
        model = GradientBoostingRegressor(learning_rate=args.lr, n_trees=args.trees,
max_depth=args.depth,
                                          min_samples_split=args.count, is_log=args.log,
is_plot=args.plot)
    if args.model == 'binary_cf':
        model = GradientBoostingBinaryClassifier(learning_rate=args.lr, n_trees=args.trees,
max_depth=args.depth,
                                                 is_log=args.log, is_plot=args.plot)
    if args.model == 'multi_cf':
        model = GradientBoostingMultiClassifier(learning_rate=args.lr, n_trees=args.trees,
max_depth=args.depth, is_log=args.log,is_plot=args.plot)
    # 训练模型
    model.fit(data)
    # 记录日志
    logger.removeHandler(logger.handlers[-1])
    logger.addHandler(logging.FileHandler('results/result.log'.format(iter), mode='w',
encoding='utf-8'))
    logger.info(data)
    # 模型预测
    model.predict(test_data)
    # 记录日志
    logger.setLevel(logging.INFO)
    if args.model == 'regression':
        logger.info((test_data['predict_value']))
    if args.model == 'binary_cf':
        logger.info((test_data['predict_proba']))
        logger.info((test_data['predict_label']))
    if args.model == 'multi_cf':
        logger.info((test_data['predict_label']))
    pass


if __name__ == "__main__":
    parser = argparse.ArgumentParser(description='GBDT-Simple-Tutorial')
    parser.add_argument('--model', default='regression', help='the model you want to use',
                        choices=['regression', 'binary_cf', 'multi_cf'])
    parser.add_argument('--lr', default=0.1, type=int, help='learning rate')
    parser.add_argument('--trees', default=5, type=int, help='the number of decision trees')
    parser.add_argument('--depth', default=3, type=int, help='the max depth of decision trees')
    # 非叶节点的最小数据数目，如果一个节点只有一个数据，那么该节点就是一个叶子节点，停止往下划分
    parser.add_argument('--count', default=2, type=int, help='the min data count of a node')
    parser.add_argument('--log', default=False, type=bool, help='whether to print the log on
the console')
    parser.add_argument('--plot', default=True, type=bool, help='whether to plot the decision
trees')
    args = parser.parse_args()
    run(args)
    pass
```

gbdt.py

```
"""
Created on : 2019/03/28
@author: Freeman, feverfc1994
"""

import abc
import math
import logging
import pandas as pd
from GBDT.decision_tree import Tree
from GBDT.loss_function import SquaresError, BinomialDeviance, MultinomialDeviance
from GBDT.tree_plot import plot_tree, plot_all_trees,plot_multi
logging.basicConfig(level=logging.INFO)
logger = logging.getLogger()
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)


class AbstractBaseGradientBoosting(metaclass=abc.ABCMeta):
    def __init__(self):
        pass

    def fit(self, data):
        pass

    def predict(self, data):
        pass


class BaseGradientBoosting(AbstractBaseGradientBoosting):

    def __init__(self, loss, learning_rate, n_trees, max_depth,
                 min_samples_split=2, is_log=False, is_plot=False):
        super().__init__()
        self.loss = loss
        self.learning_rate = learning_rate
        self.n_trees = n_trees
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.features = None
        self.trees = {}
        self.f_0 = {}
        self.is_log = is_log
        self.is_plot = is_plot

    def fit(self, data):
        """
        :param data: pandas.DataFrame, the features data of train training
```

