

# 数据挖掘第二次项目

## 并行决策树集成

陈铭涛  
16340024

June 29, 2019

## 1 CART 算法

决策树是一种树结构，每一个非叶子节点表示一个对一个特征的分裂，叶子节点存放了分类问题中的类别和回归问题中的数值。其对样本进行预测的方法是从根节点开始，根据每一个分支节点的特征属性，决定输出方向直到到达叶子节点，将叶子节点的数值作为输出值。

决策树的优点为决策过程较为容易令人理解，可解释性强。决策树的构造方法包括了 ID3, C4.5 等算法，在本次项目中主要实现的是 CART 算法。

CART 包含了分类决策树和回归决策树的算法，本次实现了其中的回归树算法。CART 构造出的决策树为一棵二叉树。对于分类问题，CART 算法通过 Gini Index 来计算数据集的纯度，以决定一个节点的分裂，其公式如下：

假设样本集合  $D$  中第  $k$  类样本比例为  $p_k (k = 1, 2 \dots |N|)$ ,

$$Gini(D) = 1 - \sum_{k=1}^{|N|} p_k^2 \quad (1)$$

$Gini(D)$  反映了从  $D$  中随机抽取两个样本其类别不同的概率，越小则代表样本纯度越高。对于属性集合  $A$  上的属性  $a$ ，将所有  $D$  上取值为  $a^v$  的样本记为  $D^v$ ，则  $a$  上的 Gini Index 为：

$$Gini(D, a) = \sum_{v=1}^{|V|} \frac{|D^v|}{|D|} Gini(D^v) \quad (2)$$

选择  $A$  中的划分点即为

$$a_* = \arg \min_{a \in A} Gini(D, a) \quad (3)$$

当要解决的问题是回归问题时，假设样本集合  $D$  中第  $i$  个样本的标签为  $y_i$ ，则最小化的目标为回归标签的平方误差和，即

$$SSE(D) = \sum_{i=1}^{|N|} (y_i - \bar{y})^2 \quad (4)$$

对于在属性  $a$  上的  $v$  值分裂的节点，设其左子树和右子树上的样本集合分别为  $D^L, D^R$ ，则其平方误差和为：

$$SSE(D, a, v) = SSE(D^L) + SSE(D^R) \quad (5)$$

则对于属性  $a$  上的划分点的选择为：

$$a_* = \arg \min_{a \in A, v \in V} SSE(D, a, v) \quad (6)$$

CART 算法进行决策树构建的方法如下：

---

**Algorithm 1:** CART 决策树构建

---

**Result:** 分类决策树或回归决策树  
 从深度为 0 开始构建;  
 对各连续特征列进行排序;  
**while** 未达到终止条件 **do**  
     **if** 特征为连续特征 **then**  
         从排序好的特征列中对每一个取值进行评判标准的计算;  
         选取令评判标准最小的取值作为分裂点;  
     **else if** 特征为类别特征 **then**  
         从样本中选取令评判标准最小的类别进行分裂;  
         根据分裂点创建新的决策树节点  
**end**

---

其中的终止条件如下：

1. 若节点中的所有样本的标签值均相同
2. 若树的深度已达到用户定义的MaxDepth数值
3. 若节点的样本数少于用户定义的MinSamplesSplit数值
4. 若节点的分裂生成的子节点样本数少于用户定义的MinSamplesLeaf数值

在实现中，由于本次项目的数据特征均为连续特征，只实现了在连续特征下的回归决策树构建。

由于对于连续变量的特征， $|V|$  可能是一个较大的数值，此时对连续变量的每一个不同值进行扫描的计算代价非常大，因此实现中添加了MaxBin参数，在寻找分裂点时程序将会将特征列分为对应的段数，每段仅测试一个分裂点。将该参数调大可以提升训练结果，但是会增加训练时间。

