中山大學
SUN YAT-SEN UNIVERSITY

本 科 生 毕 业 论 文

题　　目：大规模机器学习算法的并行实现

院　　系：软件学院

专　　业：软件工程（计算机应用）

学生姓名：盘振坚

学　　号：10389441

指导教师：潘炎（副教授）

二〇一四 年 三 月

摘　要

随着互联网的普及，业界数据开始了大规模的积累，在各行各业中逐渐形成了这样的一个共识，谁能把这些数据充分利用起来，谁就有机会成为这个领域的佼佼者，掌握先机，引领这个行业的发展。

而机器学习算法，就是把这些行业数据利用起来的利器，同时，机器学习算法所能处理的数据规模，就是评价这个算法优劣的重要标准。在机器学习算法中，决策树方法经过前人的不懈努力，在其可行性方面得到充分的理论证明。其中，Boosting 方法是决策树方法中使用最广泛同时也是效果最好的一个，同时，在 Boosting 算法的实现过程，有一些步骤（例如建树过程的节点选取）在并行情况下会有更好的时间性能。Boosting 的并行化，就是处理业界大规模数据的利剑。

本论文首先描述 Boosting 算法单线程版的原理与实现，并通过一些列序列图介绍算法的核心流程，然后使用 MPI 对 Boosting 算法单线程版进行改进，并实现 Boosting 的并行化。最后，分别使用 Boosting 单线程版与 Boosting 并行版处理几个数据集，并用图表呈现二者性能的差异。

关键词：　1；Boosting　2；并行化　3；机器学习

Abstract

As the popularity of Internet, the data of various domains have accumulated rapidly. A common sense rises in various domains that the winner who will get the opportunity is who can make good use of such data.

The solution of situlation above is the machine learning algorithm which will be the important inditator of one algorithm. Among the machine algorithms, the feasibiity of tree methods have been proved, and Boosting Method is the most effective method of them, which is pretty suit to parallelize. The Boosting with parallelization is the best method for big data.

This paper describes the simple thread implementation version of Boosting algorithm firstly and explains the principle of process. Secondly it conveys how to improve the Boosting algorithm from simple thread verson to parallelization version through MPI. Finally, it displays the performance of two version by chart with several datasets.

Keywords: 1; Boosting 2; Parallelization 3; Machine Learning

目　录

第一章 前言

1.1 项目的背景和意义

随着业界数据的积累，现在越来越重视如何把这些数据利用起来。大规模机器学习算法的流行也是出现在这样的互联网环境下。为了提高算法的精准度，必须通过收集更多的样本，升维等做法，但是这些做法往往是极其费时的。所以，我们需要通过并行化，让一个算法运行于多台机器上，这样子便能减少时间的使用。

1.2 研究开发现状分析

在目前业界比较流行的几大机器学习算法中，随机森林，Boosting，深度学习是应用最多的几种算法。随机森林[10][11]是采用其随机性来对样本进行预测，从而降低过拟合，但是，随机性是一把双刃剑，在有效避免过拟合的同时，也带来了一系列训练模型的不稳定性，这是随机森林的最大不足之处；深度学习是目前比较新的机器学习算法，在图像与音频方面的效果很好，但是，里面缺乏必要的严格证明，在选择参数方面有一定的盲目性。

而 Boosting 是树方法的一种，也是比较成熟的机器学习算法，树方法中比较著名的还有有 CART[1]和 C4.5[5]，还有 Boosting 的改进版 RGF[6]。虽然相比深度学习，树方法效果没那么好，但是，树方法在理论方面相对比较成熟，在选参方面亦有相当多的理论证明。而且，Boosting[7][8][9]算法在实现中有一些步骤注定了其转换成并行版的可能性，通过并行现实来提升 Boosting 算法在处理大规模数据时的能力与效果。

1.3 论文结构简介

论文共有六章，分别为：

第一章是前言。

简述机器学习算法的现状，并行化的意义，以及论文的结构简介。

第二章是技术与原理。

从理论角度介绍 Boosting 并行算法的每一个应用到的技术，并结合参考文献给

出相应的数学公式。本论文实现的 Boosting 并行算法主要基于三大技术：CART 决策树作为基本学习机，Boosting 方法，MPI 的使用。

第三章是模块设计。

从模块角度介绍 Boosting 算法的并行实现，首先全局介绍并行算法的整体框架，然后介绍模块之间的交互以及各模块的内部实现逻辑。其中，该算法分成三大模块：数据读取模块，特征提取模块，训练与预测模块。

第四章是单线程版实现。

本章介绍的是 Boosting 算法的单线程版实现，首先介绍一下算法的整体流程：读取数据——训练——保存 Model——预测。然后深入讲解每一步，介绍每一个步骤的内在逻辑。

第五章是并行化。

首先通过一个简单的例子介绍 MPI 并行化的主从模式，然后介绍如何把第四章所描述的 Boosting 算法单线程版并行化。

第六章是结论。

第二章 技术与原理

2.1 CART

CART（classification and regression tree[1], 分类与回归树），该模型由 Breiman 等人在 1984 年提出，是应用广泛的决策树学习方法。CART 决策树算法主要分成三个步骤：特征选择，树的生成，剪枝。

CART 决策树算法既可以用于分类问题，也可以用于回归问题。CART 是在给定输入随机变量 X 条件下输出随机变量 Y 的条件概率分布的学习方法。CART 假设决策树是二叉树，内部结点特征的取值为“是”和“否”，左分支是取值为“是”的分支，右分支是取值为“否”的分支。这样的决策树等价于递归地二分每个特征，将输入空间即特征空间划分为有限个单元，并在这些单元上确定预测的概率分布，也就是在输入特定的条件下输出的条件概率分布[2]。

在使用决策树学习方法时，绝不能忽略的问题是——如何避免过拟合。CART 决策树方法采用的剪枝过程来避免过拟合。在本论文中，使用 Boosting 提升方法来取代 CART 决策树的剪枝过程来避免过拟合。Boosting 提升方法会在下一节（2.2 节）再作介绍，接下来，介绍一下本论文所涉及的 CART 决策树的两个过程：特征选择和树的生成。

2.1.1 特征选择

CART 决策树方法的特征选择过程，就是选择哪一个特征作为决策结点的过程。特征选择在于选取对训练数据具有分类能力的特征，这样可以提高决策树学习的效率。

CART 在处理分类问题和回归问题中，对于每一次最优特征的选取，采用的是不同的评判准则：分类问题中采用基尼指数（Gimi index）最小化，回归问题采用平方误差（Square error）最小化。

2.1.2 树的生成

即建树过程，对于每一个节点的选择时，都要从特征中选取出一个 VAR 或 GINI

最小的所对应的特征切分点对<f, sp>作为节点，已选取作为树节点的特征不能被重复选取。递归改过程，直到决策树建成。

2.2 Boosting

Boosting，又名提升方法，是一种常用的统计学习方法，应用广泛且有效，在分类问题中，它通过改变训练样本的权重，学习多个分类器，并将这些分类器进行线性组合，提高分类的性能[2]。

提升方法基于这样一种思想：对于一个复杂任务来说，将多个专家的判断进行适当的综合所得出的判断，要比其中任何一个专家单独的判断好。实际上，就是“三个臭皮匠顶个诸葛亮”的道理。

Boosting 是要把多个弱学习器组合成一个强学习器，即把已有的“弱学习算法”提升（boost）为“强学习算法”。“弱学习算法”与“强学习算法”来源与 Kearns 和 Valiant 所提出的两个概念：“弱可学习（weakly learning）”和“强可学习（strongly learnable）”。在概率近似正确（probably approximately correct，PAC）学习的框架中，一个概念（一个类），如果存在一个多项式的学习算法能够学习它，并且正确率很高，那么就称这个概念是强可学习的；一个概念，如果存在一个多项式的学习算法能够学习它，学习的正确率仅比随机猜测略好，那么就称这个概念时弱可学习的。后来 Schapire 证明强可学习与弱可学习时等价的，也就是说，在 PAC 学习框架下，一个概念时强可学习的充分必要条件是这个概念时弱可学习的[2]。以上的证明为 boosting 的可行性提供理论依据。

