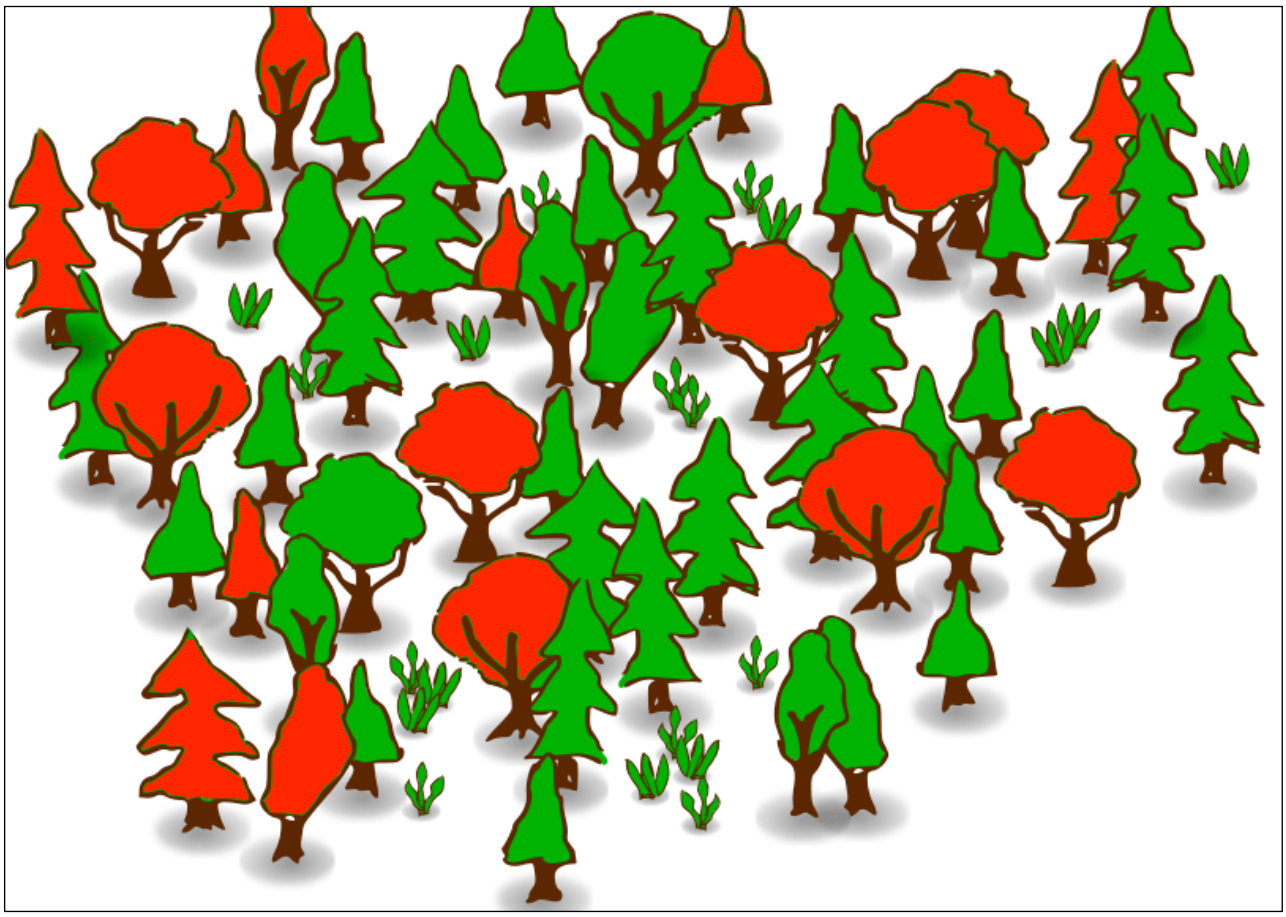
RandomForest

使用MPI进行并行化构建随机森林

谢志旺 - 12330346 - 2015-06-11



问题分析

数据可视化

首先对数据进行可视化，观察数据特征的分布情况，以选择合适的构造决策树的算法。在这里我通过python 代码对数据进行可视化。通过随机选择 10 个特征，将特征值作为横坐标值，label 值作为纵坐标值分