```
        """
        # 掐头去尾， 删除id和label，得到特征名称
        self.features = list(data.columns)[1: -1]
        # 初始化 f_0(x)
        # 对于平方损失来说，初始化 f_0(x) 就是 y 的均值
        self.f_0 = self.loss.initialize_f_0(data)
        # 对 m = 1, 2, ..., M
        logger.handlers[0].setLevel(logging.INFO if self.is_log else logging.CRITICAL)
        for iter in range(1, self.n_trees+1):
            if len(logger.handlers) > 1:
                logger.removeHandler(logger.handlers[-1])
            fh = logging.FileHandler('results/NO.{}_tree.log'.format(iter), mode='w',
encoding='utf-8')
            fh.setLevel(logging.DEBUG)
            logger.addHandler(fh)
            # 计算负梯度--对于平方误差来说就是残差
            logger.info(('------------------------------构建第%d颗树------------------------------'
% iter))
            self.loss.calculate_residual(data, iter)
            target_name = 'res_' + str(iter)
            self.trees[iter] = Tree(data, self.max_depth, self.min_samples_split,
                                    self.features, self.loss, target_name, logger)
            self.loss.update_f_m(data, self.trees, iter, self.learning_rate, logger)
            if self.is_plot:
                plot_tree(self.trees[iter], max_depth=self.max_depth, iter=iter)
        # print(self.trees)
        if self.is_plot:
            plot_all_trees(self.n_trees)


class GradientBoostingRegressor(BaseGradientBoosting):
    def __init__(self, learning_rate, n_trees, max_depth,
                 min_samples_split=2, is_log=False, is_plot=False):
        super().__init__(SquaresError(), learning_rate, n_trees, max_depth,
                         min_samples_split, is_log, is_plot)

    def predict(self, data):
        data['f_0'] = self.f_0
        for iter in range(1, self.n_trees+1):
            f_prev_name = 'f_' + str(iter - 1)
            f_m_name = 'f_' + str(iter)
            data[f_m_name] = data[f_prev_name] + \
                             self.learning_rate * \
                             data.apply(lambda x:
self.trees[iter].root_node.get_predict_value(x), axis=1)
        data['predict_value'] = data[f_m_name]


class GradientBoostingBinaryClassifier(BaseGradientBoosting):
    def __init__(self, learning_rate, n_trees, max_depth,
                 min_samples_split=2, is_log=False, is_plot=False):
        super().__init__(BinomialDeviance(), learning_rate, n_trees, max_depth,
                         min_samples_split, is_log, is_plot)

    def predict(self, data):
        data['f_0'] = self.f_0
        for iter in range(1, self.n_trees + 1):
            f_prev_name = 'f_' + str(iter - 1)
            f_m_name = 'f_' + str(iter)
            data[f_m_name] = data[f_prev_name] + \
                             self.learning_rate * \
                             data.apply(lambda x:
self.trees[iter].root_node.get_predict_value(x), axis=1)
        data['predict_proba'] = data[f_m_name].apply(lambda x: 1 / (1 + math.exp(-x)))
        data['predict_label'] = data['predict_proba'].apply(lambda x: 1 if x >= 0.5 else 0)


class GradientBoostingMultiClassifier(BaseGradientBoosting):
    def __init__(self, learning_rate, n_trees, max_depth,
                 min_samples_split=2, is_log=False, is_plot=False):
        super().__init__(MultinomialDeviance(), learning_rate, n_trees, max_depth,
                         min_samples_split, is_log, is_plot)

    def fit(self, data):
        # 掐头去尾， 删除id和label，得到特征名称
        self.features = list(data.columns)[1: -1]
        # 获取所有类别
        self.classes = data['label'].unique().astype(str)
        # 初始化多分类损失函数的参数 K
        self.loss.init_classes(self.classes)
        # 根据类别将'label'列进行one-hot处理
        for class_name in self.classes:
            label_name = 'label_' + class_name
            data[label_name] = data['label'].apply(lambda x: 1 if str(x) == class_name else 0)
            # 初始化 f_0(x)
            self.f_0[class_name] = self.loss.initialize_f_0(data, class_name)
        # print(data)
        # 对 m = 1, 2, ..., M
        logger.handlers[0].setLevel(logging.INFO if self.is_log else logging.CRITICAL)
        for iter in range(1, self.n_trees + 1):
            if len(logger.handlers) > 1:
                logger.removeHandler(logger.handlers[-1])
            fh = logging.FileHandler('results/NO.{}_tree.log'.format(iter), mode='w',
encoding='utf-8')
            fh.setLevel(logging.DEBUG)
            logger.addHandler(fh)
            logger.info(('------------------------------构建第%d颗树------------------------------'
% iter))
            # 这里计算负梯度整体计算是为了计算p_sum的一致性
            self.loss.calculate_residual(data, iter)
            self.trees[iter] = {}
            for class_name in self.classes:
                target_name = 'res_' + class_name + '_' + str(iter)
                self.trees[iter][class_name] = Tree(data, self.max_depth,
```