2.2.1 Boosting 模型

Boosting 提升方法实际采用加法模型（即基函数的线性组合）与前向分步算法。在本论文中，采用 CART 作为 boosting 的基函数$T(x;\ \Theta)$

$$\mathrm{f}_M(\mathrm{x}) = \sum_{m=1}^{M} T(x;\ \Theta_m) \tag{2.1}$$

其中，Θ_m为 CART 的参数；M 为基函数 CART 的个数。

2.2.2 Boosting 算法

Boosting 算法采用前分步算法。初始基函数$f_0(x) = 0$，第 m 步的模型是

$$f_m(x) = f_{m-1}(x) + T(x; \Theta_m) \tag{2.2}$$

其中，$f_{m-1}(x)$为当前模型，通过最小化误差 L()来确定下一个基函数的参数Θ_m，

$$\Theta_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \tag{2.3}$$

2.3 MPI

MPI (Message Passing Interface) 是一个跨语言的通讯协议，用于编写并行计算算法，支持点对点和广播；是一个消息传递应用程序接口，包括协议和语义说明，他们指明如何在各种实现中发挥其特性。MPI 的目标是高性能，大规模性，和可移植性。

同时，MPI 也是一个由众多并行计算机厂商、软件开发单位/组织、并行应用单位等共同维护的标准，自 1994 年发布以来，其标准经历了一次版本升级，从 MPI-1 升到了 MPI-2。MPI-1 实现了基本的消息通信操作，组通信操作，虚拟拓扑、可拓展数据类型管理等。MPI-2 增加了外部接口，拓张的组通信操作，并行 I/O，单向通信，进程管理，线程控制等，增加了域间通信能力[4]。

本论文所实现的 Boosting 并行算法通过 C++语言调用 MPI 接口，所采用 MPI 实现版本是 openmpi，运行环境是 ubuntu，可以通过 apt-get 配置 openmpi 并行计算开发环境。

2.3.1 MPI 的简单使用

Hello World

```cpp
#include <iostream>
#include <mpi.h>
using namespace std;

int main( int argv, char* argc[] )
{
        MPI_Init( &argv, &argc );
        cout << "Hello World!" << endl;
        MPI_Finalize();

        return 0;
}
```

图 2-1：Hello World 代码实例

如上代码实例，在 MPI_Init()与 MPI_Finalize()之间的代码会被并行化，实际上，这两者之间的代码会被复制多份，并由多个线程同时执行。

键入以下指令：

mpicxx hello.cpp –o hello

编译 hello.cpp

mpirun –n 4 hello

运行 hello，其中，参数 -n 4 代表启动 4 个线程运行 hello，输出如下

```
Hello World!
Hello World!
Hello World!
Hello World!
```

图 2-2：Hello World 输出样例

当我们开启 4 个线程运行 hello 时，操作系统其实会在 4 个不同的线程里执行同一份代码，如图 2-3 所示：

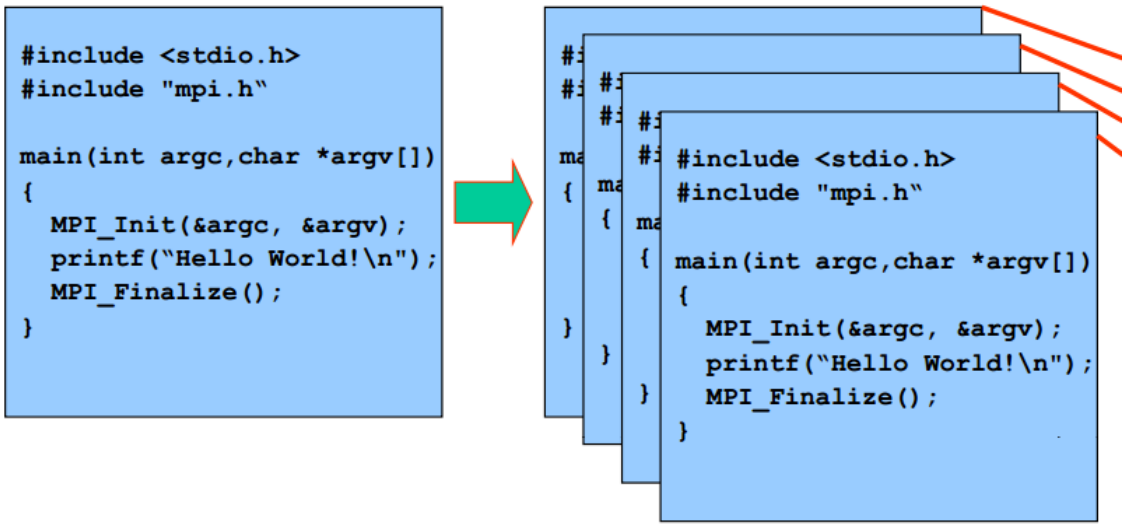

图 2-3：MPI 运行原理

各个线程所执行的代码是一样的，只有通过不同的线程号区分不同的线程，同时也是通过这个线程号来传递消息的，关于如何获得该线程号，将会在下一节（2.3.2）提到。

2.3.2 MPI 的简单消息传递

目前两种最重要的并行编程模型是数据并行和消息传递，数据并行编程模型的编程级别比较高，编程相对简单，但它仅适用于数据并行问题；消息传递编程模型的编程级别相对较低，但消息传递编程模型可以有更广泛的应用范围[4]。

MPI 属于消息传递编程模型，其各个滨兴执行的部分之间通过传递消息来交换信息、协调步伐，控制执行。消息称传递一般是面向分布式内存的，但是它也可适用于共享内存的并行机。消息传递为编程者提供了灵活的控制手段和表达并行的方法，一些用数据并行很难表达的并行算法，都可以用消息传递模型来实现，灵活性和控制手段的多样化，是消息传递并行程序能提供高的执行效率的重要原因。

MPI 的消息传递主要分成两种类型：点对点通信，广播通信。这里，通过两个简单的例子，分别来介绍一下 MPI 两种消息传递的使用，如下（图 2-4）点对点通信的代码例子：

```cpp
#include <iostream>
#include <mpi.h>
using namespace std;

int main( int argc, char *argv[] )
{
        int rank;
        char msg[20];
        MPI_Status status;

        MPI_Init( &argc, &argv );
        MPI_Comm_rank( MPI_COMM_WORLD, &rank );
        if ( 0 == rank ) {
                strcpy( msg, "Hello, process 1" );
                MPI_Send( msg, strlen(msg)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD );
        }
        else {
                MPI_Recv( msg, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status );
                cout << "received: " << msg << endl;
        }

        MPI_Finalize();
        return 0;
}
```

图 2-4：点对点通信样例代码

用 mpicxx 编译，并启动 2 个线程运动该程序得到如下结果（图 2-5）：

```
received: Hello, process 1
```

图 2-5：点对点通信输出

本例子实现的是 process 0 想 process 1 发送一个字符串“Hello, process 1”，process 2 接收到字符串并将其显示出来。

其中主要用到了几个 MPI 的接口：

1. MPI_Comm_rank(MPI_COMM_WORLD, &rank);
2. MPI_Send(msg, strlen(msg)+1, MPI_CHAR, 1, 99, MPI_COMM_WORLD);
3. MPI_Recv(msg, 20, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);

MPI_Comm_rank()，是获取当前线程的线程号，其中 MPI_COMM_WORLD 是线程域，只有在同一线程域中的各线程才能通信，没有特殊指定的话，所创建的线程都会放在 MPI_COMM_WORLD 这个默认线程域中。

MPI_Send()，是传递消息，前三个参数是信件，后三个参数是信封。信件里三个参数分别是所要发送消息的储存地址，长度，数据类型；信封里三个参数分别是目标线程号，Tag，线程域，其中 Tag 可以用来标识该条传递的信息。

MPI_Recv()，是接收信息，与 MPI_Send 相似，前三个参数是信件，后三个参数是信封。信件里三个参数分别是所接收消息要储存的地址，长度，数据类型；信封里三个参数分别是源线程号，Tag，线程域。

接下来，介绍一下 MPI 的广播通信，如下代码样例（图 2-6）：

```
#include <iostream>
#include <mpi.h>
using namespace std;

int main( int argc, char* argv[] )
{
	int rank, value;

	MPI_Init( &argc, &argv );
	MPI_Comm_rank( MPI_COMM_WORLD, &rank );
	do {
		if ( rank == 0 );
			cin >> value;

		MPI_Bcast( &value, 1, MPI_INT, 0, MPI_COMM_WORLD );
		cout << "Process " << rank << " got " << value << endl;
	} while ( value >= 0 );
	MPI_Finalize();

	return 0;
}
```

图 2-6：广播通信样例代码

用 mpicxx 编译，并启动 5 个线程运动该程序得到如下结果（图 2-7）：

图 2-7：广播通信输出

本例子主线程读入一个数字，然后广播出去，同一线程域中所有线程（包括主线程自身）收到该信息，并输出到终端。

该例子主要用到一个 MPI 的接口:

MPI_Bcast(&value, 1, MPI_INT, 0, MPI_COMM_WORLD);

