

# 随机森林基础算法实现

## 一、随机森林算法

算法实现了基于 CART(Classification And Regressio Tree)决策树的随机森林算法。随机森林算法，随机选取部分样本做进行单个决策树的训练，并且随机选取部分特征进行训练。

### 数据集

使用 Optical recognition of handwritten digits dataset 手写数字集作为训练随机森林的样本。将手写数字的不同位置的像素值作为离散特征输入到决策树的节点中。

### 节点分裂依据

采用 CART 进行二叉树分裂。采取 Gini 系数最小准则作为节点分裂阈值与特征选择的依据。Gini 系数使用以下公式进行计算。设置三种节点分裂的终止条件：一、当节点中所有样本均为同一类别；二、当节点中的样本个数少于设置的最小节点样本个数时；三、当节点的深度要求达到设置的最大深度。

$$Gini = P_{left} * (1 - \sum_{k=1}^{N_{left}} P_{left(k)}^2) + P_{right} * (1 - \sum_{k=1}^{N_{right}} P_{right(k)}^2)$$

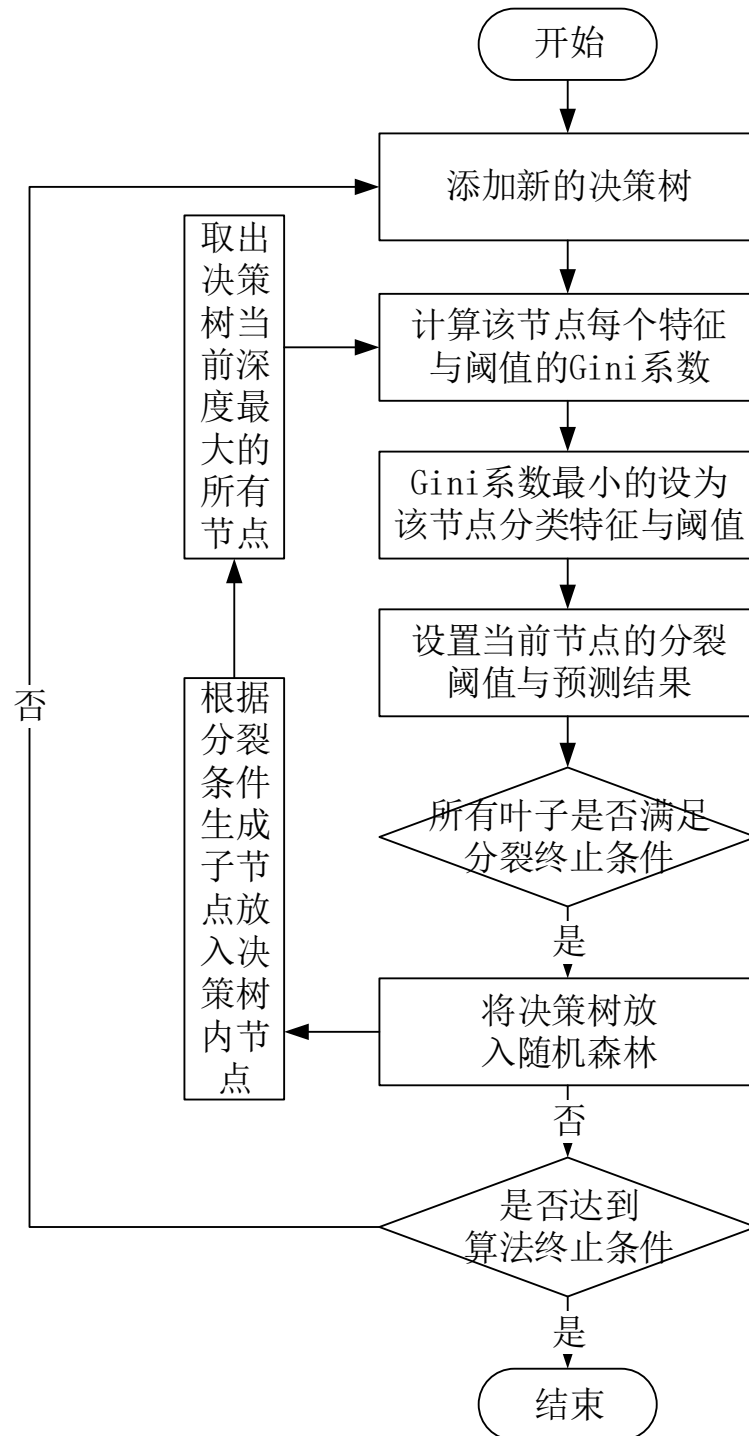
### 算法终止条件

设置两个终止条件：一、算法的精度要求达到设置的精度；二、算法的随机森林中的决策树个数达到设置的阈值上线。

### 预测结果产生

随机森林算法处理分类问题，采取统计每个决策树投票的方式，以得票数最高的分类结果作为最终的分类结果。

### 随机森林训练流程图



## 二、程序代码

### 编程语言与环境

在 ubuntu 下，借助于 cmake 工具组织使用 C++语言编程实现。算法程序仅使用 C++标准模板库 STL 编程进行实现。无需配置安装其他开源库。

## 程序说明

### 数据集加载与显示

主要思路：需要从文本文件中读取数据集，因此编写了以下类 ReadData 和 Digit。类的包含关系是 ReadData 类中包含 Digit 类的列表。

同时有以下的模板函数用于将字符串数据类型<string>转换为<int>数据类型。

```
template <class Type>
Type stringToNum(const std::string &str);
```

#### class ReadData 功能：用于加载存储数据集

数据	含义
std::string datasdir;	数据集存储路径
int num;	数据集个数
std::vector<Digit> datas;	数据集
函数	功能
void read_datas();	从 txt 中读取数据集
ReadData(std::string dir);	构造函数，设置数据集存储路径 (存储路径)

#### class Digit 功能：用于存储单个手写数字样本

数据	含义
std::vector<int> feature;	特征
int label;	标签
函数	功能
void get_labe(int &label);	获取标签
void get_feature(std::vector<std::string> &strArray);	从字符串中获取特征 (字符串类型的特征)