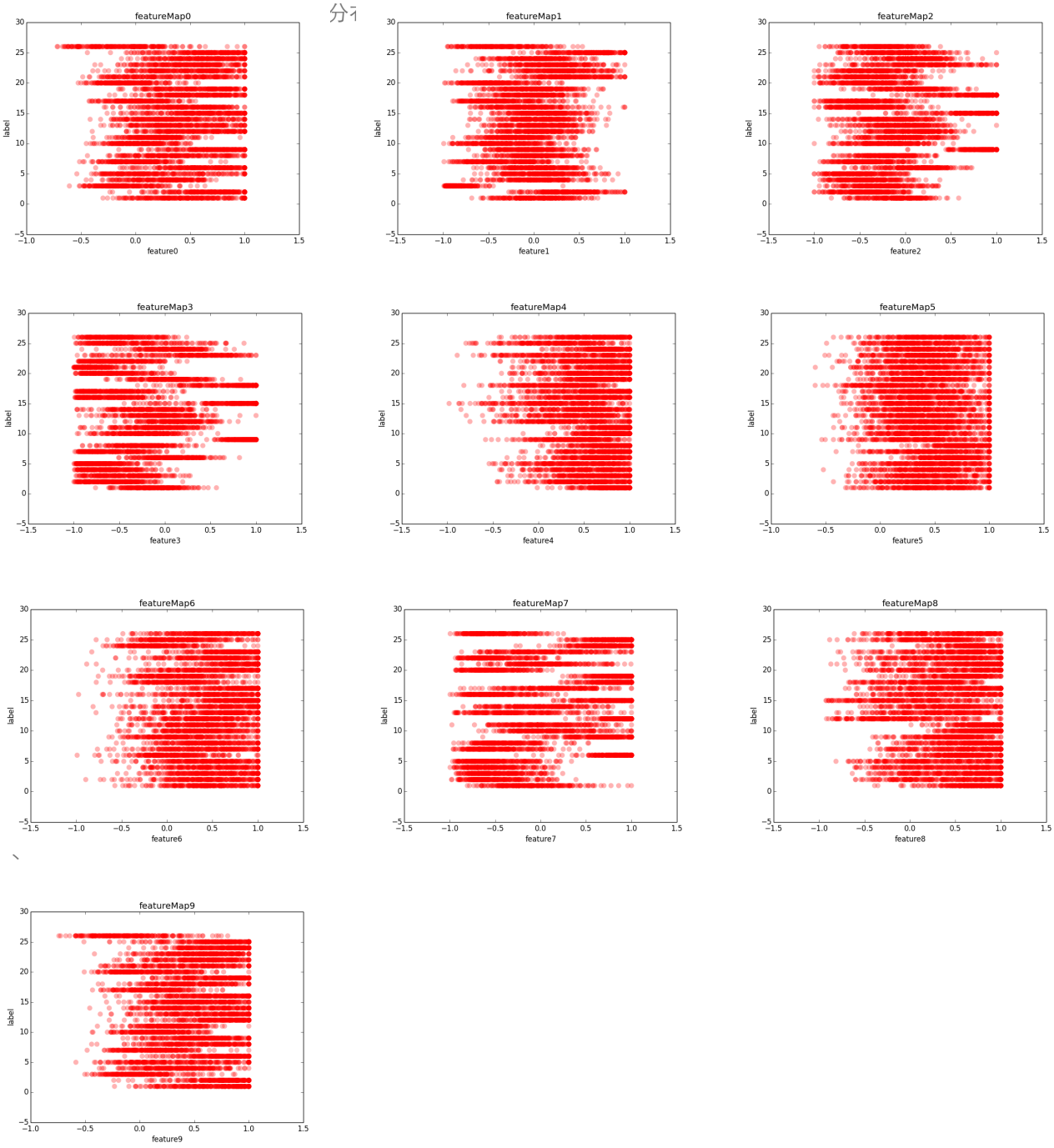


Fig1: 数据集可视化

可以看出，特征是连续的。所以在构建决策树的时候需要使用连续特征的算法。通过数据描述已知 label 是离散的，一共 26 类，所以采用CART分类树。

数据分析

根据数据可视化的结果选择 CART 分类树作为随机森林的树进行随机森林的构建。因为数据是 TA 根据卷积神经网络从声音资源提取的特征，所以目前的特征已经可以进行处理，暂时不考虑对特征的提取。