## 2 并行化实现

在决策树训练过程中可选的实现并行化的位置为 [1]：

1. 并行化进行对每一层节点的分裂
2. 每个节点的分裂中对每个特征进行并行化处理
3. 在每一层中对每个特征进行并行化处理

在实现的代码中，选择的是在训练一层的过程中对各节点与特征进行并行化处理，每个节点上测试完所有特征的分裂可能后根据评判标准选择最优的分裂点进行分裂。

### 3 Gradient Boosting

Gradient Boosting 是一种结合多个弱学习器的集成方法，每一步中训练的模型尝试去“纠正”先前输出的模型。

设训练集为  $\{(x_i, y_i)\}_{i=1}^n$ ，模型预测值  $\hat{y} = F(x)$ ，学习率  $\alpha$ ，使用的可微分损失函数为均方误差：

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (7)$$

对损失函数除 2 后求偏导数得到：

$$\frac{\partial L(y, F(x))}{\partial F(x)} = F(x) - y \quad (8)$$

即模型输出与标签的残差取负数。由梯度下降算法的思想可知延残差方向移动可降低损失函数。由此可得当损失函数为均方误差时 Gradient Boosting 中每步的模型更新方法。

综上，令算法迭代次数为  $M$ ，Gradient Boosting 算法如下：

---

**Algorithm 2:** Gradient Boosting

---

**Result:** 模型  $F_M(x)$

初始化  $F_0(x) = \arg \min_{\gamma} L(y, \gamma)$ ;

**for**  $m = 1$  **to**  $M$  **do**

1. 计算残差数值：

$$r_m = - \left[ \frac{\partial L(y, F_{m-1}(x))}{\partial F_{m-1}(x)} \right] = y - F_{m-1}(x) \quad (9)$$

2. 以  $\{(x, r_m)\}$  作为训练集训练一个弱学习模型（此处为决策树） $h_m(x)$

3. 更新模型  $F_m(x) = F_{m-1}(x) + \alpha h_m(x)$

**end**

---

在代码实现中，每次迭代增加了对学习率的从 0 至 1 的线性搜索，从而不使用固定的学习率，而是在每一步中找到令当前步最优的学习率：

$$\alpha_m = \arg \min_{\alpha} L(y, F_{m-1}(x) + \alpha h_m(x)) \quad (10)$$

学习率搜索过程可以并行进行，使用这种方法可以在较少迭代次数下较固定学习率获得更好的成绩，在迭代数较大的时候分数可能不如固定较小的学习率。

由以上算法说明可知 Gradient Boosting 整体是串行运行的，因此在 Gradient Boosting 算法中只能对基学习器即决策树的训练部分进行并行的实现。

### 4 Random Forest

Random Forest 采用 Bagging 的方法进行决策树集成。Bagging 集成的算法如下：

输入训练集  $(X, Y)$ ，训练轮数  $M$ ，采样大小  $n$ ，基学习器训练算法  $H$

对于训练集有放回采样获得  $M$  个大小为  $n$  的训练集  $\{(X_i, Y_i)\}_{i=1}^M$

---

**Algorithm 3:** Bagging

---

**Result:** 模型  $F(x)$

**for**  $m = 1$  *to*  $M$  **do**

$h_m = H(X_m, Y_m)$

**end**

对于分类问题，采用投票法获得最终模型，对于回归问题，采用取平均方法获得最终模型；

---

随机森林算法则在 Bagging 算法的基础上增加了随机属性选择，即在决策树中的每个节点，先从属性集合中选出一个大小为  $k$  的子集，再从该子集中选择划分点进行分裂。 $k$  的取值决定了模型的随机程度。

随机森林中通过加入属性扰动来对只有样本扰动的 Bagging 集成方法进行改进，使得最终集成泛化性因为个体学习器之间的差异性增加而增加。

由随机森林的算法可知，不同的基学习器之间是可以并行进行训练的，但是由于在实现中已经对决策树实现了并行训练，若对基学习器也并行化只会增加上下文切换的开销，因此没有实现基学习器的并行训练。

## 5 代码实现

出于内存、速度和并行化实现的考虑，本次项目选择了使用 Rust 语言实现，原因是 Rust 的 RAII 机制使得资源可以及时地释放，提升内存利用率；由编译器提供的静态检查可以避免多线程时线程不安全的情况，降低 debug 难度；作为通过 LLVM 后端编译为机器代码的静态语言 Rust 可以在相同的实现下获得比 Python 等语言更高的速度。使用的编译器版本为 `rustc 1.35.0`。

