算法说明

数据集说明

特征值和标签可以是字符串或者数值，可能包含缺失值（数据点的某几项特征值缺失）。

数据点的表示方法$(x_i^m, y_i^m) \in I_m$ 其中x_i^m为特征值向量，y_i^m为标签。m 表示数据点属于I_m，i 表示该数据点在I_m中的 id.

adult_slices 文件里共有 100 个数据集，train.csv 是训练数据集，test.csv 是测试数据集，训练集的数据点为 300 个左右，test 数据集数据点大概 160 个。这一文件的数据集是从 adult 数据集里切分。源数据集网址：https://archive.ics.uci.edu/ml/datasets/adult

算法 1

输入：M 个数据集（$I_1, I_2, \ldots, I_M$）。

输出：

对于数据点x_i^m，得到S_{ij}^m，即在 party j 的数据集I_j上与x_i^m最为相似的数据点，该数据点在I_j上的 id 为 t。表示为$S_{ij}^m = t$.所有数据点的S_{ij}^m集合为$\boldsymbol{S}_m$。

算法 2

简介：

boosting 算法是一种训练多个弱（基）学习器（分类或者回归），并通过结合方法组成一个强学习器的算法过程。弱学习器是以序列方式逐一训练，每轮迭代训练一个弱学习器，每个新训练出来的弱学习器拟合之前所有弱学习器的训练误差。结合方法是将性能好的弱学习器的输出分配更高权重，而性能差的分配较低权重。算法 2 中的弱学习器是决策树，基于的算法是 GBDT（xgboost 版本），处理的是分类任务。

作为在多个数据集上的训练算法，算法 2 利用预训练好的分类器在其他 party 上相似的数据点上的一阶导（g）和二阶导（h），来加强 boosting 算法在本地数据集上的特征训练，使得结果模型预测性能更佳、模型复杂度更低。

输入：训练好的决策树集合（可选输入）、GBDT 参数、数据集（有多个$I_1, I_2, \ldots, I_M$），其中训练集为P_m，其他的数据集作为辅助。算法 1 的输出$\boldsymbol{S}_m$。

输出：新训练的一棵决策树

算法具体过程参照论文。其中第 19 行不需要实现。

说明：在其中一个数据集I_m上训练，其他的数据集作为辅助（传输梯度和海森矩阵值）

和原 GBDT 算法区别：

- GBDT 是一个多次迭代的过程，算法 2 的实现仅为 GBDT 的其中一轮迭代。如果是 GBDT 算法的第一轮迭代，则没有训练好的决策树集合作为输入，否则有该项输入。
- 算法 2 采用的是 GBDT 的 xgboost 实现版本，采用了损失函数的一阶导和二阶导，而原本的 GBDT 算法仅采用了损失函数的一阶导。采用了损失函数二阶导可以加速训练过程，但是会增加额外的计算开销。
- 算法 2 利用了数据集I_m之外的信息，对一阶导和二阶导进行求和，而 GBDT 算法的导数无需求和。

决策树集合的输入项获取：

直接在数据集上训练多个决策树无需拟合残差。

损失函数和导数的计算方法：

二分类：

损失函数

$$L(y, F) = log(1 + exp(-2yF)), y \in -1, 1$$

在数据点 x 处的 g 值计算：

计算残差:

$$\tilde{y}_i = -[\frac{\partial L(y, F(x_i))}{\partial F(x_i)}]_{F(x)=F_{m-1}(x)} = \frac{2y_i}{1 + exp(2y_i F_{m-1}(x_i))} \quad (14)$$

对应于 gbdt_org 中 model.py 文件里

```
100                 residual[id] = 2.0*y_i/(1+exp(2*y_i*f[id]))
```

其中 Fm-1 或者 f(x)对应于代码中的 f 字典，即 gbdt_org 中 model.py 文件里

```
109                 f[id] += learn_rate*node.get_predict_value()
```

h 值计算结果

$$\frac{4\,y^2\,e^{-2\,y\,f(x)}}{\log(2)\left(e^{-2\,y\,f(x)}+1\right)}-\frac{4\,y^2\,e^{-4\,y\,f(x)}}{\log(2)\left(e^{-2\,y\,f(x)}+1\right)^2}$$

所以如果有决策树集合作为输入，I_m每个数据点应该改为（x,-G/H），其中 G 和 H 分别是和 x 相似的数据点的 g 之和和 h 之和. 否则数据点为（x, y）

决策树的训练（对应于算法 2 里的第 18 行）

参考文献 XGBoost: A Scalable Tree Boosting System 里的三种分割点选择方式（算法 1，2，3）文献链接：http://dmlc.cs.washington.edu/data/pdf/XGBoostArxiv.pdf

这三种算法作用于决策树基本算法的第 8 行

6.基本算法

输入：训练集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
　　属性集 $A=\{a_1,a_2,\ldots,a_d\}$.
过程：函数 TreeGenerate(D, A)
1: 生成结点 node;
2: **if** D 中样本全属于同一类别 C **then**
3:　将 node 标记为 C 类叶结点; **return**　　递归返回，情形(1).
4: **end if**
5: **if** $A=\varnothing$ **OR** D 中样本在 A 上取值相同 **then**
6:　将 node 标记为叶结点，其类别标记为 D 中样本数最多的类; **return**　　递归返回，情形(2).
7: **end if**
8: 从 A 中选择最优划分属性 a_*;　　我们将在下一节讨论如何获得最优划分属性.
9: **for** a_* 的每一个值 a_*^v **do**
10:　为 node 生成一个分支；令 D_v 表示 D 中在 a_* 上取值为 a_*^v 的样本子集;
11:　**if** D_v 为空 **then**
12:　　将分支结点标记为叶结点，其类别标记为 D 中样本最多的类; **return**　　递归返回，情形(3).
13:　**else**
14:　　以 TreeGenerate(D_v, $A\setminus\{a_*\}$)为分支结点　　从 A 中去掉 a_*.
15:　**end if**
16: **end for**
输出：以 node 为根结点的一棵决策树

图 4.2　决策树学习基本算法

图片来源 https://zhuanlan.zhihu.com/p/126294494

代码文件介绍：

gbdt_org: 原本的 gbdt 实现，基学习器是决策树，包括二分类和多分类，均只采用一阶导计算。GitHub 网址：https://github.com/liudragonfly/GBDT

gbdt_binary：根据 gbdt_org 改写。在一百个数据集（adult_slices）上采用基于 GBDT 的训练方式（二分类，仅采用了一阶导 g），在每一轮迭代中随机选择 15 个数据集，每个数据

集分别训练一个决策树，训练完毕后将这些决策树汇总起来，用于下一轮决策树训练的残差计算。该过程不断重复直到达到迭代次数上限。可以参考该文件里 input_datas.py 的输入文件读取方式。