void show();	显示数据
Digit();	//构造函数

### 随机森林算法

主要思路：训练主要通过 RandomForest 类的函数实现，Tree 类和 Node 类仅仅存储必要的数据以减少内存的使用。训练的样本所占用的内存仅在训练单个决策树时存在，不会独立存储。随机读取的数据集仅仅采用打乱数据集索引的方式，也不会对数据集进行复制存储。函数参数传递也都是用引用传递以减少数据的复制。

类的包含关系是 RandomForest 类中包含 Tree 类的列表。Tree 类中包含 Node 类的列表。

**class RandomForest: 功能：训练决策树并存储，并生成随机森林**

数据	含义
double S;	随机训练样本的比例
std::vector<Tree> Forest;	决策树
int max_depth;	最大深度
int min_sample_count;	决策树节点的最小样本数
int max_categories;	分类个数
int number_of_feature;	随机选择使用的特征个数
double regression_accuracy;	精度终止条件
int max_num_of_trees_in_the_forest;	森林中决策树的最大数量
double forest_accuracy;	当前精度
int num_of_trees_in_the_forest;	当前森林决策树个数
函数	功能
int train(std::vector<Digit> &datas, std::vector<Digit> &test);	训练随机森林 (训练用的数据集, 计算精度用的测试集)
Node set_node_with_gini(std::vector<Digit> &datas, std::vector<int> &datas_random_index, std::vector<int> &feature_random_index);	计算分裂指标 (训练用的数据集, 当前节点的数据集样本的索引, 训练用的特征的索引)
int predict(Digit &digit);	随机森林预测函数(单个样本)
RandomForest(double S, int max_depth, int min_sample_count, int max_categories, int number_of_feature, double regression_accuracy, int max_num_of_trees_in_the_forest);	构造函数，设置随机森林的相关参数 (训练随机使用的样本比例, 决策树最大深度, 分支最小样本数量, 最大分类数量, 随机选择使用的特征个数, 终止精度, 随机森林决策树最大个数)

**class Tree 功能：用于存储单个决策树以及其中的节点**

数据	含义
std::vector<Node> nodes	决策树的节点
函数	功能
void push_node(Node &node);	增加节点
int predict(Digit &digit);	单个决策树预测结果(单个样本)
Tree();	构造函数