算法分析与设计

CART分类树的构建

CART 分类树的构建中，主要有两个特点：一是根据 gini 增益进行分割特征的选择，而是分类树在每个节点只进行二分。所以到最后的分类树是一棵二叉树。构建 CART 树有两个需要注意的地方，这两个地方是降低建树时间的关键：

- CART 分类树一般使用递归法进行构建，但是递归会造成建树时间很长。所以这里我采用的方法是迭代建树。使用一个栈来模拟深搜，在这个过程中进行树的构建。
- 在对每个特征列查找最佳分割点的时候，需要在特征值的每个突变点进行 gini 增益的计算，如果直接计算的话，这里的时间复杂度将是 O(n^3)。这个时间复杂度明显太高。所以在我的实现中，首先对特征值进行排序，这样遍历一遍就可以完成所有突变点的处理。另一方面是，我使用两个 map 变量（left，right）来保存突变值两边的特征值所对应的 label 的数量（也就是 left 保存小于突变值的特征值对应的 label 数量，right 保存大于突变值的特征值对应的 label 数量）。于是在遇到新的特征值时，只需要交换 left 与 right 在突变值之间的 label 值的对应数量，而不用遍历整个特征列。

上面两个点是降低程序执行时间的关键。而因为特征值是连续的，所以在每次根据特征值进行数据集的分割时，并不删除特征列，因为可能该特征列在后面的树节点分割是还是最佳分割特征。

下面是 CART 树的构建算法：

输入：训练数据集 D，停止训练条件

输出：构件好的 CART 决策树

1. 设节点的训练数据集为 D，计算现有特征对该数据集的基尼系数，对每个特征列，根据特征列的每个突变值，计算根据该特征值进行分割后的数据集的 gini 系数和 gini 增益。遍历特征列的突变值，找到该特征的最小 gini 增益以及对应的分割值。下面是 gini 系数和 gini 增益的计算公式：

   Gini 系数：$gini = \sum_{i=1}^{m} \left(1 - p_i^2\right)$

   Gini 增益：$giniGain = \sum_{i=1}^{N} \frac{m_i}{N} gini_i$

2. 遍历所有特征，找到所有特征中 gini 增益最小的那个特征以及其分割值。然后将数据集根据这个分割值分割成两个数据集。以生成原节点的左右两个子节点。
3. 对左右两个子节点重复 1、2 步，直到满足停止条件为止。

在构建 CART 分类树的时候，为了建树方便，我在树的每个节点保存了自己的数据集。这个数据集在完成建树后就不再需要了，因为随机森林需要建多棵树，为了内存使用的效率，在每次建完树后都会将树的每个节点保存的数据集清空。

随机森林的构建

随机森林由许多的决策树组成，树之间是没有关联的，当测试数据进入随机森林时，其实就是让每一颗决策树进行分类，最后取所有决策树中分类结果最多的那类为最终的结果。也就是说，随机森林就是一个包含多棵决策树的分类器。随机森林有如下几个特点：

- 有放回的随机取样：在训练一棵决策树的时候，随机森林是通过有放回随机抽样的方式选择训练的数据集。
- 随机选择部分特征列：训练一个决策树的时候并不是选择所有特征列，而是随机不放回地随机抽样部分特征。抽样的数目一般远小于特征列总数。
- 决策树完全分裂：根据前两个方式选择的数据集具有很大的随机性，树与树之间的关联也会降到最低。因此建树的时候即使不进行剪枝也不会造成过拟合。