程序实现中使用的第三方库如下：

1. rayon: 提供基于迭代器的便捷地编写并行代码的方法
2. rand: 提供随机数生成
3. csv: 提供对 csv 文件的读取
4. indicatif: 提供命令行进度条实现
5. ndarray: 提供类似 numpy 的多维数组的操作
6. num-traits: 提供数值类型上的一些实用方法，如最大最小值等
7. log: 程序日志
8. pretty\_env\_logger: 程序日志输出
9. num\_cpus: 获取系统 CPU 核心数量
10. serde: 提供将结构体变量序列化的功能
11. serde\_json: 用于以 json 格式将序列化后的模型变量保存至文件

项目中包括的主要代码文件如下

- data\_frame.rs: 使用一个二维的 `ndarray` 作为程序使用的 `DataFrame` 类型，并定义了类型别名 `V` 作为全局的数据存储类型，可设为 `f64` 或 `f32`，此外还包含了 `csv` 文件读写等其他实用函数
- learner.rs: 定义了一个学习器 `Learner` trait，包括了类似于 `sklearn` 的 `fit` 和 `predict` 两个方法，决策树，Boosting 和 Random Forest 都需实现该 trait。
- tree.rs: 包括了 `DecisionTree` 类型，实现了 CART 算法，可对单个决策树进行训练和预测。
- boosting.rs: 包括了 `GradientBoosting` 类型的定义与训练和预测的实现。
- random\_forest.rs: 包括了 `RandomForest` 类型的定义与训练和预测的实现。
- utils 目录: 包括了数个实用功能，如获取数据列排序序列，交叉验证，模型分数计算等。

bin 目录下的包含 main 函数的可执行代码文件如下：

- boost\_cv.rs: 使用 GDBT 进行交叉验证
- boost\_predict.rs: 使用 GDBT 进行训练并输出在测试集上的预测结果
- cv.rs: 使用单棵决策树进行交叉验证
- parallel\_performance.rs: 接收一个命令行整数作为程序最多使用的线程数，进行一次单棵决策树的训练，用于测试并行化的效率
- predict.rs: 使用单棵决策树进行训练并输出测试集预测结果
- rf\_cv.rs: 使用随机森林进行交叉验证
- rf\_predict.rs: 使用随机森林进行训练并输出测试集预测结果

调参的方式为在 bin 目录下的可执行代码文件中找到 `tree_config` 和 `boost_config` 或 `forest_config` 并修改对应参数字段的数值。

程序运行前需将训练数据放置在项目上层目录下的 data 文件夹下。

运行任一 bin 目录下的代码的方法为在项目目录下执行命令(\*nix 系统下),其中 `EMTM_LOG=info` 的作用是使程序将日志输出到命令行：

---

```
EMTM_LOG=info cargo run --release --bin {EXEC_NAME}
```

---

所有代码都需在 `release` 模式下进行编译，否则速度可能会有 20 到 100 倍的减慢。

## 6 并行化表现

由于并行化的实现主要位于单棵决策树的训练中，对于并行化表现的测试主要针对单棵决策树的训练与预测。

在命令行下使用如下命令测试了从单线程到 12 线程下对单棵决策树训练时的时间：

---

```
for ((i = 1; i<=12;i++)) cargo run --release --bin parallel_performance $i
>>../threads_performance.txt
```