```
self.min_samples_split,
                                                    self.features, self.loss, target_name,
logger)
                self.loss.update_f_m(data, self.trees, iter, class_name, self.learning_rate,
logger)
            if self.is_plot:
                plot_multi(self.trees[iter], max_depth=self.max_depth, iter=iter)
        if self.is_plot:
            plot_all_trees(self.n_trees)

    def predict(self, data):
        """
        此处的预测的实现方式和生成树的方式不同，
        生成树是需要每个类别的树的每次迭代需要一起进行，外层循环是iter，内层循环是class
        但是，预测时树已经生成，可以将class这层循环作为外循环，可以节省计算成本。
        """
        for class_name in self.classes:
            f_0_name = 'f_' + class_name + '_0'
            data[f_0_name] = self.f_0[class_name]
            for iter in range(1, self.n_trees + 1):
                f_prev_name = 'f_' + class_name + '_' + str(iter - 1)
                f_m_name = 'f_' + class_name + '_' + str(iter)
                data[f_m_name] = \
                    data[f_prev_name] + \
                    self.learning_rate * data.apply(lambda x:
                                                    self.trees[iter]
[class_name].root_node.get_predict_value(x), axis=1)

        data['sum_exp'] = data.apply(lambda x:
                                     sum([math.exp(x['f_' + i + '_' + str(iter)]) for i in
self.classes]), axis=1)

        for class_name in self.classes:
            proba_name = 'predict_proba_' + class_name
            f_m_name = 'f_' + class_name + '_' + str(iter)
            data[proba_name] = data.apply(lambda x: math.exp(x[f_m_name]) / x['sum_exp'],
axis=1)
        # TODO: log 每一类的概率
        data['predict_label'] = data.apply(lambda x: self._get_multi_label(x), axis=1)

    def _get_multi_label(self, x):
        label = None
        max_proba = -1
        for class_name in self.classes:
            if x['predict_proba_' + class_name] > max_proba:
                max_proba = x['predict_proba_' + class_name]
                label = class_name
        return label
```

decision_tree.py

```
"""
Created on : 2019/03/30
@author: Freeman, feverfc1994
"""


class Node:
    def __init__(self, data_index, logger=None, split_feature=None, split_value=None,
is_leaf=False, loss=None,
                 deep=None):
        self.loss = loss
        self.split_feature = split_feature
        self.split_value = split_value
        self.data_index = data_index
        self.is_leaf = is_leaf
        self.predict_value = None
        self.left_child = None
        self.right_child = None
        self.logger = logger
        self.deep = deep

    def update_predict_value(self, targets, y):
        self.predict_value = self.loss.update_leaf_values(targets, y)
        self.logger.info(('叶子节点预测值: ', self.predict_value))

    def get_predict_value(self, instance):
        if self.is_leaf:
            self.logger.info(('predict:', self.predict_value))
            return self.predict_value
        if instance[self.split_feature] < self.split_value:
            return self.left_child.get_predict_value(instance)
        else:
            return self.right_child.get_predict_value(instance)


class Tree:
    def __init__(self, data, max_depth, min_samples_split, features, loss, target_name,
logger):
        self.loss = loss
        self.max_depth = max_depth
        self.min_samples_split = min_samples_split
        self.features = features
        self.logger = logger
        self.target_name = target_name
        self.remain_index = [True] * len(data)
        self.leaf_nodes = []
        self.root_node = self.build_tree(data, self.remain_index, depth=0)

    def build_tree(self, data, remain_index, depth=0):
```

```
        """
        此处有三个树继续生长的条件:
            1: 深度没有到达最大, 树的深度假如是3,  意思是需要生长成3层, 那么这里的depth只能是0, 1
                所以判断条件是 depth < self.max_depth - 1
            2: 点样本数 >= min_samples_split
            3: 此节点上的样本的 target_name 值不一样 (如果值 一样说明已经划分得很好了, 不需要再分)
        """
        now_data = data[remain_index]