前两个取样方法合称为 bootstrap 法。

下面是随机森林的构建算法：

输入：训练数据集 D，树的数量

输出：构件好的随机森林

1. 对数据集 D，应用 bootstrap 法抽取训练数据集，用这个数据集训练一个决策树模型。
2. 重复步骤 1 直到构建了足够数量的决策树。

算法的并行化

MPI

MPI(Message Passing Interface)是于1994年5月发布的一种消息传递接口。它是一个库，是一种标准或规范的代表。另外，MPI是一个消息传递编程模型，它提供与C和Fortran语言的绑定。
在本程序使用的是 MPI 提供的进程间通信机制，通过进程之间的消息传递来完成整个随机森林的构建。

构建并行化随机森林

在本程序中，利用 MPI 提供的进程间通信机制，采用主从控制的方式来进行随机森林的构建。
首先是控制进程。在程序中选择选择 rank 值为 0 的进程作为控制进程，它的任务是监控其余执行进程的工作。控制进程不进行具体的建树和预测操作，它只是进行任务分发和结果收集的操作。第二是执行进程。在程序中，除了 rank 值为 0 的进程之外的都是执行进程。执行进程的任务就是接收控制进程分发的任务并将结果返回给控制进程。在我的实现中，有两个特点如下：

- 每个执行进程不会将建好的树发送到控制进程，而是各自保存。这样做有两个原因，一是 MPI 要进行自定义的数据结构的发送非常麻烦，必须首先确定数据结构的大小。但是因为随机森林是完全分裂建树，所以很难确定最终树的大小。另一个原因是，建好随机森林进行预测的时候，控制进程需要执行进程进行预测工作，如果这个时候全部建好的树是保存在控制进程，那么就需要再次将树发送到执行进程。这个通信过程完全是资源的浪费。基于这两个原因，每个执行进程各自保存建树结果。
- 在进行预测的时候，每个执行进程独立进行预测工作，预测完成后将结果发送到控制进程，由控制进程进行最终结果的选择。

在使用 MPI 进行随机森林的并行化的时候，需要注意进程间的同步，也就是在进入下一步工作之前需要全部进程都准备好。

算法的优化

特征选取

特征选取一般有两种方法，平均不纯度减少（mean decrease impurity）和平均精确率减少（Mean decrease accuracy）。

平均不纯度减少（mean decrease impurity）

随机森林由多个决策树构成。决策树中的每一个节点都是关于某个特征的条件，为的是将数据集按照不同的响应变量一分为二。利用不纯度可以确定节点（最优条件），对于分类问题，通常采用基尼不纯度或者信息增益，对于回归问题，通常采用的是方差或者最小二乘拟合。当训练决策树的时候，可以计算出每个特征减少了多少树的不纯度。对于一个决策树森林来说，可以算出每个特征平均减少了多少不纯度，并把它平均减少的不纯度作为特征选择的值。

平均精确率减少（Mean decrease accuracy）

直接度量每个特征对模型精确率的影响。主要思路是打乱每个特征的特征值顺序，并且度量顺序变动对模型的精确率的影响。很明显，对于不重要的变量来说，打乱顺序对模型的精确率影响不会太大，但是对于重要的变量来说，打乱顺序就会降低模型的精确率。

平均不纯度减少是在选择最佳分割特征时用到的方法。这里主要说一下我的代码里面对**平均精确率减少**的方法的使用。步骤如下：

- 首先是根据源数据集训练一个模型，并用这个模型对测试数据集进行测试，得到一个标准错误率。
- 对每个特征，随机改变该特征的特征值在数据集中的顺序，然后再用这个新的数据集训练一个新的模型，用这个新的模型用测试数据集进行测试，得到一个新的错误率，然后看这个错误率与标准错误率的距离（差值的绝对值大小）。
- 上面两步能得到每个特征的特征值顺序随机改变后得到的模型与标准模型的测试误差的距离大小。距离大说明该特征对模型的影响大，反之亦然。那么就可以将距离小于一定阈值，也就是对模型的影响小到一定程度的特征从数据集删除。