---

将获得的训练速度相对单线程下训练速度的提升数据进行绘图如下：

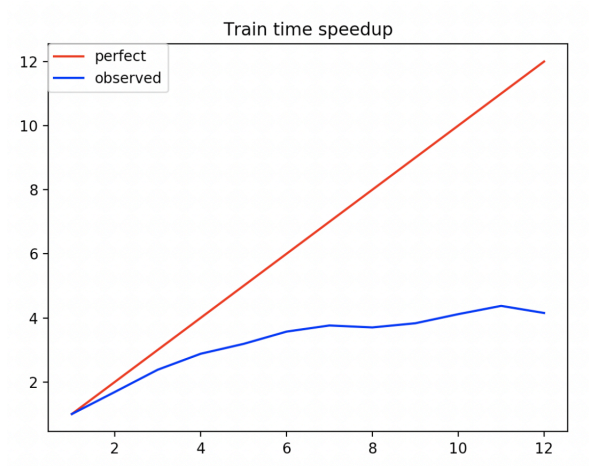


Figure 1: 并行训练速度提升情况

预测速度的数据进行绘图如下：

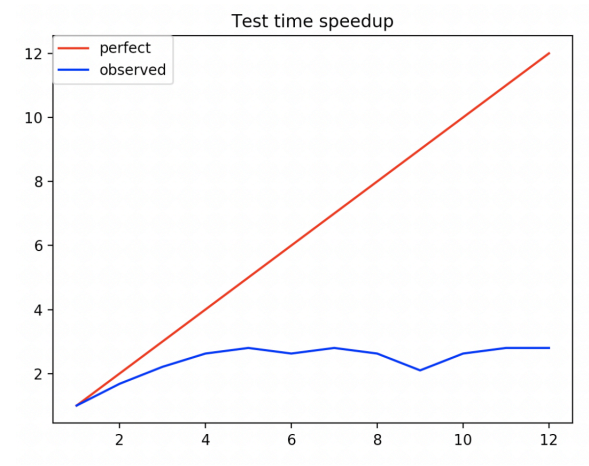


Figure 2: 并行预测速度提升情况

可见当线程数提升到一定程度下对训练速度的提升大约收敛在 4 倍，而对预测速度的提升不大。

## 7 验证

以下程序测试均在一台搭载 6 核 12 线程 CPU，运行 macOS 系统的笔记本电脑上运行，数据精度为 *f64*。

验证的标准为  $R^2$ ，其计算方法如下：

$$\begin{aligned}\bar{y} &= \frac{1}{n} \sum_{i=1}^n y_i, \\ SS_{tot} &= \sum_i (y_i - \bar{y})^2, \\ SS_{res} &= \sum_i (y_i - f_i)^2, \\ R^2 &= 1 - \frac{SS_{res}}{SS_{tot}}\end{aligned}\tag{11}$$

其中  $y_i$  为第  $i$  个样本的实际观察值,  $f_i$  为第  $i$  个样本的模型预测值。  $R^2$  的取值通常在 0 与 1 之间, 越接近 1 代表预测值与真实值匹配程度越高。

使用 LightGBM [2] 构建一个模型运行 3 折交叉验证进行对比:

```
In [6]: run_cross_validation(trains, labels, lgb.LGBMRegressor, params)

fit_time: [24.27433276 22.56340122 23.01793098]
Average fit_time: 23.285222
score_time: [10.66991425 9.20692468 8.52207994]
Average score_time: 9.466306
test_mse: [-0.33291203 -0.33487001 -0.34085721]
Average test_mse: -0.336213
train_mse: [-0.33630982 -0.33519583 -0.33240139]
Average train_mse: -0.334636
test_r2: [0.156102 0.15480654 0.15376245]
Average test_r2: 0.154890
train_r2: [0.15816958 0.15915648 0.15922097]
Average train_r2: 0.158849
```

Figure 3: LightGBM 交叉验证结果

对单棵决策树进行交叉验证获得的结果如下:

在训练集上获得的平均  $R^2$  分数为 0.14947528261278345

在验证集上获得的平均  $R^2$  分数为 0.143995380629807

训练时间平均为 682423 ms.

```
Finished release [optimized + debuginfo] target(s) in 0.84s
Running `target/release/cv`
INFO cv > Train data shape: [10000004, 13]
INFO cv > Label data shape: [1, 10000004]
INFO cv > Load time: 6919ms
INFO ensembles_rs::utils::cross_validate > train time: 712503, predict time: 295
INFO ensembles_rs::utils::cross_validate > train: 0.14879519294338295, validation: 0.14471150292761092
INFO ensembles_rs::utils::cross_validate > train time: 658188, predict time: 254
INFO ensembles_rs::utils::cross_validate > train: 0.1489242194641912, validation: 0.14538125132877167
INFO ensembles_rs::utils::cross_validate > train time: 676580, predict time: 341
INFO ensembles_rs::utils::cross_validate > train: 0.15070643543077622, validation: 0.14189338763303838
CrossValidateScore { train_time: [712503, 658188, 676580], predict_time: [295, 254, 341],
train_score: [0.14879519294338295, 0.1489242194641912, 0.15070643543077622], validation_score: [0.14471150292761092, 0.14538125132877167, 0.14189338763303838] }
```