        if depth < self.max_depth - 1 \
                and len(now_data) >= self.min_samples_split \
                and len(now_data[self.target_name].unique()) > 1:
            se = None
            split_feature = None
            split_value = None
            left_index_of_now_data = None
            right_index_of_now_data = None
            self.logger.info(('--树的深度: %d' % depth))
            for feature in self.features:
                self.logger.info(('----划分特征: ', feature))
                feature_values = now_data[feature].unique()
                for fea_val in feature_values:
                    # 尝试划分
                    left_index = list(now_data[feature] < fea_val)
                    right_index = list(now_data[feature] >= fea_val)
                    left_se = calculate_se(now_data[left_index][self.target_name])
                    right_se = calculate_se(now_data[right_index][self.target_name])
                    sum_se = left_se + right_se
                    self.logger.info(('------划分值:%.3f,左节点损失:%.3f,右节点损失:%.3f,总损失:%.3f'
%
                                      (fea_val, left_se, right_se, sum_se)))
                    if se is None or sum_se < se:
                        split_feature = feature
                        split_value = fea_val
                        se = sum_se
                        left_index_of_now_data = left_index
                        right_index_of_now_data = right_index
            self.logger.info(('--最佳划分特征: ', split_feature))
            self.logger.info(('--最佳划分值: ', split_value))

            node = Node(remain_index, self.logger, split_feature, split_value, deep=depth)
            """
            trick for DataFrame, index revert
            下面这部分代码是为了记录划分后样本在原始数据中的的索引
            DataFrame的数据索引可以使用True和False
            所以下面得到的是一个bool类型元素组成的数组
            利用这个数组进行索引获得划分后的数据
            """
            left_index_of_all_data = []
            for i in remain_index:
                if i:
                    if left_index_of_now_data[0]:
                        left_index_of_all_data.append(True)
                        del left_index_of_now_data[0]
                    else:
                        left_index_of_all_data.append(False)
                        del left_index_of_now_data[0]
                else:
                    left_index_of_all_data.append(False)

            right_index_of_all_data = []
            for i in remain_index:
                if i:
                    if right_index_of_now_data[0]:
                        right_index_of_all_data.append(True)
                        del right_index_of_now_data[0]
                    else:
                        right_index_of_all_data.append(False)
                        del right_index_of_now_data[0]
                else:
                    right_index_of_all_data.append(False)

            node.left_child = self.build_tree(data, left_index_of_all_data, depth + 1)
            node.right_child = self.build_tree(data, right_index_of_all_data, depth + 1)
            return node
        else:
            node = Node(remain_index, self.logger, is_leaf=True, loss=self.loss, deep=depth)
            if len(self.target_name.split('_')) == 3:
                label_name = 'label_' + self.target_name.split('_')[1]
            else:
                label_name = 'label'
            node.update_predict_value(now_data[self.target_name], now_data[label_name])
            self.leaf_nodes.append(node)
            return node


def calculate_se(label):
    mean = label.mean()
    se = 0
    for y in label:
        se += (y - mean) * (y - mean)
    return se
```

loss_function.py

```
"""
Created on : 2019/03/30
@author: Freeman, foreverfc1994
"""
import math
```

```python
import abc


class LossFunction(metaclass=abc.ABCMeta):

    @abc.abstractmethod
    def initialize_f_0(self, data):
        """初始化 F_0 """

    @abc.abstractmethod
    def calculate_residual(self, data, iter):
        """计算负梯度"""

    @abc.abstractmethod
    def update_f_m(self, data, trees, iter, learning_rate, logger):
        """计算 F_m """

    @abc.abstractmethod
    def update_leaf_values(self, targets, y):
        """更新叶子节点的预测值"""