MPI_Bcast()，共需要 5 个参数，其中前三个是信件，后两个是信封。该接口与 MPI_Send()有点相似，不同之处主要有两点：(1) MPI_Send()只用用于发送消息，而 MPI_Bcast()既可以用来发送消息，又可以用来接收消息。对于信件里的参数，当发送消息时，第一个参数是所要发送消息储存的地址位置，当接收消息时，第一个参数是所接收消息存放的地址位置，第 2，3 个参数与 MPI_Send()相同，分别是长度和数据类型。(2) 信封部分，MPI_Bcast 只需要两个参数：源线程号和线程域，无论用于发送还是用于接收，都是需要源线程号，而不需要 Tag 参数。

第三章 模块设计

本论文采用 C++语言实现 Boosting 算法的并行化，本章介绍 Boosting 算法的模块设计。

3.1 整体框架

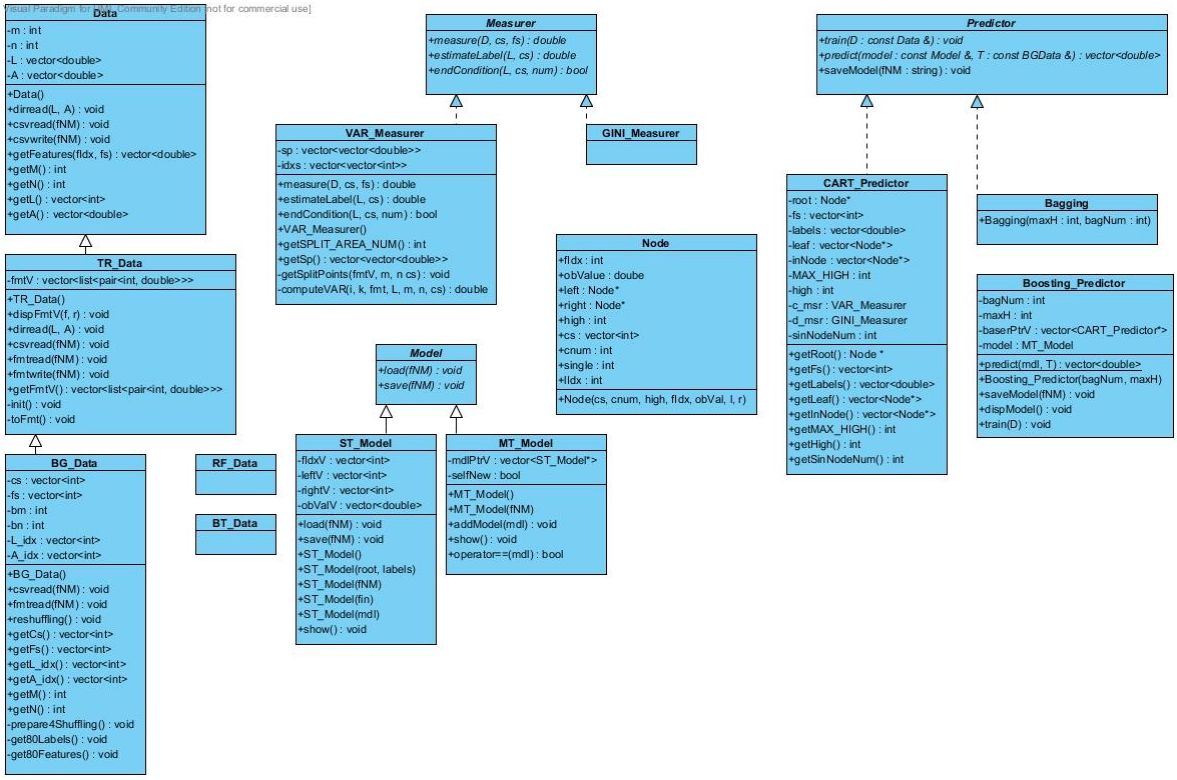

图 3-1：Boosting 单线程版整体框架

Boosting 算法的单线程版实现主要分成三大模块：数据读取模块，特征评估模块，训练与预测器模块。

3.2 数据读取模块

这一部分的职责主要有 3 个：

1. 从文本读入并保存数据，文本可以使通用的.csv 文件，也可以是方便树状模型计算的数据集格式.fmt 文件。
2. 对读进来的数据进行适当的处理，方便作为训练机的输入。
3. 把数据写入文件，同时也可以完成从.csv 到.fmt 文件的转换

该模块主要有两层继承关系，Data 和 TR_Data;

其中，Data 是基类，实现.csv 文件数据的读写功能，TR_Data 类拓展了 Data 类，采用了一种更加适合树状模型的存储格式.fmt，并对格式提供了相应的与.csv 转换方法。

3.3 特征评估模块

特征评估模块有两个主要的特征评估器，VAR_Measurer 和 GINI_Measurer，前者用于解决回归问题时的特征评估，后者用于解决分类问题时的特征评估；二者都继承于纯虚类 Measurer，纯虚类 Measurer 定义了三个纯虚方法：measure()，estimateLabel()，endCondition()。

measure()：特征评估的主要功能函数，大部分计算量的核心模块所在，同时也是下一步要并行化以降低时间复杂度的地方，其返回被评估特征的一个定量的数据结构 MS，见图 3-2，fIdx 是特征下标；obVal 是切分点；msVal 是特征选取的衡量值，如回归问题使用 VAR（方差），分类问题使用 GINI 值，二者都是对一个特征被选取作为本次建树的结点是否恰当的一个定量的评估。

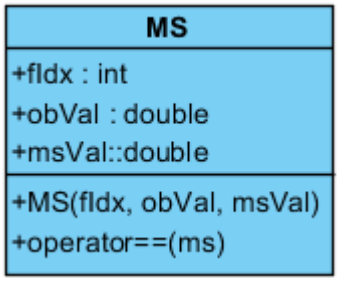

图 3-2：MS 类

estimateLabel()：建树完成之后，对叶子预测值的一个计算，如回归问题是被分到该叶子节点上所有样本所对应预测值的均值，分类问题是取被分到该叶子节点上比例最高的那一个类别。

endCondition()：建树过程中，判断一个节点是否应该作为叶子节点的条件。

3.4 训练与预测模块

该模块是整个 Boosting 单线程版算法的核心，该模块里面有两个主要的类 CART_Predictor 和 Boosting_Predictor，分别是 Boosting 算法的基本学习机和主要训

练预测机。为了该项目的可拓展性，二者都继承自纯虚基类 Predictor，该基类定义了三个纯虚函数：train()，predict()，saveModel()，其中 predict()是静态函数，只要有了 model（模型），就可以直接通过类名调用 predict()方法对数据进行预测。

train()：用来通过训练集训练出用来预测的 model（模型），该过程比较耗时，model（模型）一旦训练完成，可以重复使用。这里的 model（模型）定义了一个 Model 类来存在相关的信息与对应的方法，见图 3-3。

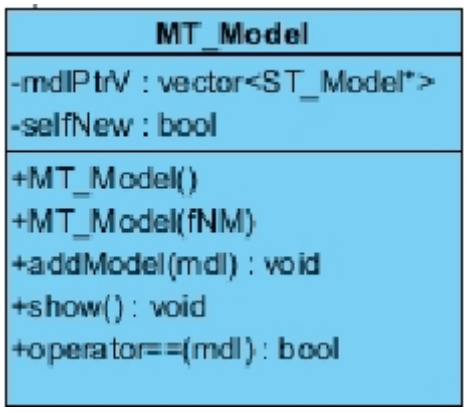

图 3-3：Model 类

predict()：利用训练出来的 model（模型）来对数据进行预测

saveModel()：保存训练出来的 model（模型）到文件，文件后缀为.mdl。

第四章 单线程版实现

本章首先介绍 Boosting 算法的基本思路，通过其单线程版的实现来从细节上介绍 Boosting 的每一实现步骤。接下来，举个简单的例子来说明一下特征评估选取过程。