Figure 4: 单棵决策树训练交叉验证结果

使用 Gradient Boosting 训练 150 步, 设置单棵树最大生长至 2 层, 进行交叉验证获得的结果如下:

在训练集上获得的平均  $R^2$  分数为 0.14595633826167811

在验证集上获得的平均  $R^2$  分数为 0.14329480642795986

训练时间平均为 223031 ms.

```
lr: 0.13
INFO ensembles_rs::boosting > lr 0.12 at step 149.
INFO ensembles_rs::boosting > Pred Score: 0.14566673131840036
lr: 0.12
INFO ensembles_rs::utils::cross_validate > train time: 217482, predict time: 767
INFO ensembles_rs::utils::cross_validate > train: 0.14566673131840036, validation: 0.1455456685234573
CrossValidateScore { train_time: [220638, 230975, 217482], predict_time: [1013, 819, 767],
train_score: [0.147332346601316, 0.14486993686531802, 0.14566673131840036], validation_score: [0.1384225183769906, 0.14591623238343165, 0.1455456685234573] }
```

Figure 5: GBDT 训练交叉验证结果

使用 Random Forest 训练, 决策树数量为 150, 不限制决策树生长深度, 进行交叉验证获得的结果如下:

在训练集上获得的平均  $R^2$  分数为 0.15712251333333332

在验证集上获得的平均  $R^2$  分数为 0.15363599666666664

训练时间平均为 1355728 ms.

```
2019-06-03 12:27:33 INFO ensembles_rs::random_forest > score at step 98: 0.09041327
2019-06-03 12:27:44 INFO ensembles_rs::random_forest > score at step 99: 0.10266656
2019-06-03 12:28:00 INFO ensembles_rs::utils::cross_validate > train time: 1441653, predict time: 5298
2019-06-03 12:28:00 INFO ensembles_rs::utils::cross_validate > train: 0.15630662, validation: 0.1556645
CrossValidateScore { train_time: [1403810, 1221723, 1441653], predict_time: [5083, 4956, 5298], train_score: [0.15816838, 0.15689254, 0.15630662], validation_score: [0.15134245, 0.15390104, 0.1556645] }
```

Figure 6: 随机森林训练交叉验证结果

四个训练中使用取样工具查看内存占用峰值分别如下:

1. LightGBM: 5.2G



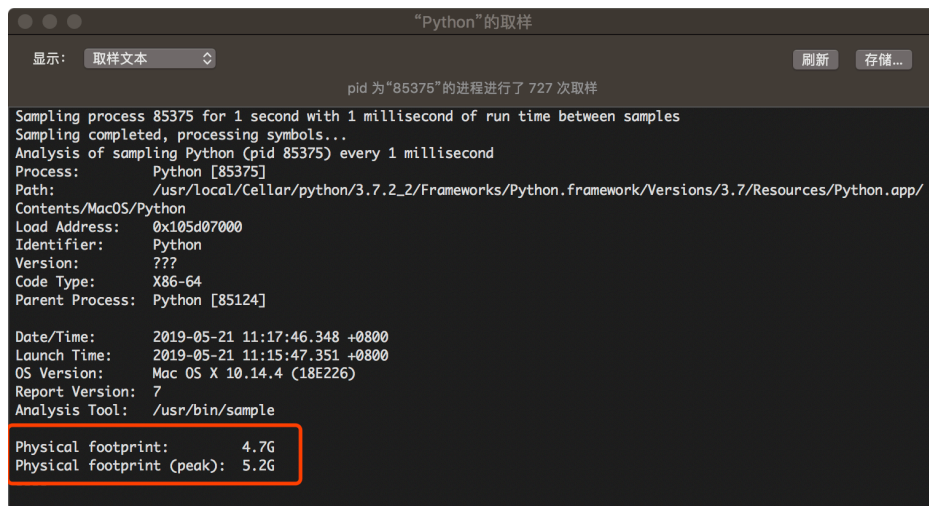


Figure 7: LightGBM 内存占用

## 2. 单决策树: 3.2G

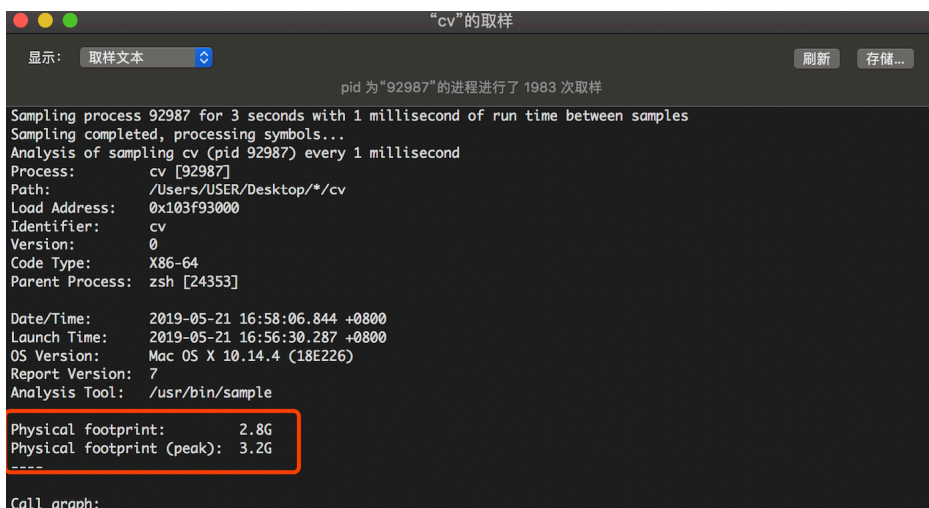
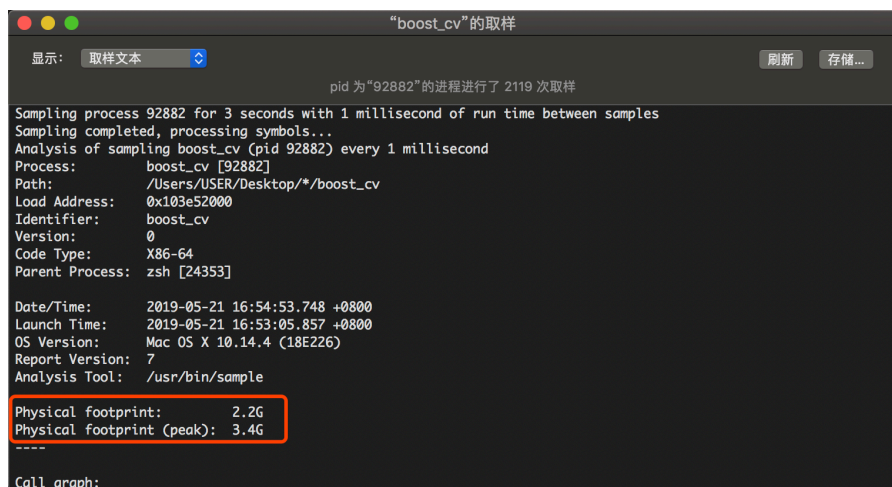