    @abc.abstractmethod
    def get_train_loss(self, y, f, iter, logger):
        """计算训练损失"""


class SquaresError(LossFunction):

    def initialize_f_0(self, data):
        data['f_0'] = data['label'].mean()
        return data['label'].mean()

    def calculate_residual(self, data, iter):
        res_name = 'res_' + str(iter)
        f_prev_name = 'f_' + str(iter - 1)
        data[res_name] = data['label'] - data[f_prev_name]

    def update_f_m(self, data, trees, iter, learning_rate, logger):
        f_prev_name = 'f_' + str(iter - 1)
        f_m_name = 'f_' + str(iter)
        data[f_m_name] = data[f_prev_name]
        for leaf_node in trees[iter].leaf_nodes:
            data.loc[leaf_node.data_index, f_m_name] += learning_rate * leaf_node.predict_value
        # 打印每棵树的 train loss
        self.get_train_loss(data['label'], data[f_m_name], iter, logger)

    def update_leaf_values(self, targets, y):
        return targets.mean()

    def get_train_loss(self, y, f, iter, logger):
        loss = ((y - f) ** 2).mean()
        logger.info(('第%d棵树: mse_loss:%.4f' % (iter, loss)))


class BinomialDeviance(LossFunction):

    def initialize_f_0(self, data):
        pos = data['label'].sum()
        neg = data.shape[0] - pos
        # 此处log是以e为底, 也就是ln
        f_0 = math.log(pos / neg)
        data['f_0'] = f_0
        return f_0

    def calculate_residual(self, data, iter):
        # calculate negative gradient
        res_name = 'res_' + str(iter)
        f_prev_name = 'f_' + str(iter - 1)
        data[res_name] = data['label'] - 1 / (1 + data[f_prev_name].apply(lambda x: math.exp(-
x)))

    def update_f_m(self, data, trees, iter, learning_rate, logger):
        f_prev_name = 'f_' + str(iter - 1)
        f_m_name = 'f_' + str(iter)
        data[f_m_name] = data[f_prev_name]
        for leaf_node in trees[iter].leaf_nodes:
            data.loc[leaf_node.data_index, f_m_name] += learning_rate * leaf_node.predict_value
        # 打印每棵树的 train loss
        self.get_train_loss(data['label'], data[f_m_name], iter, logger)

    def update_leaf_values(self, targets, y):
        numerator = targets.sum()
        if numerator == 0:
            return 0.0
        denominator = ((y - targets) * (1 - y + targets)).sum()
        if abs(denominator) < 1e-150:
            return 0.0
        else:
            return numerator / denominator

    def get_train_loss(self, y, f, iter, logger):
        loss = -2.0 * ((y * f) - f.apply(lambda x: math.exp(1+x))).mean()
        logger.info(('第%d棵树: log-likelihood:%.4f' % (iter, loss)))


class MultinomialDeviance:

    def init_classes(self, classes):
        self.classes = classes

    @abc.abstractmethod
    def initialize_f_0(self, data, class_name):
        label_name = 'label_' + class_name
        f_name = 'f_' + class_name + '_0'
```

```
        class_counts = data[label_name].sum()
        f_0 = class_counts / len(data)
        data[f_name] = f_0
        return f_0

    def calculate_residual(self, data, iter):
        # calculate negative gradient
        data['sum_exp'] = data.apply(lambda x:
                                     sum([math.exp(x['f_' + i + '_' + str(iter - 1)]) for i in
self.classes]),
                                     axis=1)
        for class_name in self.classes:
            label_name = 'label_' + class_name
            res_name = 'res_' + class_name + '_' + str(iter)
            f_prev_name = 'f_' + class_name + '_' + str(iter - 1)
            data[res_name] = data[label_name] - data[f_prev_name] / data['sum_exp']

    def update_f_m(self, data, trees, iter, class_name, learning_rate, logger):
        f_prev_name = 'f_' + class_name + '_' + str(iter - 1)
        f_m_name = 'f_' + class_name + '_' + str(iter)
        data[f_m_name] = data[f_prev_name]
        for leaf_node in trees[iter][class_name].leaf_nodes:
            data.loc[leaf_node.data_index, f_m_name] += learning_rate * leaf_node.predict_value
        # 打印每棵树的 train loss
        self.get_train_loss(data['label'], data[f_m_name], iter, logger)