现有如下小型训练数据集（5 个样本，每个样本 4 个特征），储存在.csv 文件：

	L	F1	F2	F3	F4
C1	1	0	0	0.5	0.4
C2	1	0	0	0.5	0.5
C3	1	0.5	0	0.5	0
C4	2	0.7	0.5	0.5	0.8
C5	2	0.9	0.5	0.5	0.9

图 3-4：事例数据集

4.1 Boosting 算法整体流程

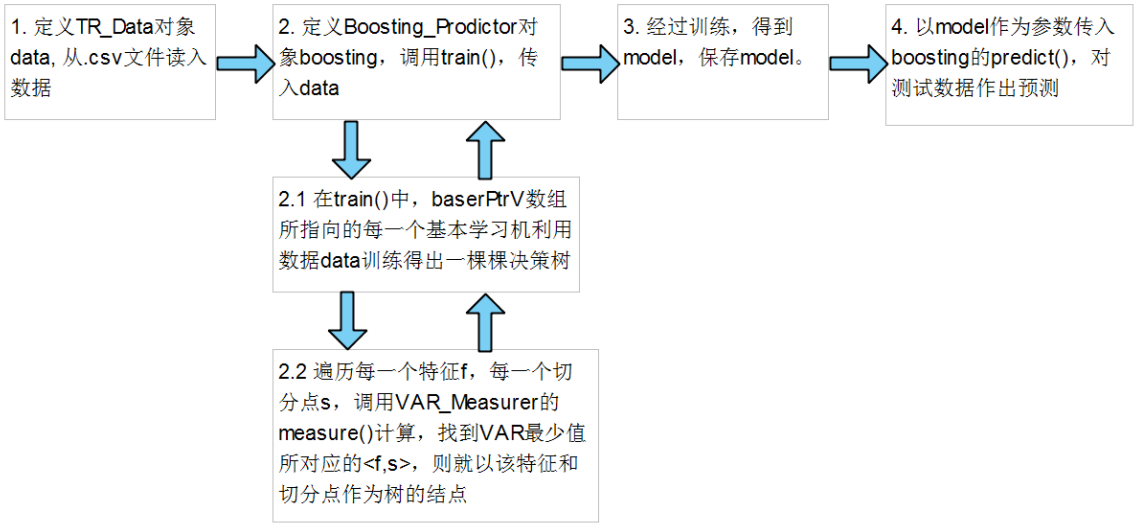

图 4-1：Boosting 算法整体流程

```
1. void pro1Test( int bagNum, int maxH=10 )
2. {
3.     TR_Data D;
4.     D.csvread( "dataset/pro1.csv" );
5.
6.     Boosting_Predictor bt( bagNum, maxH );
7.     bt.train( D );
8.
9.     bt.dispModel();
10.    bt.saveModel( "model/pro1.bmmdl" );
11.
12.    vector<double> p;
13.    MT_Model mdl( "model/pro1.bmmdl" );
14.    p = Boosting_Predictor::predict( mdl, D );
15.    double RMSE = rmse( p, D.getL() );
16.
17.    cout << RMSE << endl;
18. }
```

图 4-2：Boosting 算法调用代码样例

图 4-1 是 Boosting 单线程版算法的整体流程，图 4-2 是 Boosting 算法调用代码样例。

4.2 第一步，数据读取

如图 4-2 的第 3，4 行，实例化 TR_Data，调用实例方法 csvread()，从 pro1.csv 中把训练集数据读进来。

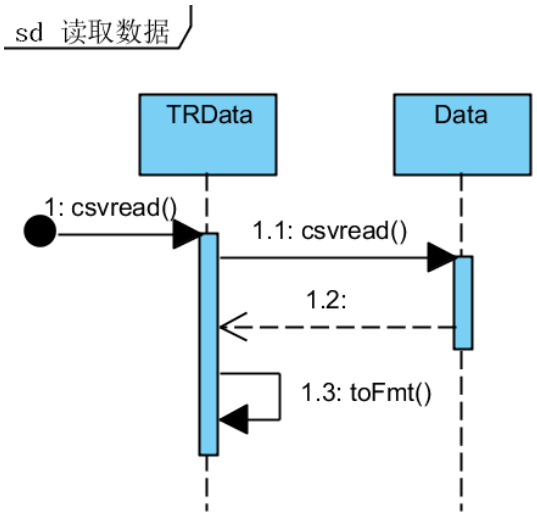

图 4-3：读取数据的序列图

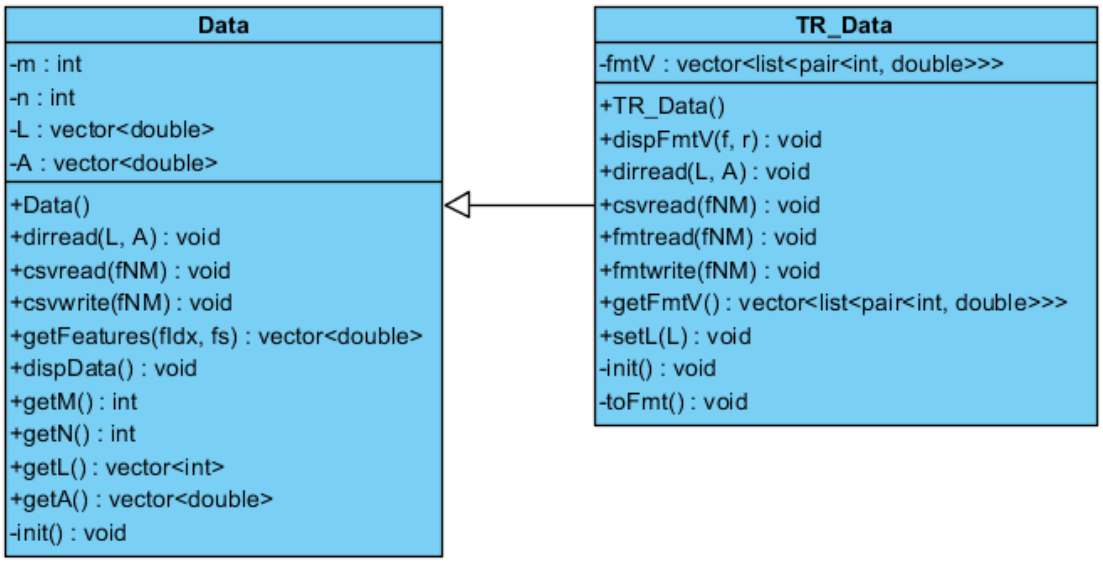

图 4-4：Data 与 TR_Data 的类图

4.2.1 数据集的存储方式

如图 4-3，调用 TR_Data 对象的 csvread()方法之后，TR_Data 对象会把训练集储存成以下数据：

int m	样本数
int n	特征数
vector<double> L	经验值，m x 1
vector<double> A	特征矩阵，m x n
vector<list<pair<int,double>>> fmtV	便于计算的 fmt 格式

图 4-5：数据集的存储方式

4.2.2 数据集的两种格式

本算法支持训练数据集有两种格式：.csv 格式和.fmt 格式。如图 4-6，其中.csv 格式是通用的数据集格式，每一个数据用逗号隔开；.fmt 是本算法自定义的格式，是一种方便本算法特征评估的一种存储格式。

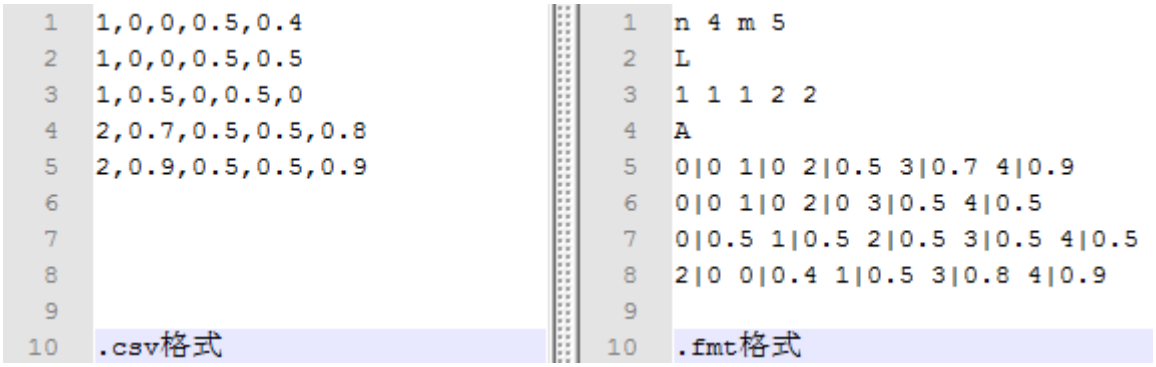

图 4-6：数据集的两种格式

从图 4-6 两种不同格式的对比可以看出，.fmt 格式存储了比.csv 更多的信息，其中，第 1 行的 n 与 m 分别是特征数与样本数，第 2，3 行的 L 代表训练集的经验值，第 4~10 行是 A，代表训练集的特征值，这里的特征值与.csv 的特征值部分不一样，这里每一行是训练集的一个特征，每一行的每个元素以“样本序号|相应特征值”来记录，每一行是按照特征值来排序的，而不是根据样本序号来排序的，这样做的目的是方便在下一节（4.3）所提到的特征评估过程中需找该特征的切分点。

4.3 第二步，训练

如图 4-2 的第 6，7 行，给出基本学习机个数 bagNum 与树的最大高度 maxH 实例化 Boosting_Predictor，调用实例方法 train()并传入 TR_Data 对象，开始训练。

