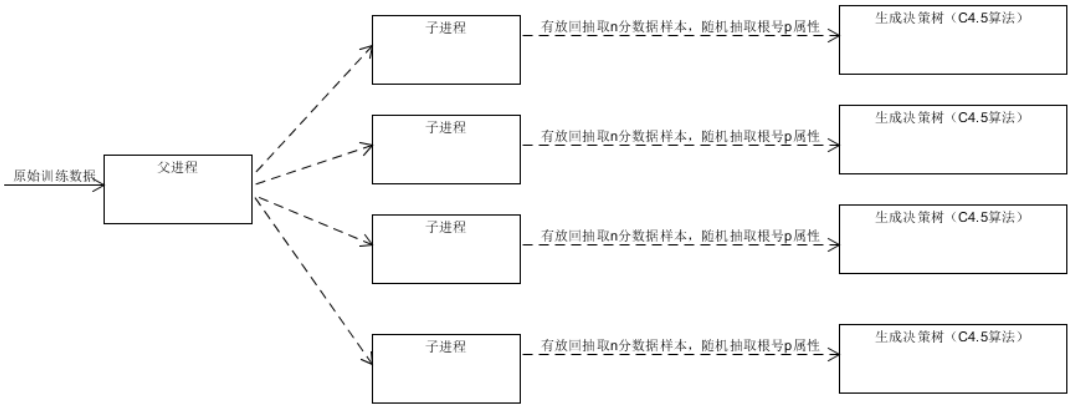
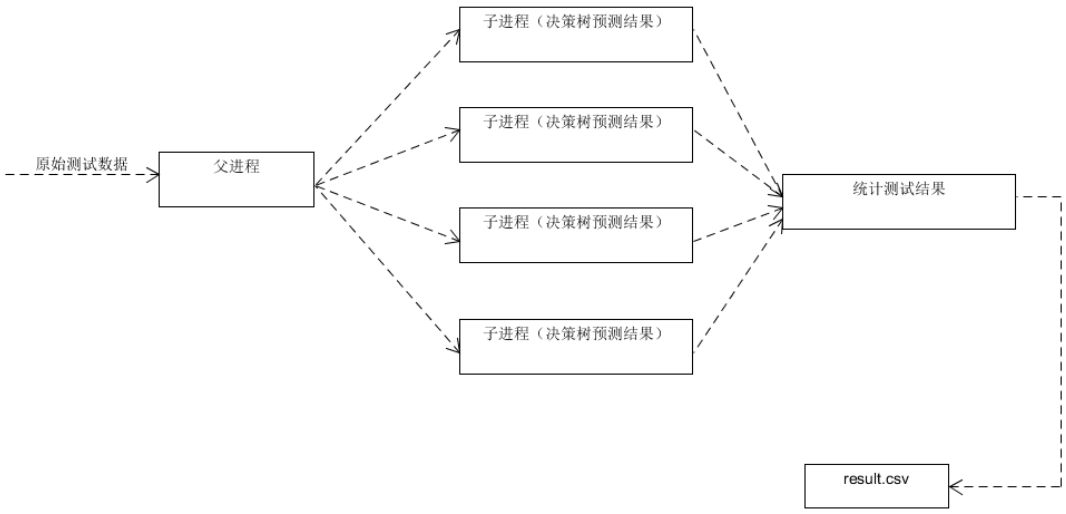
# Random Forest 实验报告

## 一．问题分析

（1）本次实验的数据集是第二次作业中使用非线性预测分类的问题。在题目中，训练集一共有 6238 条数据，而测试集一共有 1559 条数据，一共有 617 个属性，分类结果有 26 个标签。使用随机森林算法先用训练数据进行训练，训练好模型之后，最后用训练的模型对测试数据进行测试，最后比较结果正确性，训练算法流程如图所示：




对测试数据进行预测的时候，算法流程图如下：




## 二．算法分析与设计

（1）随机森林即通过构建随机的分类决策树，最后通过分类决策树的投票决定最后的分类，有点类似于“三个臭皮匠，胜过一个诸葛亮”的思想。因此，要想构建随机森林，首先一定要构建决策树，而在题目之中，使用了连续区间之间属性值，因此采取了 C4.5 的算法来构建决策树