    def update_leaf_values(self, targets, y):
        numerator = targets.sum()
        if numerator == 0:
            return 0.0
        numerator *= (self.classes.size - 1) / self.classes.size
        denominator = ((y - targets) * (1 - y + targets)).sum()
        if abs(denominator) < 1e-150:
            return 0.0
        else:
            return numerator / denominator

    def get_train_loss(self, y, f, iter, logger):
        loss = -2.0 * ((y * f) - f.apply(lambda x: math.exp(1+x))).mean()
        logger.info(('第%d棵树: log-likelihood:%.4f' % (iter, loss)))
```

tree_plot.py

```
"""
Created on : 2019/04/7
@author: Freeman, feverfc1994
"""

from PIL import Image
import pydotplus as pdp
from GBDT.decision_tree import Node, Tree
import os
import matplotlib.pyplot as plt


def plot_multi(trees: dict, max_depth: int, iter: int):
    trees_traversal = {}
    trees_nodes = {}
    for class_index in trees.keys():
        tree = trees[class_index]
        res = []
        root = tree.root_node
        traversal(root,res)
        trees_traversal[class_index] = res
        # 获取所有节点
        nodes = {}
        index = 0
        for i in res:
            p, c = i[0], i[1]
            if p not in nodes.values():
                nodes[index] = p
                index = index + 1
            if c not in nodes.values():
                nodes[index] = c
                index = index + 1
        trees_nodes[class_index] = nodes
        # 通过dot语法将决策树展示出来
    trees_edges = {}
    trees_node = {}
    for class_index in trees.keys():
        trees_node[class_index] = ''
        trees_edges[class_index] = ''
    for depth in range(max_depth):
        for class_index in trees.keys():
            for nodepair in trees_traversal[class_index]:
                if nodepair[0].deep == depth:
                    p, c = nodepair[0], nodepair[1]
                    l = len([i for i in range(len(c.data_index)) if c.data_index[i] is True])
                    pname = str(list(trees_nodes[class_index].keys())
[list(trees_nodes[class_index].values()).index(p)])
                    cname = str(list(trees_nodes[class_index].keys())
[list(trees_nodes[class_index].values()).index(c)])
                    if l > 0:
                        trees_edges[class_index] = trees_edges[class_index] + pname + '->' +
cname + '[label=\"' + str(p.split_feature) + (
                                '<' if p.left_child == c else '>=') + str(p.split_value) + '\"]' +
';\n'

                    trees_node[class_index] = trees_node[class_index] + pname +
```

```
'[width=1,height=0.5,color=lemonchiffon,style=filled,shape=ellipse,label=\"id:' + str(
                        [i for i in range(len(p.data_index)) if p.data_index[i] is True]) +
'\"];\n' + \
                            (
                                cname +
'[width=1,height=0.5,color=lemonchiffon,style=filled,shape=ellipse,label=\"id:' + str(
                                    [i for i in range(len(c.data_index)) if
                                     c.data_index[i] is True]) + '\"];\n' if l > 0 else '')
                    if c.is_leaf and l > 0:
                        trees_edges[class_index] = trees_edges[class_index] + cname + '->' +
cname + 'p[style=dotted];\n'
                        trees_node[class_index] = trees_node[class_index] + cname +
'p[width=1,height=0.5,color=lightskyblue,style=filled,shape=box,label=\"' + str(
                            "{:.4f}".format(c.predict_value)) + '\"];\n'
                else:
                    continue
            dot = '''digraph g {\n''' + trees_edges[class_index] + trees_node[class_index] +
'''}'''
            graph = pdp.graph_from_dot_data(dot)
            # 保存图片+pyplot展示
            graph.write_png('results/NO.{}_{}_tree.png'.format(iter, class_index))
        plt.ion()
        plt.figure(1, figsize=(30, 20))
        plt.axis('off')
        plt.title('NO.{} iter '.format(iter))
        class_num = len(trees.keys())
        if class_num / 3 - int(class_num / 3) <0.000001:
            rows = int(class_num/3)
        else:
            rows = int(class_num/3)+1
        for class_index in trees.keys():
            index = list(trees.keys()).index(class_index)
            plt.subplot(rows, 3, index+1)
            img = Image.open('results/NO.{}_{}_tree.png'.format(iter, class_index))
            img = img.resize((1024, 700), Image.ANTIALIAS)
            plt.axis('off')
            plt.title('NO.{}_class {}'.format(iter, class_index))
            plt.rcParams['figure.figsize'] = (30.0, 20.0)
            plt.imshow(img)
        plt.savefig('results/NO.{}_tree.png'.format(iter))
        plt.pause(0.01)