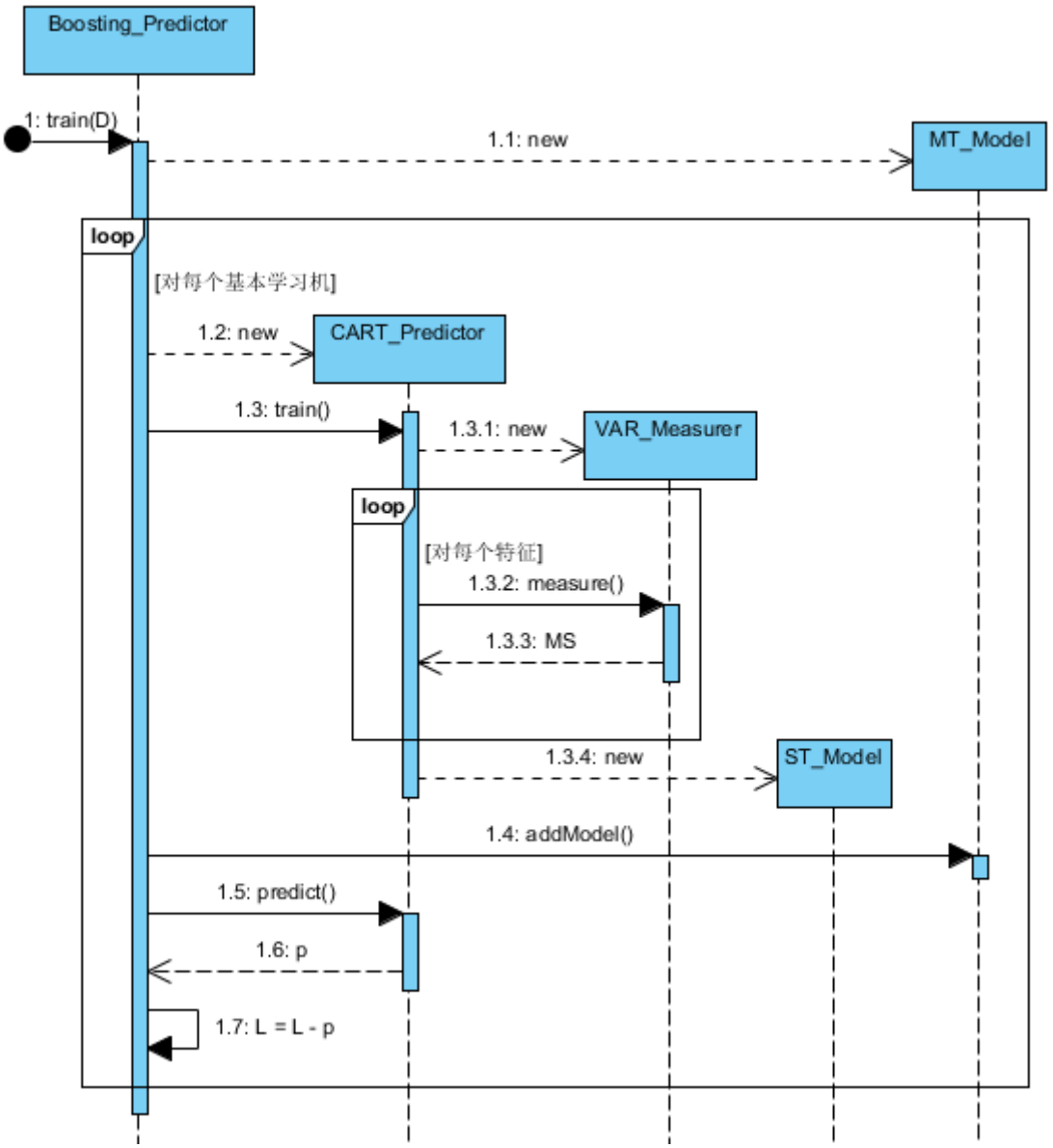

图 4-7：训练过程的序列图

4.3.1 基本学习机

如图 4-7，调用 CART_Predictor 的 train()方法之后，首先会新建一个 MT_Model 对象，然后根据 CART_Predictor 对象里有用户指定的 bagNum（基本学习机个数）开始循环，新建 bagNum 个基本学习机，即 CART，记录在 CART_Predictor 对象的 baserPtrV 属性里，如图 4-8。

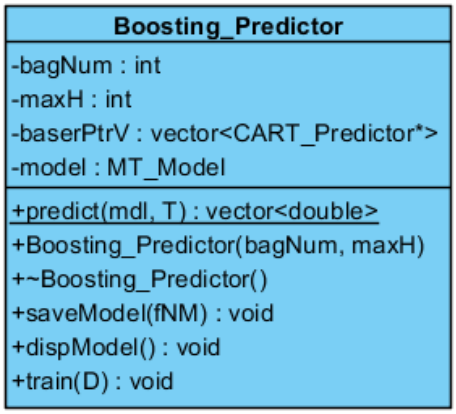

图 4-8：Boosting_Predictor 类图

对于每一个基本学习机 CART，都根据所输入的数据集进行训练，训练过程即 CART 的建树过程，该过程每新增一个节点，都要经过特征评估过程，该过程正是本算法计算量的所在，也是本算法的难度所在，亦是下一步要进行并行化的地方，特征评估的具体思路，将会在下一节（4.3.2）详细介绍。

经过一个个特征评估过程选好节点建好一棵 CART 树之后，就会把该得到一个模型（ST_Model），然后使用该模型对训练集进行预测，得到预测值 p，求出预测值与真实值之差（L-p），作为下一个基本学习机 CART 建树的经验值。如此循环下去，直到建立好 bagNum 棵 CART 树之后，就会得到 bagNum 个 ST_Model，合在一起，就是 Boosting_Predictor 的得到训练出来的模型（MT_Model）了。

4.3.2 特征评估

接下来，详细介绍一下 CART 建树过程选取每一个节点时所必须的特征评估过程，特征评估主要是通过最小化<特征，切分点>与该训练数据的 VAR 值（回归问题）或 GINI 值（分类问题），来选取出最佳的<特征，切分点>作为 CART 建树的节点。

接下来，通过介绍处理分类问题时 VAR 的计算，来详细介绍一下特则评估这个过程。如图 4-9 是调用 measure()时的序列图：

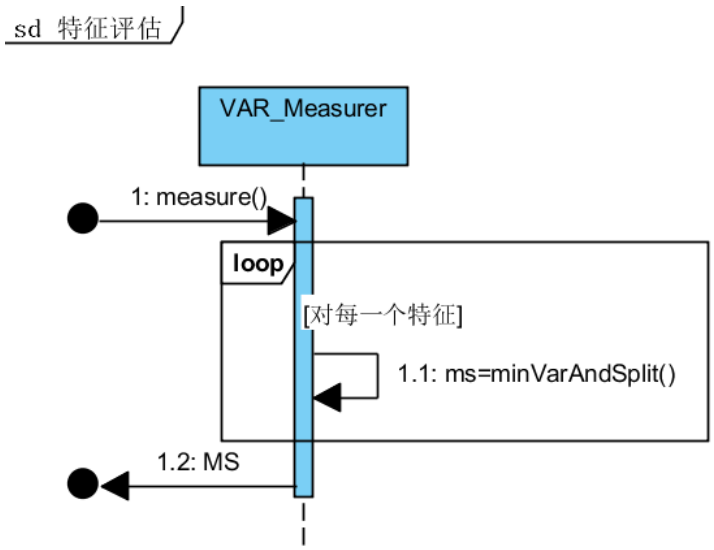

图 4-9：调用 measure()时的序列图

图 4-10 的 VAR_Measurer 的内部详细结构：

```
class VAR_Measurer: public Measurer
{
public:
    // implement virtual methods
    MS measure( const TR_Data &D, const vector<int> &cs, const vector<int> &fs );
    double estimateLabel( const vector<double> &L, const vector<int> &cs );
    bool endCondition( const vector<double> &L, vector<int> cs, int num );

    // own methods
    VAR_Measurer();              // constructor for accurater
    const vector<int>& getPart1() const { return this->part1; }
    const vector<int>& getPart2() const { return this->part2; }

private:
    MS minVarAndSp( const vector<double> L, const list< pair<int,double> > F, double sqSums,
double sums, int nums, const vector<int> cs );
    double computeVAR( double sqSums, double sums, int nums, double sum1, int num1);

    // res
    int minFIdx;
    double minSpVal;
    double minVAR;
    int m1;
    int m2;
    vector<int> part1;
    vector<int> part2;
};
```

