实验报告

13331233 孙中阳

问题分析

问题描述

This is a classification task. Please use what you have learned from our course and write an algorithm to train a classifier.

There are 2 classes and 11392-dimensional features for each sample. There are 2177020 training examples and 220245 testing examples.

数据格式

训练数据

```
label index0:value0 index2:value2 index3:value3 ...
```

测试数据

```
id index0:value0 index1:value1 index2:value2 ...
```

提交数据

```
id,label
```

问题特征及注意事项

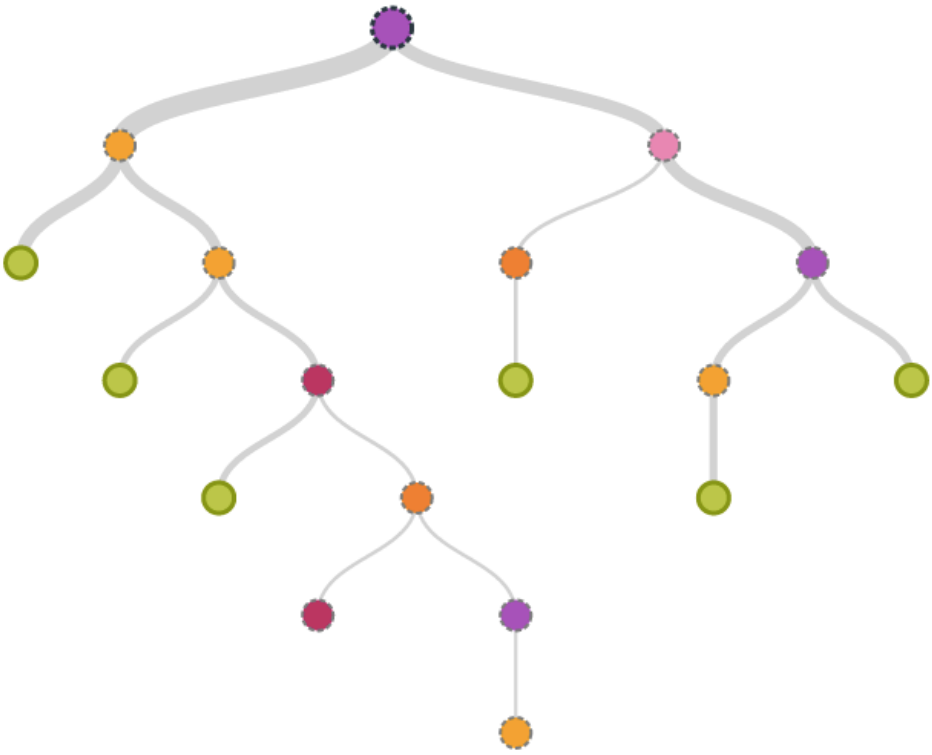
1.相较一般主机的处理能力，本次实验数据量较大，内存可能会成为性能瓶颈
2.要求使用分类器，本次试验的偏重分类而不是求值
3.数据量较大，运算量也会随之提高

处理思路

1.采用小内存机器运行小部分数据测试程序，大内存机器运行全部数据并输出的方法更有效率的得到结果，大内存机器可选择租用按量付费的云服务器
2.分类器可采用决策树，易于设计且效果较好。同时对于较大训练数据的训练集，宜采用随机森林的方法避免过拟合
3.一个线程只能同时计算一棵决策树，因此采用多线程同时计算决策树可有效使用计算资源