def plot_tree(tree: Tree, max_depth: int, iter: int):
    """
            展示单棵决策树
    :param tree: 生成的决策树
    :param max_depth: 决策树的最大深度
    :param iter: 第几棵决策树
    :return:
    """
    root = tree.root_node
    res = []
    # 通过遍历获取决策树的父子节点关系，可选有traversal 层次遍历 和traversal_preorder 先序遍历
    traversal(root, res)

    # 获取所有节点
    nodes = {}
    index = 0
    for i in res:
        p, c = i[0], i[1]
        if p not in nodes.values():
            nodes[index] = p
            index = index + 1
        if c not in nodes.values():
            nodes[index] = c
            index = index + 1

    # 通过dot语法将决策树展示出来
    edges = ''
    node = ''
    # 将节点层次展示
    for depth in range(max_depth):
        for nodepair in res:
            if nodepair[0].deep == depth:
                # p,c分别为节点对中的父节点和子节点
                p, c = nodepair[0], nodepair[1]
                l = len([i for i in range(len(c.data_index)) if c.data_index[i] is True])
                pname = str(list(nodes.keys())[list(nodes.values()).index(p)])
                cname = str(list(nodes.keys())[list(nodes.values()).index(c)])
                if l > 0:
                    edges = edges + pname + '->' + cname + '[label=\"' + str(p.split_feature) +
(
                        '<' if p.left_child == c else '>=') + str(p.split_value) + '\"]' +
';\n'

                node = node + pname +
'[width=1,height=0.5,color=lemonchiffon,style=filled,shape=ellipse,label=\"id:' + str(
                    [i for i in range(len(p.data_index)) if p.data_index[i] is True]) +
'\"];\n' + \
                        (cname +
'[width=1,height=0.5,color=lemonchiffon,style=filled,shape=ellipse,label=\"id:' + str(
                    [i for i in range(len(c.data_index)) if c.data_index[i] is True]) +
'\"];\n' if l > 0 else '')
                if c.is_leaf and l > 0:
                    edges = edges + cname + '->' + cname + 'p[style=dotted];\n'
                    node = node + cname +
'p[width=1,height=0.5,color=lightskyblue,style=filled,shape=box,label=\"' + str(
                        "{:.4f}".format(c.predict_value)) + '\"];\n'
            else:
                continue
        dot = '''digraph g {\n''' + edges + node + '''}'''
        graph = pdp.graph_from_dot_data(dot)
        # 保存图片+pyplot展示
```

```
        graph.write_png('results/NO.{}_tree.png'.format(iter))
        img = Image.open('results/NO.{}_tree.png'.format(iter))
        img = img.resize((1024, 700), Image.ANTIALIAS)
        plt.ion()
        plt.figure(1, figsize=(30, 20))
        plt.axis('off')
        plt.title('NO.{} tree'.format(iter))
        plt.rcParams['figure.figsize'] = (30.0, 20.0)
        plt.imshow(img)
        plt.pause(0.01)