图 4-10：VAR_Measurer 的内部详细结构

在 VAR_Measurer 内部，采用了一些技巧来提高时间的效率。众所周知，对回归问题时，建决策树过程是根据选取 VAR 最小的一组<特征，切分点>对来作为决策树

上的一个结点的。每一组<特征，切分点>的 VAR 的计算公式是：

$$VAR(f,s)=\frac{m_1}{M}v(X_1)+\frac{m_2}{M}v(X_2)$$

(4.1)

其中 f 是特征，s 是切分点，M 是样本总数，X_1和X_2是样本被切分点 s 分成的两部分，m_1是X_1所含有的样本数，m_2是X_1所含有的样本数，$\mathrm{v}(X_1)$是计算X_1样本的方差，$\mathrm{v}(X_2)$是计算X_2样本的方差。

此处方差的计算公式是

$$v(X)=\frac{1}{m}\sum_{i=1}^{m}(x_i-\bar{x})^2$$

(4.2)

其中 m 是被分配到该堆样本数，x_i是 X 中的一个样本，$\bar{x}$是 X 的均值，$\bar{x}=\frac{1}{m}\sum_{i=1}^{m}x_i$。

这个方差计算公式平方后再求和，其中存在着一些重复的计算，现在对这个公式进行处理，有

$$v(X)=\frac{1}{m}\sum_{i=1}^{m}{x_i}^2-\left(\frac{1}{m}\sum_{i=1}^{m}x_i\right)^2$$

(4.3)

易知，(4.2)和(4.3)是等价的。将(4.3)代入(4.1)得：

$$VAR(f,s)=\frac{1}{M}\left[\sum_{i=1}^{M}{x_i}^2-\frac{1}{m_1}\left(\sum_{i=1}^{m_1}x_i\right)^2-\frac{1}{m_2}\left(\sum_{i=1}^{m_2}x_i\right)^2\right]$$

(4.4)

此处，优化所采用的基本思路是：把那些需要大量重复计算的先算好并存储起来。这样子的话，当后面的步骤需要再次使用这些结果时，就可以直接使用，节省了计算量。

首先，我们就必须先找到哪些结果是我们后面步骤将会大量重复用到的。见图 4-11：

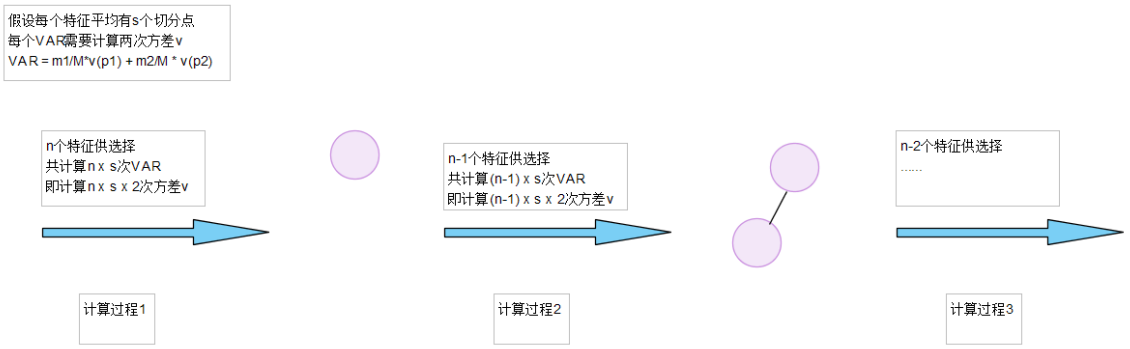

图 4-11：VAR 重复计算分析

对计算过程 1，共需要计算 $n \times s$ 次 VAR，根据公式(4.4)，每一次 VAR 计算可以分成三部分：

（1）$\sum_{i=1}^{M} x_i^2$

（2）$\frac{1}{m_1}\left(\sum_{i=1}^{m_1} x_i\right)^2$

（3）$\frac{1}{m_2}\left(\sum_{i=1}^{m_2} x_i\right)^2$

对每一个特征 f，第（1）部分都是相同的，这个结果对于同一个特征 f 但不同切分点 s 的计算时，可以保留下来以避免重复的计算量。而第（2）（3）部分则根据不同的切分点 s 而不同，故该两部分要根据不同切分点 s 计算。

在本算法实现时，第（2）（3）部分的计算在对每一个切分点 s 时在 minVarAndSp() 时计算。

4.4 第三步，保存 Model

如图 4-2 的第 9，10 行，可通过 Boosting_Predictor 对象的 dispModel()方法来向终端输出树状图来查看每一棵 CART 树；同时也可以通过 Boosting_Predictor 对象的 saveModel()方法来保存模型，以供下一次预测使用。

4.5 第四步，预测

如图 4-2 的第 14 行，调用 Boosting_Predictor 类方法的 predict()并传入 model 和测试数据集，便可以对测试数据集进行预测。

第五章 并行化

本章介绍如何使用 MPI 把上述 Boosting 单线程版改成并行版，使用 MPI 实现并行化，实质上是通过 MPI 所提供的消息传递机制，来协同各线程之间的工作，达到多个线程之间的协助，用线程数来换时间，从而降低原算法的时间开销。

MPI 并行程序主要分为两种模式：对等模式和主从模式。前者是各线程做同样的工作，而后者是由主线程分配工作给从线程做，等从线程完成工作之后，主线程回收计算的结果，再作下一步的处理。主从模式是使用得相对比较多，本论文 Boosting 的并行化使用的是主从模式。主从模式的工作原理如下：