（2）C4.5 决策树算法主要改进于 ID3 决策树算法。在 C4.5 的决策树算法流程图如下：

**Input:** an attribute-valued dataset $D$

Tree = {}
**if** $D$ is “pure” OR other stopping criteria met **then**
    terminate
**end if**
**for all** attribute $a \in D$ **do**
    Compute information-theoretic criteria if we split on $a$
**end for**
$a_{best}$ = Best attribute according to above computed criteria
Tree = Create a decision node that tests $a_{best}$ in the root
$D_v$ = Induced sub-datasets from $D$ ba image $a_{best}$
**for all** $D_v$ **do**
    $\text{Tree}_v = \text{C4.5}(D_v)$
    Attach $\text{Tree}_v$ to the corresponding branch of Tree
**end for**
**return** Tree

（3）在 C4.5 算法之中，最重要的是使用的信息增益率的方法对节点进行分裂，避免了信息增益带来的一些问题。信息增益率定义如下：

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

其中:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

$$Gain(A) = Info(D) - Info_A(D)$$

Gain(A)为 ID3 算法之中的信息增益，在 ID3 算法之中：

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

（4）信息熵选择具有最高信息增益的属性来作为节点 N 的分裂属性。该属性使结果划分中的元组分类所需信息量最小

（5）信息增益定义为原来的信息需求（即仅基于类比例）与新需求（即对 A 划分之后得到的）之间的差

（6）信息增益率使用“分裂信息”值将信息增益规范化，是对信息增益的一种更加科学标准的度量

（7）在使用 C4.5 算法中，可以处理连续区间的值，而不是离散的分类决策，具体做法是遍历所有样本对应属性中所有不同的值，然后对所有不同的值进行排序，最后所有对应的值都进行子树划分，选出最优的划分策略

（8）构建完 C4.5 决策树之后，就可以进行随机森林的处理。首先，对于要生成的决策树，采取随机又放回抽样方法，随机抽取 n 个样本作为决策树的训练样本（其中，n 等于原来样本的个数 N），在这些样本之中，随机不放回抽取 m 个属性进行测试（m << M，其中 M 为原来样本属性值，这里 m 取根号 M）。最后，重复上述方法生成不同的决策树，并用生成的训练数据训练决策树。最后通过投票取最多的方式对测试数据进行预测

（9）随机森林算法流程图如下：

**Require:** Sequential training example $\langle x, y \rangle$
**Require:** The size of the forest: $T$
**Require:** The minimum number of samples: $\alpha$
**Require:** The minimum gain: $\beta$

// For all trees
**for** $t$ from 1 to $T$ **do**
  k $\leftarrow$ Poisson($\lambda$)
  **if** $k > 0$ **then**
    // Update k times
    **for** $u$ from 1 to $k$ **do**
      $j = \text{findLeaf}(x)$.
      updateNode$(j, \langle x, y \rangle)$.
      **if** $|\mathcal{R}_j| > \alpha$ and $\exists s \in \mathcal{S} : \Delta L(\mathcal{R}_j, s) > \beta$ **then**
        Find the best test:
        $s_j = \arg\max_{s \in \mathcal{S}} \Delta L(\mathcal{R}_j, s)$.
        createLeftChild$(\mathbf{p}_{jls})$
        createRightChild$(\mathbf{p}_{jrs})$
      **end if**
    **end for**
  **else**
    Estimate $OOBE_t \leftarrow$ updateOOBE$(\langle x, y \rangle)$
  **end if**
**end for**
Output the forest $\mathcal{F}$.

## 三．算法优化与并行化

（1）单决策树算法优化：

(1) 在 C4.5 算法中，需要遍历所有属性的不同值作为分裂决策的依据。但是，通过对训练数据进行分析可以知道，虽然不同的值有很多，但是他们大部分非常接近，没有必要独立作为不同的分割值，而且，他们之间都在区间[-1, 1]上。因此，经过测试权衡，发现采取 5-10 个阈值就可以了，没有必要采取太多的值。在采取值的时候，使用的等区间分割的方法，即找出最小和最大，然后在最大值与最小值之间进行等区间划分

(2) 在决策树上，是根据信息增益率进行划分的，实际上等到被划分的属性越来越多，剩下的属性的重要性以及可分性权值就会越来越低。因此，可以采用限制决策树深度或者限制决策树分类的数量来进行优化。比如，如果剩下待划分属性很少（比如低于 5 个），可以停止分裂属性了。又或者划分的树深度已经到达某一个值（比如大于 8），这时候也可以停止继续分裂。通过这些方式，生成决策树的速度会大大加快

(3) 在代码优化上，可以将递归构建决策树变成非递归形式实现，也可以通过尾递归形式实现代码等策略，这样能够节省栈空间分配以及调用函数的时间，可以大大加快代码运行速度