这样得到的数据集比源数据集小，特征也更为相关。训练速度会变快，结果也会更准确。

特征值合并

特征值合并首先是随机选择一些特征列进行合并，比如值相加，相乘等，然后用合并后的数据集进行准确度测试，看特征列的合并对模型的影响。我主要采用了两种合并方式：

- 对每棵树都进行一次特征列合并，合并比例为特征列比例。

- 所有树将相同下标的特征列合并，其中特征列的选择是随机的。

试验结果分析

以下的测试都采用测试数据集。首先将训练数据集随机分割为 trainFin.csv 和 testCV.csv 两部分， 比例为 7:3。然后用 trainFin.csv 训练一个随机森林，用 testCV.csv 数据集进行测试。

并行化前后时间的对比

构建树的数目 / 棵	数据集比例	特征列比例	并行化前 / s	并行化后 / s	执行速度提升倍数
50	0.8	0.15	448	113	4.0
100	0.8	0.15	911	202	4.5

从表格可以看出，并行化后的执行速度基本上有 4~5 倍的提升，随着建树的数量增加速度提升会越来越明显。在我的程序中没有使用 openMP 等其他并行化技术，这是因为经过查看，在使用 MPI 的时候计算机的所有核都已经满负荷运行，计算机内存基本处于使用了 47% 的状态。因为我是一台计算机跑的，所有就资源利用率来说基本上已经用到了，没有必要再使用 openMP 技术。

运行时间截图（50棵树）：

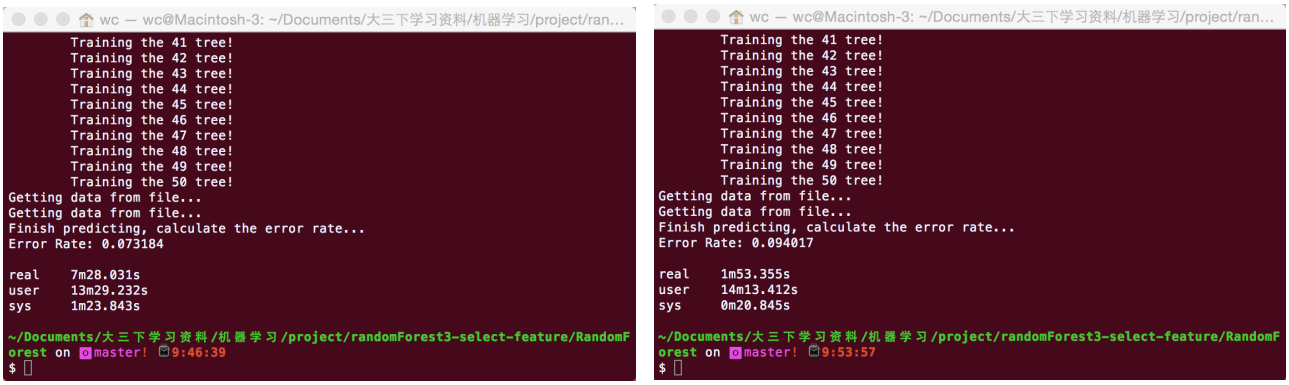


Fig2：并行前后运行时间截图（左：并行化前，右：并行化后）

计算机运行状态（11个进程）：

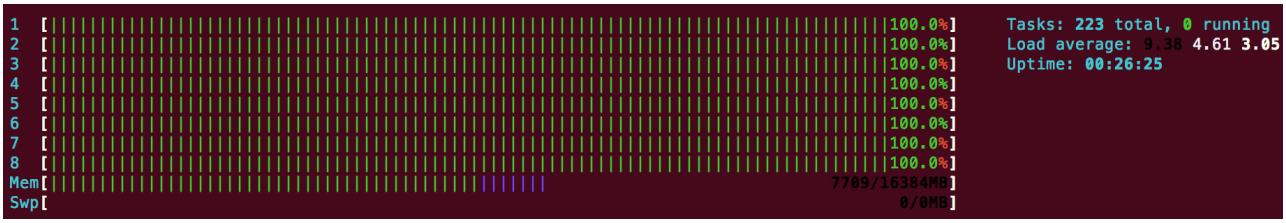


Fig3：设置为 11 个进程并行跑程序时计算机的运行状态

树的数量对结果的影响

树的数量	**50**	**100**	**500**	**1000**
数据集比例	0.8	0.8	0.8	0.8
特征列比例	0.15	0.15	0.15	0.15
分类准确度	91.5%	92.1%	94.3%	94.5%
运行时间 / s	115	202	1133	2305