**class Node 功能：存储节点信息，并提供函数用于外部调用设置节点参数**

数据	含义
int parent;	父节点索引
int left;	子节点索引
int right;	子节点索引
int depth;	当前节点深度
int split_feature;	分裂使用的特征
int split_threshold;	分裂阈值
int result;	该节点的预测结果，比例最大分类
bool same;	是否所有的样本都相同 相同也就不需要继续分裂
函数	功能
void set_node(int parent, int left, int right, int depth, int split_feature, int split_threshold, int result, bool same)	设置节点参数(父节点索引, 子节点索引, 子节点索引, 节点深度, 分裂使用的特征, 分裂阈值, 该节点的预测结果, 是否所有的样本都相同)
void set_split_param(int &split_feature, int &split_threshold, int result, bool same);	设置分裂参数(分裂使用的特征, 分裂阈值, 该节点的预测结果, 是否所有的样本都相同)
Node();	节点构造函数

## 运行程序

1. 给编译 build.sh 增加可执行权限

```
sudo chmod +x build.sh
```

2. 编译程序

```
./build.sh
```

3. 运行程序

```
./ main
```

4. 示例说明

读取数据集

```
std::string optdigits_train_datas; //训练集文件名
```

```
optdigits_train_datas = "optdigits.tra";
```

```
ReadData optdigits_train(optdigits_train_datas);
```

```
std::string optdigits_test_datas; //测试集文件名
```

```
optdigits_test_datas = "optdigits.tes";
```

```
ReadData optdigits_test(optdigits_test_datas);
```

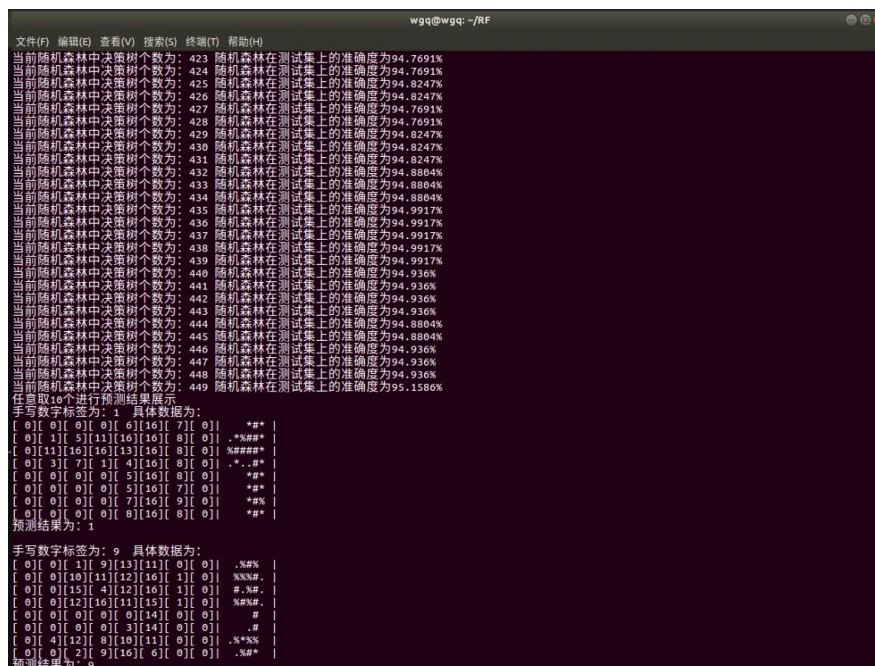
训练随机森林

```
RandomForest MyRandomForest(0.8, //训练随机使用的样本比例
25, //决策树最大深度
10, //分支最小样本数量
10, //最大分类数量
16, //随机选择使用的特征个数
0.95, //终止精度
1000); //随机森林决策树最大个数
MyRandomForest.train(optdigits_train.datas, optdigits_test.datas);
```

使用随机森林进行预测

```
std::cout << "任意取 10 个进行预测结果展示" << std::endl;
std::vector<int> datas_random_index(optdigits_test.datas.size());
for (int i = 0; i < optdigits_test.datas.size(); ++i)
{
    datas_random_index[i] = i;
}
srand(unsigned(time(0))); //随机种子
std::random_shuffle(datas_random_index.begin(), datas_random_index.end());
for (int i = 0; i < 10; ++i)
{
    optdigits_test.datas[datas_random_index[i]].show();
    std::cout << "预测结果为: " <<
    MyRandomForest.predict(optdigits_test.datas[datas_random_index[i]]) << std::endl
    << std::endl;
}
```