Figure 8: 单棵决策树训练内存占用

## 3. Gradient Boosting: 3.4G



```
显示: 取样文本 刷新 存储...
pid 为 "92882" 的进程进行了 2119 次取样

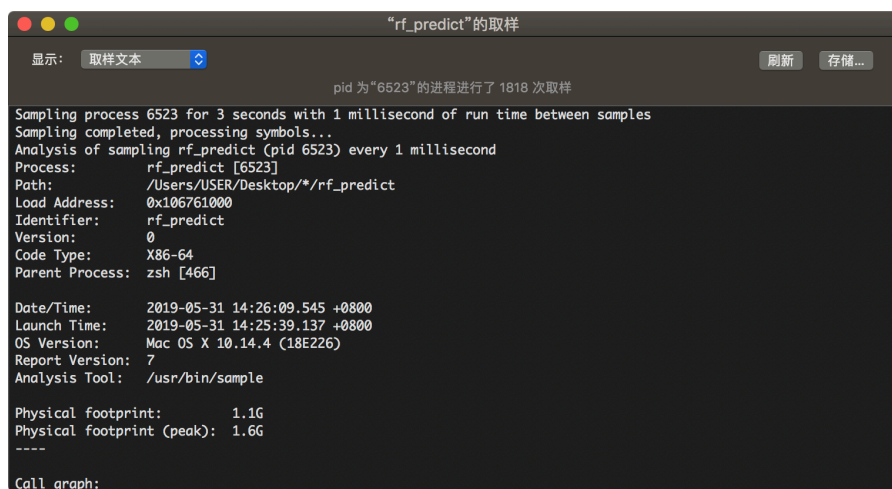
Sampling process 92882 for 3 seconds with 1 millisecond of run time between samples
Sampling completed, processing symbols...
Analysis of sampling boost_cv (pid 92882) every 1 millisecond
Process:      boost_cv [92882]
Path:         /Users/USER/Desktop/*/boost_cv
Load Address: 0x103e52000
Identifier:    boost_cv
Version:      0
Code Type:    X86-64
Parent Process: zsh [24353]

Date/Time:    2019-05-21 16:54:53.748 +0800
Launch Time:  2019-05-21 16:53:05.857 +0800
OS Version:   Mac OS X 10.14.4 (18E226)
Report Version: 7
Analysis Tool: /usr/bin/sample

Physical footprint: 2.2G
Physical footprint (peak): 3.4G
-----
Call graph:
```

Figure 9: Gradient Boosting 训练内存占用

#### 4. Random Forest: 1.6G



```
显示: 取样文本 刷新 存储...
pid 为 "6523" 的进程进行了 1818 次取样

Sampling process 6523 for 3 seconds with 1 millisecond of run time between samples
Sampling completed, processing symbols...
Analysis of sampling rf_predict (pid 6523) every 1 millisecond
Process:      rf_predict [6523]
Path:         /Users/USER/Desktop/*/rf_predict
Load Address: 0x106761000
Identifier:    rf_predict
Version:      0
Code Type:    X86-64
Parent Process: zsh [466]

Date/Time:    2019-05-31 14:26:09.545 +0800
Launch Time:  2019-05-31 14:25:39.137 +0800
OS Version:   Mac OS X 10.14.4 (18E226)
Report Version: 7
Analysis Tool: /usr/bin/sample

Physical footprint: 1.1G
Physical footprint (peak): 1.6G
-----
Call graph:
```

Figure 10: Random Forest 训练内存占用

## 8 Kaggle 分数

使用单棵决策树训练至 10 层后提交至 Kaggle 获得的分数为 0.16087:

Submission and Description	Public Score	Use for Final Score
<a href="#">decision-tree-10.csv</a> 2 hours ago by Miguel Chan <a href="#">add submission details</a>	0.16087	<input type="checkbox"/>

Figure 11: 单棵决策树分数

使用 Gradient Boosting, Learning Rate 固定为 0.25, 基学习器最大训练至 3 层, 训练步数为 400 时的分数为 0.16957:

[GBDT-400-3-6-400-0.25-0.25.csv](#)

0.16957

17 hours ago by [Miguel Chan](#)

[add submission details](#)

Figure 12: lr=0.25, GBDT

使用 Random Forest, 限制单棵决策树生长至 10 层, 属性子集大小为 3, SubSample 取 0.632, 分数为 0.17433:

[sub.csv](#)

0.17433

3 hours ago by [Miguel Chan](#)

[add submission details](#)

Figure 13: 随机森林

## References

- [1] Zhanpeng Fang. Parallel gradient boosting decision trees, May 2015.
- [2] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 3146–3154. Curran Associates, Inc., 2017.