从表格可以看出，树的增加对准确度有一定程度上的提升，但是随着树的增加对准确度的提升程度也在逐渐变小。基本上在 500 棵树以后就没有提升了。但是增加树的数量会造成运行时间的增加。因此 500 棵树是比较好的选择。

数据集比例对结果的影响

树的数量	100	100	100	100
数据集比例	**0.3**	**0.6**	**0.8**	**1**
特征列比例	0.15	0.15	0.15	0.15
分类准确度	91.4%	92.4%	93.4%	93.6%
运行时间 / s	82	172	168	276

从表格可以看出，数据集比例的增加对于分类的准确度来说有一定程度的提升，尤其当数据集从很小开始增大的时候。但是当数据集大小比例为 0.8 开始，增大数据集对分类的准确率基本没有多大的提升了。这样看来，0.8 比例的数据集是比较合适的。

特征列比例对结果的影响

树的数量	100	100	100	100
数据集比例	0.8	0.8	0.8	0.8
特征列比例	**0.1**	**0.15**	**0.3**	**0.5**
分类准确度	91.9%	92.1%	92.7%	92.3%
运行时间 / s	141	202	643	1441

从表格可以看出，增加特征列比例在一定范围内可以增加分类准确率。在特征列比例小于 0.5 的时候，增加特征列比例都有一定作用。但是在特征列增加到一定程度后，准确率反而会降低，这是因为树之间的数据集的关联度增加了，造成了过拟合。因此，选择特征列比例为 0.15 较为适合。