（2）随机森林并行化优化：

(1) 随机森林每颗决策树是相互独立的，因此，完全可以采用并行化策略进行优化。并行化策略有线程并行，进程并行（多核并行）以及多机并行。针对测试数据集的特点，在代码中采取了多进程并行的方式。多线程以及多进程能够充分利用 CPU 的吞吐量，从而实现计算资源的最大化利用，不失为单机情况下优化算法的选择。

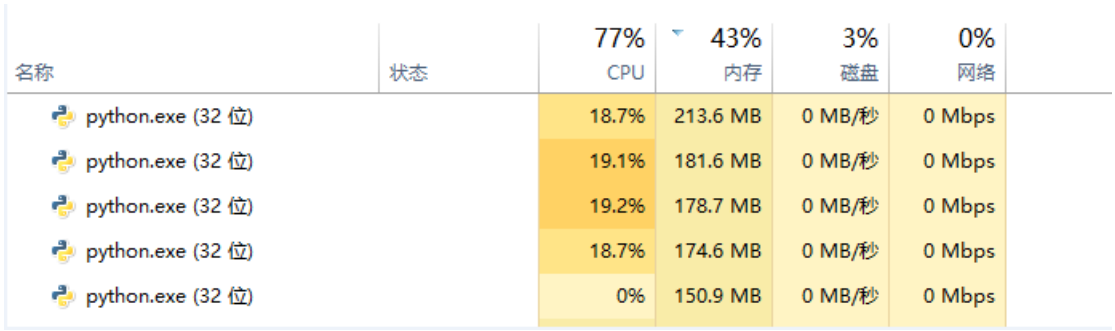
## 四．代码实现及结果分析

（1）本代码测试了串行化和并行化之间的时间差距：

单进程情况下：


这个时候实际上 CPU 的吞吐量仍然没有饱和，此时训练 100 课树的时间为：


使用多进程的情况下（图中挂起的(CPU 为 0%)是主进程）：



| 名称 | 状态 | 77% CPU | 43% 内存 | 3% 磁盘 | 0% 网络 |
|---|---|---|---|---|---|
| python.exe (32 位) | | 18.7% | 213.6 MB | 0 MB/秒 | 0 Mbps |
| python.exe (32 位) | | 19.1% | 181.6 MB | 0 MB/秒 | 0 Mbps |
| python.exe (32 位) | | 19.2% | 178.7 MB | 0 MB/秒 | 0 Mbps |
| python.exe (32 位) | | 18.7% | 174.6 MB | 0 MB/秒 | 0 Mbps |
| python.exe (32 位) | | 0% | 150.9 MB | 0 MB/秒 | 0 Mbps |

训练 100 棵树运行时间为：


可以看出，使用多进程能够使速度提高到原来的 2 倍（渣机只有两个核，没有办法），因此，使用并行化确实能够提高代码的运行速度

测试集结果如下表：

| 随机森林大小 | 10 | 50 | 500 |
|---|---|---|---|
| 随机样本数 | 6238 | 6238 | 6238 |

| | | | |
|---|---|---|---|
| 分裂阈值个数 | 5 | 5 | 5 |
| 随机属性值 | 24 | 24 | 24 |
| 预测准确率 | 75.20% | 88.87% | 91．32% |

| | | | |
|---|---|---|---|
| 随机森林大小 | 50 | 50 | 50 |
| 随机样本数 | 6238 | 6238 | 6238 |
| 分裂阈值个数 | 5 | 10 | 20 |
| 随机属性值 | 24 | 24 | 24 |
| 预测准确率 | 85.52% | 86.77% | 86.90% |

通过上述实验可以发现，随着决策树的增长，预测的准确度会不断提高，但是分裂阈值的增加却不会对预测准确度提高有太大影响

## 五．测试心得及体会

通过这次实验，我对随机森林算法有了进一步的理解，对于并行化实验有了比较深刻的认识。数据挖掘，作为一门极其热门和和具有良好发展前景的学科，在大数据处理方面有着独特的优势。但是，数据的增长远远快过算法的优化速度，并行化成为我们无法回避并且折中的考虑，基于并行化，能够使庞大的数据实现分布式运行，大大加快运行速度。因此，MPI，OpenMP，Spark（Dpark）都具有非常大的前景。本次实验中，我使用的多进程的方式进行并行化，实现了加速的效果。但是，由于我使用的是 python 进行编写的，算法不能够得到很好的优化，自己在优化方面做得还不够，以后要尝试去做 Spark 多机化并行以及使用 C++重写并行化算法

总之，通过这次实验，获益良多，希望自己以后也能够有机会参与大数据的处理及优化问题，真正让数据挖掘这门学科发挥价值