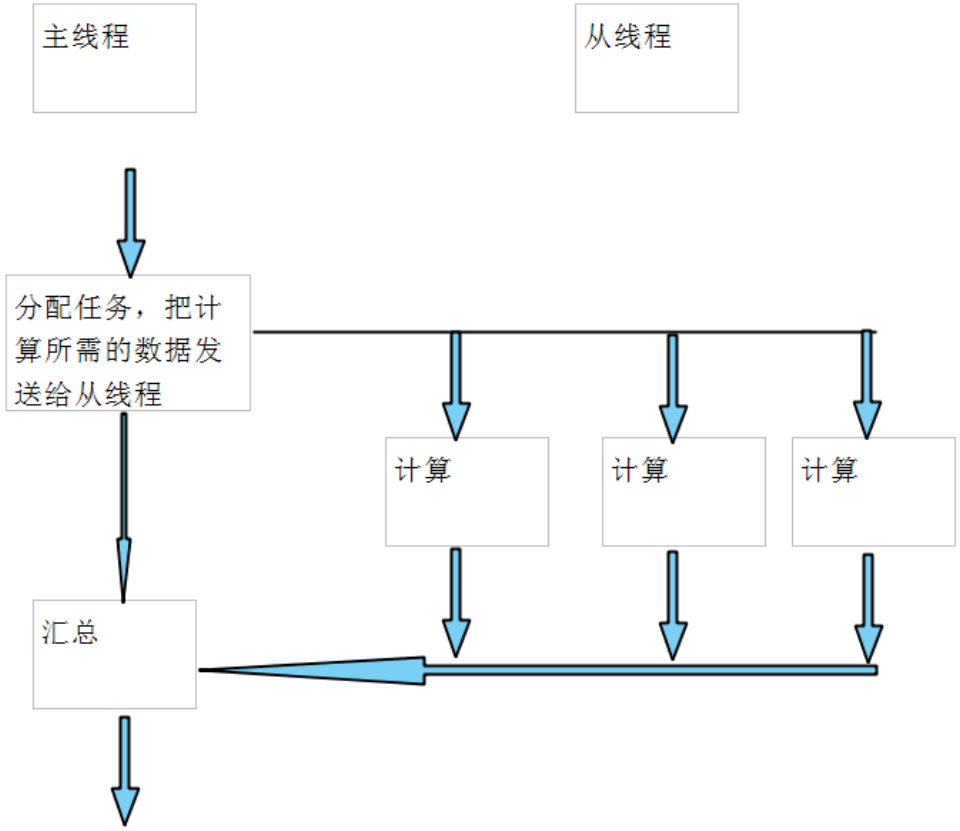

图 5-1：主从模式工作原理

首先，通过简单的例子来介绍如何使用 MPI 来实现主从模式。

5.1 并行化主从模式的简单例子

这个例子使用主从模式来实现 Ax=b，矩阵与向量的乘法：

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \tag{5.1}$$

以下是一个简单的使用 MPI 库并行实现上述矩阵与向量乘法的代码

```
#include <iostream>
#include <mpi.h>
using namespace std;

int main( int argc, char *argv[] )
{
    const int sz = 5;
    const int master = 0;

    int rank, numprocs;
    int numsent;
    int A[sz][sz], b[sz], c[sz], buffer[sz];
    //int ans, row;
    MPI_Status status;

    MPI_Init( &argc, &argv );
    MPI_Comm_rank( MPI_COMM_WORLD, &rank );
    MPI_Comm_size( MPI_COMM_WORLD, &numprocs );

    if ( master == rank ) {

        // stop the more thread
        if ( numprocs > sz+1 ) {
            //cout << "Too many Threads" << endl;
            for ( int i = sz+1; i < numprocs; i++ )
                MPI_Send( 0, 0, MPI_INT, i, 0, MPI_COMM_WORLD );
        }

        // init
        for ( int i = 0; i < sz; i++ ) {
            b[i] = 1;
            for ( int j = 0; j < sz; j++ )
                A[i][j] = i;
        }
        numsent = 0;

        // send b
        MPI_Bcast( b, sz, MPI_INT, master, MPI_COMM_WORLD );

        // send a each line
        for ( int i = 0; i < (numprocs-1<sz?numprocs-1:sz); i++ ) {
            MPI_Send( A[i], sz, MPI_INT, i+1, i+1, MPI_COMM_WORLD );
            numsent++;
        }

        // recived the result from each thread
        int ans;
        int sender, idx;
        for ( int i = 0; i < sz; i++ ) {
            MPI_Recv( &ans, 1, MPI_INT, MPI_ANY_SOURCE, MPI_ANY_TAG, MPI_COMM_WORLD, &status );
            sender = status.MPI_SOURCE;
            idx = status.MPI_TAG;
            c[idx-1] = ans;

            // if not finish send
            if ( numsent < sz ) {
                MPI_Send( A[numsent], sz, MPI_INT, sender, numsent+1, MPI_COMM_WORLD );
                numsent++;
            }
            else    // stop the thread
                MPI_Send( 0, 0, MPI_INT, sender, 0, MPI_COMM_WORLD );
        }

        // the result
        cout << "The result: ";
        for ( int j = 0; j < sz; j++ ) cout << c[j] << " ";
        cout << endl;
    }
```

```
    // not master thread
    else {
        // receive b
        MPI_Bcast( b, sz, MPI_INT, master, MPI_COMM_WORLD );

        // loop for receiving the line of A
        int buf[sz];
        while(1) {
            MPI_Recv( buf, sz, MPI_INT, master, MPI_ANY_TAG, MPI_COMM_WORLD, &status );

            if ( status.MPI_TAG != master ) {
                int row = status.MPI_TAG;
                int ans = 0;
                for ( int i = 0; i < sz; i++ ) ans += buf[i]*b[i];
                MPI_Send( &ans, 1, MPI_INT, master, row, MPI_COMM_WORLD );
            }
            else
                break;  // finish this thread
        }
    }

    MPI_Finalize();
    return 0;
}
```

图 5-2：矩阵向量乘法代码

要实现上述(5.1)矩阵 A 与向量 x 的乘法，需要求矩阵的每一行与向量 x 的内积，这个过程可以通过并行化来降低时间复杂度。具体过程如图 5-3，可分为三步：

1. 主线程把向量 x 广播给每一个线程，然后把每一行发送给每一个从线程
2. 从线程拿到这些数据之后，开始计算内积，计算完成后就把结果返回给主线程
3. 主线程汇总结果

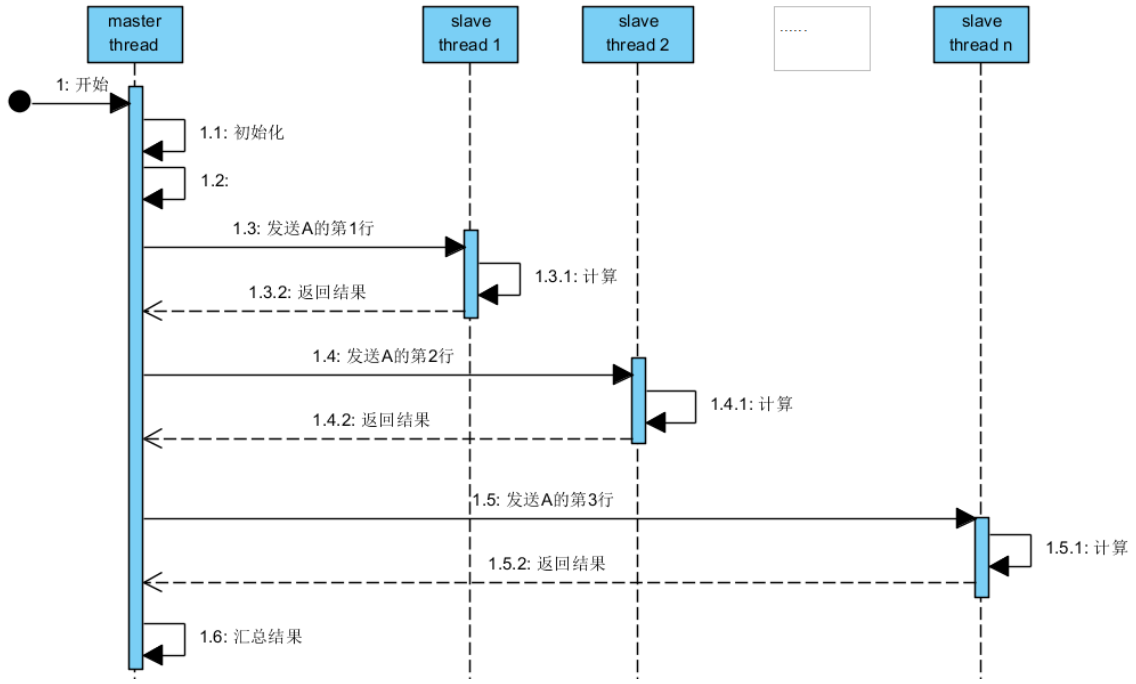

图 5-3：矩阵向量乘法具体流程

5.2 Boosting 算法并行化

Boosting 算法的并行化，主要针对的是 VAR_Measurer，上面（第四章 4.3.2 节）提到，VAR_Measurer 模块是在 CART 建树过程没加入一个节点就会调用一次 VAR_Measurer 的 measure()，而在 measure()里面，针对每一个特征，都会调用 minVar()来选取出最小 VAR 所对应的切分点 s，而并行化改进主要体现在 measure()里：由用户决定开始 n 个线程，其中，0 号线程为 master（主线程），1 ~ n-1 号线程为从线程，