综上所述，随机森林的合适参数为：

数据集比例：0.8

特征列比例：0.15

树的数量：　500

测试的平均准确率：

在下面的测试中，进程数为 11，数据集比例为 0.8，特征列比例为 0.15，树的数量为 500. 多次训练的结果如下：

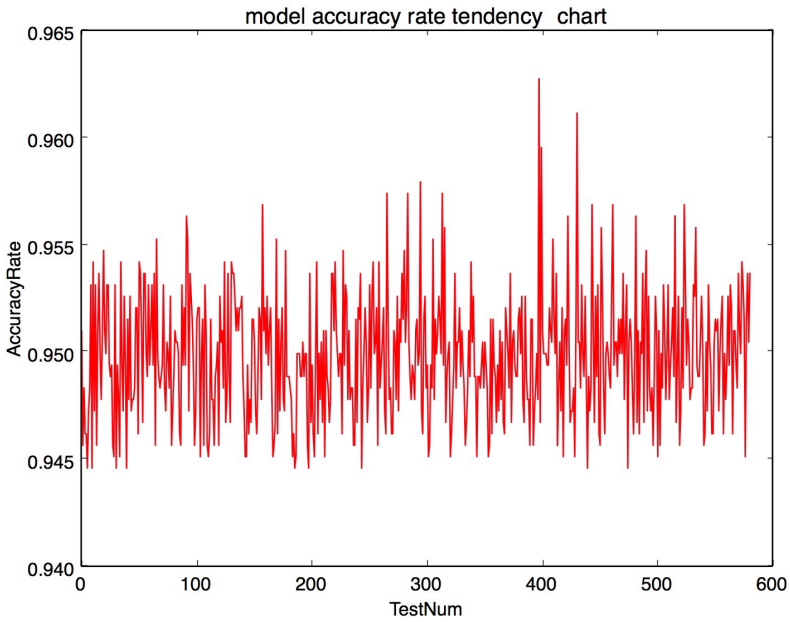


Fig4：多轮测试的模型准确度变化

经过多轮测试得到的平均准确度为 95%。

在以上的所有测试中，计算机配置如下：

处理器：2.3GHz Intel Core i7

内存：16GB 1600 MHz DDR3

操作系统：OS X Yosemite

算法优化结果分析

本程序我采用了 5 种方法进行结果优化，其效果分别描述如下：

- 组合随机森林：组合随机森林是建立多个随机森林模型，然后利用混淆矩阵来得到结果。经过测试，效果提升不明显，基本上有 0.5% 的分类准确率的提升
- 特征选择：这个方法在前面的特征选择部分有进行描述。测试结果是对模型有一定提升，但是这种方法的运行时间太长，因为要对每个特征列进行乱序后建模。而且因为建树的时候的数据集是随机选择的，所以这个方法的结果也有一定的随机性。
- 用回归树代替分类树：这种方法需要控制树的层数。我在每棵树的数据集中数据数和特征数都设置为原数据集的 0.5 倍。但是测试结果来看准确度基本只有 70%。
- 在树的每个分裂点随机选择特征列：这种方法是每棵树不再是在建树前就随机选择特征，而是在每个节点分裂的时候随机选择一定比例的特征列进行求 gini 增益的计算，然后从这些特征列选择分割特征。经过测试，这种方法会使得程序执行时间增加，但是准确度并没有什么变化。
- 随机选择特征列进行合并：这个方法在前面有描述。使用这种方法稳定性太差，在某些情况下分类准确率会有很大提升（到 97%），但是其他情况可能又下降到 94%。这个方法可以用来选择特征，也就是找

到可以使得分类准确率提高的特征列来建模型。但这个方法是个好的思路，利用随机合并特征列来找到某些有较强关联的特征列，可以在最终模型上提高准确率。

本程序的重点是对随机森林的并行化，我写了两种版本，第一种是控制进程完全控制一切，执行进程建好的树全部要发送到控制进程，这样就需要控制树的大小，因为 MPI 发送自定义数据类型需要提前设定，所以我将树的层数控制在7层以内。但是这种方法得到的随机森林的准确率基本只有 60%。经过检查发现是分类树的限制太大的原因。在前文说过，随机森林因为选择训练样本的随机性，所以已经可以很大程度上避免过拟合，这里的层数限制反而造成了欠拟合。

第二种方案是前面描述的最终并行化方法，也就是每个执行进程各自保存训练好的分类树。这种方法因为不需要进行决策树的传递，一方面降低了性能损耗，另一方面因为不需要控制分类树的大小，分类树持续分类到不能再分类为止。这个方案达到的效果较佳，在 CV 测试集上的准确度基本上在 95% 左右。

另一方面需要考虑的就是到底是对一棵树的创建并行化还是多个进程并行化独立建树。因为在随机森林中的每棵分类树是独立的，所有多个进程并行化独立建树可以减少建树过程中的消息传递开销，所以选择这种方式进行建树。

在对随机森林的分类准确度提升方面，我所做的工作有算法优化结果分析里面描述的 5 中方法。在提交的代码中我只附上了特征选择和随机选择特征列进行合并部分的代码。对于组合随机森林、用回归树代替分类树及在树的每个分裂点随机选择特征列这三类方法，经过测试基本没有多大效果，所以我直接去掉了这部分代码。可以通过控制变量来控制是否需要进行特征列合并。另外，程序里面提供的随机选择特征列进行合并是特征值合并中的第二类方法。

总结

这个项目是我做的第一份分布式机器学习算法。刚开始我的实现是用 redis + python 来实现多机器分布式的。但是一个方面因为是递归建树，然后在寻找最佳分裂特征时也没有进行算法优化，再加上 python 本身的执行速度也稍有欠缺，所以执行之间非常长。后来听到老师的建议后，改用 c++，非递归建树，查找最佳分裂特征优化以及使用 MPI 框架来实现并行化。最终效果的提升明显。

老师介绍是说师兄的实现中使用回归树，最终实现的准确度有 98%，但是我按照老师介绍的方法，实际上的效果却非常差，但是给老师发邮件询问老师并没有回复我。而我自己找到的方法到最后最高都只能到 97%。有点遗憾。希望师兄可以给我发一份邮件介绍一下你的实现，非常感谢。

因为尝试各类的分类准确率提升方法，所以整个项目花的时间比较长。因为时间的关系，没有再实现多机器的并行。后面打算尝试的是算法本身实现线程的并行，然后加上多台机器并行执行，应该可以较大程度提升执行速度。