5. 结果展示



```
wqg@wqg: ~/RF
当前随机森林中决策树个数为: 423 随机森林在测试集上的准确度为94.7691%
当前随机森林中决策树个数为: 424 随机森林在测试集上的准确度为94.7691%
当前随机森林中决策树个数为: 425 随机森林在测试集上的准确度为94.8247%
当前随机森林中决策树个数为: 426 随机森林在测试集上的准确度为94.8247%
当前随机森林中决策树个数为: 427 随机森林在测试集上的准确度为94.7691%
当前随机森林中决策树个数为: 428 随机森林在测试集上的准确度为94.7691%
当前随机森林中决策树个数为: 429 随机森林在测试集上的准确度为94.8247%
当前随机森林中决策树个数为: 430 随机森林在测试集上的准确度为94.8247%
当前随机森林中决策树个数为: 431 随机森林在测试集上的准确度为94.8247%
当前随机森林中决策树个数为: 432 随机森林在测试集上的准确度为94.8804%
当前随机森林中决策树个数为: 433 随机森林在测试集上的准确度为94.8804%
当前随机森林中决策树个数为: 434 随机森林在测试集上的准确度为94.8804%
当前随机森林中决策树个数为: 435 随机森林在测试集上的准确度为94.9917%
当前随机森林中决策树个数为: 436 随机森林在测试集上的准确度为94.9917%
当前随机森林中决策树个数为: 437 随机森林在测试集上的准确度为94.9917%
当前随机森林中决策树个数为: 438 随机森林在测试集上的准确度为94.9917%
当前随机森林中决策树个数为: 439 随机森林在测试集上的准确度为94.9917%
当前随机森林中决策树个数为: 440 随机森林在测试集上的准确度为94.936%
当前随机森林中决策树个数为: 441 随机森林在测试集上的准确度为94.936%
当前随机森林中决策树个数为: 442 随机森林在测试集上的准确度为94.936%
当前随机森林中决策树个数为: 443 随机森林在测试集上的准确度为94.936%
当前随机森林中决策树个数为: 444 随机森林在测试集上的准确度为94.8804%
当前随机森林中决策树个数为: 445 随机森林在测试集上的准确度为94.8804%
当前随机森林中决策树个数为: 446 随机森林在测试集上的准确度为94.936%
当前随机森林中决策树个数为: 447 随机森林在测试集上的准确度为94.936%
当前随机森林中决策树个数为: 448 随机森林在测试集上的准确度为94.936%
当前随机森林中决策树个数为: 449 随机森林在测试集上的准确度为95.1586%
任意取10个进行预测结果展示
手写数字标签为: 1 具体数据为:
[ 0][ 0][ 0][ 0][ 0][ 0][ 0][ 0][ 0][ 0]  **
[ 0][ 1][ 5][ 11][ 16][ 16][ 0][ 0]  .***
[ 0][ 11][ 16][ 16][ 13][ 16][ 0][ 0]  .***
[ 0][ 3][ 7][ 1][ 4][ 16][ 0][ 0]  .*.
[ 0][ 0][ 0][ 0][ 5][ 16][ 0][ 0]  .**
[ 0][ 0][ 0][ 0][ 5][ 16][ 7][ 0]  .**
[ 0][ 0][ 0][ 0][ 7][ 16][ 9][ 0]  .**
[ 0][ 0][ 0][ 0][ 8][ 16][ 0][ 0]  .**
预测结果为: 1
手写数字标签为: 9 具体数据为:
[ 0][ 0][ 1][ 9][ 13][ 11][ 0][ 0]  .%#%
[ 0][ 0][ 10][ 11][ 12][ 16][ 1][ 0]  .%#%#
[ 0][ 0][ 15][ 4][ 12][ 16][ 1][ 0]  .#.#%#
[ 0][ 0][ 12][ 16][ 11][ 13][ 1][ 0]  .%#%#
[ 0][ 0][ 0][ 0][ 0][ 14][ 0][ 0]  .#
[ 0][ 0][ 0][ 0][ 3][ 14][ 0][ 0]  .#
[ 0][ 4][ 12][ 8][ 10][ 11][ 0][ 0]  .%#%#
[ 0][ 0][ 2][ 9][ 16][ 0][ 0][ 0]  .%#%#
预测结果为: 9
```