def plot_all_trees(numberOfTrees: int):
    '''
          将所有生成的决策树集中到一张图中展示
    :param numberOfTrees: 决策树的数量
    :return:
    '''
    # 每行展示3棵决策树 根据决策树数量决定行数
    if numberOfTrees / 3 - int(numberOfTrees / 3) > 0.000001:
        rows = int(numberOfTrees / 3)+1
    else:
        rows = int(numberOfTrees / 3)
    # 利用subplot 将所有决策树在一个figure中展示
    plt.figure(1, figsize=(30,20))
    plt.axis('off')
    try:
        for index in range(1, numberOfTrees + 1):
            path = os.path.join('results', 'NO.{}_tree.png'.format(index))
            plt.subplot(rows, 3, index)
            img = Image.open(path)
            img = img.resize((1000, 800), Image.ANTIALIAS)
            plt.axis('off')
            plt.title('NO.{} tree'.format(index))
            plt.imshow(img)
        plt.savefig('results/all_trees.png', dpi=300)
        plt.show()
        # 由于pyplot图片像素不是很高, 使用方法生成高质量的图片
        image_compose(numberOfTrees)
    except Exception as e:
        raise e


def image_compose(numberOfTrees: int):
    '''
          将numberOfTrees棵决策树的图片拼接到一张图片上
    :param numberOfTrees: 决策树的数量
    :return:
    '''

    png_to_compose = []
    # 获取每张图片的size
    for index in range(1,numberOfTrees+1):
        png_to_compose.append('NO.{}_tree.png'.format(index))
    try:
        path = os.path.join('results', png_to_compose[0])
        shape = Image.open(path).size
    except Exception as e:
        raise e
    IMAGE_WIDTH = shape[0]
    IMAGE_HEIGET = shape[1]
    IMAGE_COLUMN = 3

    if len(png_to_compose)/IMAGE_COLUMN - int(len(png_to_compose)/IMAGE_COLUMN) > 0.0000001:
        IMAGE_ROW = int(len(png_to_compose)/IMAGE_COLUMN)+1
    else:
        IMAGE_ROW = int(len(png_to_compose) / IMAGE_COLUMN)
    # 新建一张用于拼接的图片
    to_image = Image.new('RGB', (IMAGE_COLUMN*IMAGE_WIDTH, IMAGE_ROW*IMAGE_HEIGET), '#FFFFFF')
    # 拼接图片
    for y in  range(IMAGE_ROW):
        for x in range(IMAGE_COLUMN):
            if y*IMAGE_COLUMN+x+1 > len(png_to_compose):
                break
            path = os.path.join('results', 'NO.'+str(y*IMAGE_COLUMN+x+1)+'_tree.png')
            from_image = Image.open(path)
            to_image.paste(from_image, (x*IMAGE_WIDTH, y*IMAGE_HEIGET))

    to_image.save('results/all_trees_high_quality.png')


def traversal_preorder(root: Node, res: list):
    '''

          先序遍历决策树获取节点间的父子关系
    :param root: 决策树的根节点
    :param res:  存储节点对(父节点,子节点)的list
    :return: res
    '''
    if root is None:
        return
    if root.left_child is not None:
        res.append([root, root.left_child])
        traversal_preorder(root.left_child, res)
    if root.right_child is not None:
        res.append([root, root.right_child])
        traversal_preorder(root.right_child, res)


def traversal(root: Node, res: list):
    '''

              层次遍历决策树获取节点间的父子关系
        :param root: 决策树的根节点
```

```
        :param res:  存储节点对(父节点,子节点)的list
        :return: res
        '''
    outList = []
    queue = [root]
    while queue != [] and root:
        outList.append(queue[0].data_index)
        if queue[0].left_child != None:
            queue.append(queue[0].left_child)
            res.append([queue[0], queue[0].left_child])
        if queue[0].right_child != None:
            queue.append(queue[0].right_child)
            res.append([queue[0], queue[0].right_child])
        queue.pop(0)


if __name__ =="__main__":
    plot_all_trees(10)
    # image_compose(10)
```

6. 总结

本文章从GBDT算法的原理到实例详解进行了详细描述，但是目前只写了回归问题，GitHub上的代码也是实现了回归、二分类、多分类以及树的可视化，希望大家继续批评指正，感谢各位的关注。

---------- § ----------

7. 参考资料

1. 李航 《统计学习方法》
2. Friedman J H . Greedy Function Approximation: A Gradient Boosting Machine[J]. The Annals of Statistics, 2001, 29(5):1189-1232.