具体步骤如图 5-4：

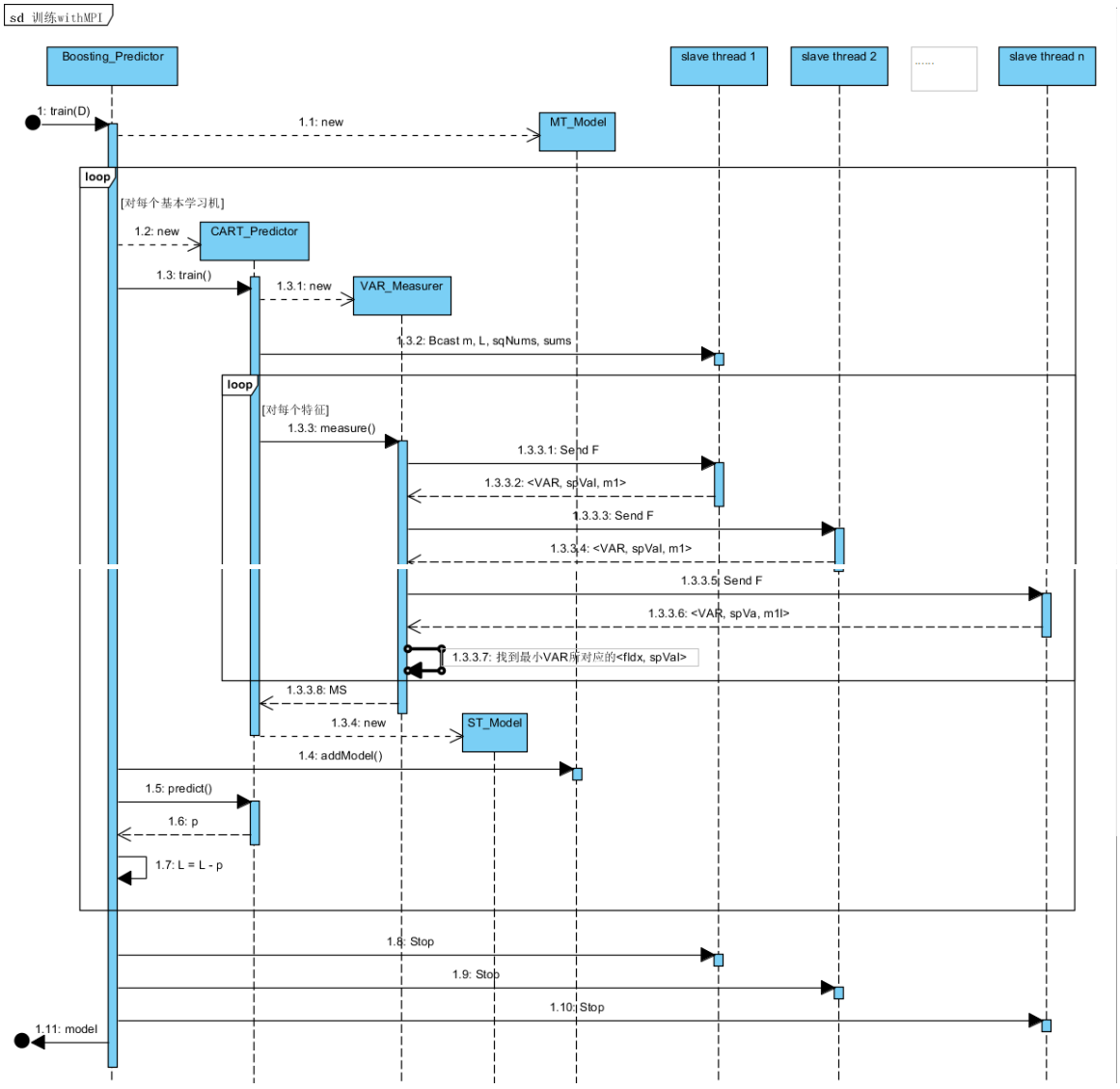

图 5-4：Boosting 并行算法具体流程

1. 主线程负责整体算法的推进
2. 当执行到 measure()时，把每一个特征（即每一列）的 minVar()过程，分配给从线程做。首先，master 会把 m, n, L, sqmuns（经验值的平方和），sums（经验值的和）广播给各从线程，然后，把对应该特征的一列发给某一个从线程。

3. 从线程负责计算并找出最小 VAR 所对应的切分点 s，并把结果返回给 master
4. 然后 master 汇总从线程所发来的结果，找出众结果中 VAR 最小的<f, s>
5. 用<f, s>作为建树的新节点

其中第 2 步中，用户所给定的从线程数往往会小于训练集的特征数，故在这里会涉及到如何对从线程的复用过程，即 master 会多次分配工作，从线程一完成目前的工作返回结果给 master 后，master 收到结果的同时，就再把还没计算的一列发给这个从线程，让其继续进行计算工作，直到每一列都计算完，且建树过程完成，就发送信息过去结束各从线程。

第六章 结论

回顾本次毕业设计的写作过程，我的代码实现过程是：CART——Bagging——RandomForest——Boosting，把树方法中典型的几种学习方法都实现一次，而且，在这基础上，逐渐把一些线性回归，逻辑式回归，k-mean 等学过的机器学习算法都一一实现，从而实现一个自己的机器学习算法库。在加深自己对理论理解的同时，也进一步扎实了自己对 C++的驾驭能力。

在实现过程中，我采用了 C++语言的继承机制，使用接口，抽象，设计模式等方法来保证目前算法库的可拓展性，以满足日后进一步把算法加进库中。同时，我采用了自己写的单元测试框架，来保证每一个模块的正确性，“攘外必先安内”只有保证了内部正确性才能一步步地往下开发新的模块，以实现我们所想要的功能。

同时，在写作过程中，由于需要涉及到算法的并行实现，我自学了 MPI 的使用以及并行程序设计的一些原则与模式，并应用到了毕业设计中。该过程我验证了自己学习一个新函数库的学习方法——Demo 法的正确性。

然而，Demo 法虽然可以让我根据“控制变量”以及“奥卡姆剃刀”原则，立马就可以上手一个新函数库，并正确使用它。但是，该方法的缺陷就是“后劲不足”，Demo 法只能让我们学到一些使用该函数库的表层知识，至于如何熟练使用，如何巧妙应用，如何深入了解其内部的原理，还是需要在后面花时间去深入阅读文档，去了解其真正的运作原理。

这次毕业设计的写作，使我在并行算法方面踏出了第一步，让我在思考如何解决问题时提供了一个更加广阔与有效的解决方法——并行化。

致谢

随着毕业设计的收尾，我的大学本科生涯也即将接近尾声，在毕业设计写作过程中，促使我反思了一下大学本科四年我的所学所得。

首先，我要感谢中山大学，这个即将成为我的母校的地方，在即将离开的最后这些日子里，我越发发现她的美，她的温柔。感谢中山大学为我提供的一个那么富有学习氛围，人文关怀，书香萦绕的校园环境，在这里，我启蒙，我成长。

其次，我要感谢我的导师，潘炎老师。他是一名负责任的老师，也是我大学本科里最喜欢的一位老师，我喜欢听他的课，喜欢看着他在黑板上一笔一画把每一条晦涩难懂的公式讲得清清楚楚的场景。从某种意义上，潘炎老师是我在机器学习数据挖掘方面的启蒙导师，是他让我感受机器学习算法的奇妙以及博大精深，让我对这个领域产生极大的兴趣，样本——训练——预测，这个机器学习的过程，也不禁潜移默化到了我的生活中解决实际问题的思路中。谢谢潘炎老师，谢谢您为我打开了机器学习这一扇不可思议的大门。

最后，我要感谢我身边的同学，朋友，是你们陪伴我度过大学这四年精彩的本科生活，你们所给我的回忆，将会在我的脑海里永久驻留。

参考文献

[1] L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. Classification and Regression Trees. Wadsworth Advanced Books and Software, Belmont, CA, 1984

[2] 李航，统计学习方法，北京：清华大学出版社，2012

[3] 都志辉，高性能计算之并行编程技术——MPI 并行程序设计，北京：清华大学出版社，2001

[4] 张武生、薛巍、李建江、郑伟民，MPI 并行程序设计实例教程，北京：清华大学出版社，2009

[5] J. Ross Quinlan. C4.5: Programs for Machine Learning. Morgan Kaufmann, 1993

[6] Rie Johnson, Tong Zhang. Leaning Nonlinear Functions Using Regularized Greedy Forest. 2012

[7] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an applicaiton to boosting. J. Comput. Syst. Sci., 55(1):119-139, 1997

[8] Jerome Friedman. Greedy function approximation: A gradient boosting machine. The Annals of Statistics, 29, 2001

[9] Rebort E. Schapire. The boosting approach to machine learning: An overview. Nonlinear Estimation and Calssification, 2003

[10] Leo Breiman. Bagging predictors. Machine Learning, 24:123-140, August 1996

[11] Leo Breiman. Random forests. Machine Learning, 45:5-32, 2001

毕业论文成绩评定记录

指导教师评语：   成绩评定： 指导教师签名：　　年　　月　　日
答辩小组或专业负责人意见：   成绩评定： 签名（章）：　　年　　月　　日
院系负责人意见：   成绩评定： 签名（章）：　　年　　月　　日

附表一、毕业论文开题报告

论文（设计）题目：

（简述选题的目的、思路、方法、相关支持条件及进度安排等）

学生签名：　　年　　月　　日

指导教师意见：

1、同意开题（ ） 2、修改后开题（ ） 3、重新开题（ ）

指导教师签名：　　年　　月　　日

附表二、毕业论文过程检查情况记录表

指导教师分阶段检查论文的进展情况（要求过程检查记录不少于 3 次）：

第 1 次检查

学生总结：

指导教师意见：

第 2 次检查

学生总结：

指导教师意见：

第 3 次检查

学生总结：

指导教师意见：

第 4 次检查

学生总结：

指导教师意见：

学生签名：　　年　　月　　日

指导教师签名：　　年　　月　　日

总体完成情况	指导教师意见： 1、按计划完成，完成情况优（ ） 2、按计划完成，完成情况良（ ） 3、基本按计划完成，完成情况合格（ ） 4、完成情况不合格（ ） 指导教师签名：　　年　　月　　日

附表三、毕业论文答辩情况

答辩人		专 业	
论文（设计）题目			
答辩小组成员			

答辩记录：

记录人签名：　　年　　月　　日

学术诚信声明

本人所呈交的毕业论文，是在导师的指导下，独立进行研究工作所取得的成果，所有数据、图片资料均真实可靠。除文中已经注明引用的内容外，本论文不包含任何其他人或集体已经发表或撰写过的作品或成果。对本论文的研究作出重要贡献的个人和集体，均已在文中以明确的方式标明。本毕业论文的知识产权归属于培养单位。本人完全意识到本声明的法律结果由本人承担。

本人签名：　　日期：