算法原理

决策树



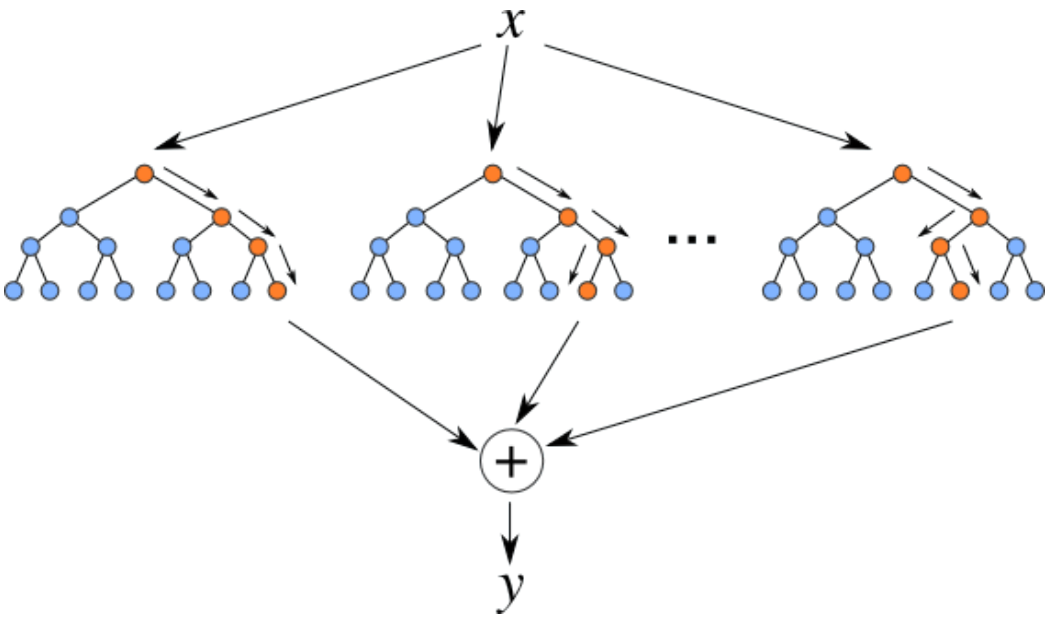
一棵决策树是一个树形数据结构，测试数据根据节点对应的属性值判断自己下一步应该进入哪个子节点，最终到达叶子节点获取最终的结果。

训练决策树的方法主要通过以下步骤来实现：
1.选取一定量的训练集数据和一部分的属性
2.对每个属性，根据选取此属性后获得的划分计算GINI指数，熵等，最终获得

一个最好划分的属性
3.将计算好的属性ID存入节点，根据该属性计算出新的属性集合，划分出新的训练数据集合，然后将其分配给子节点继续计算

决策树的优点：
1.易于理解，比较直观，不需要过多的数学基础
2.计算简便，不需要较长的运行时间

随机森林



一棵决策树能够根据一定的样本和属性获取结果，那么很多棵树根据自己的结果共同决定效果理论上会更好。就像是一个专家有一部分的领域知识，很多专家综合自己的领域知识投票，获得的结果会更加准确。这样可以避免少数几棵树造成过拟合的问题。

程序设计

系统架构

模块设计

IO：读取training.txt和test.txt文件中的数据并保存在hash map中，保存在hash map中有助于后面查找属性值时迅速定位属性，且不需要保存为0的属性，这样可以节省大量空间
create thread:根据输入设置每个线程要完成的树的数量，然后开启线程
thread:每个线程独立选取样例，属性并建树
trees:每个线程将建好的树汇总
output:用测试集数据遍历每棵树，得到结果并输出到本地文件，同时训练集中10000个数据的测试结果也打印在屏幕上

另外可以选择是否直接开始下一轮建树，这样可以省去IO部分消耗的时间

关键代码

IO:

```
void read_data(string file_name) //此处可能有巨大的传递开销
{
    //读取数据，可根据file_name决定读取训练数据或者测试数据
    int count = 0;  //标记已经读取的数据的数目
    unordered_map<int, unordered_map<int, bool> > result;
    ifstream input(file_name);
    string line, tmp_tuple;
    while(getline(input, line))
    {
        cout << count << endl;
        istringstream string_line(line);  //采用istringstrea
m规避空格

        if(file_name == TRAINING_FILE_NAME)
    {
            string_line >> tmp_tuple; //训练数据是带结果的，优
先保存结果
            if(tmp_tuple == "1") TRAINING_REAULT[count] = t
rue; //本程序中大量采用bool值保存数据，这样可节省一定的内存空间
            else TRAINING_REAULT[count] = false;

        while(string_line >> tmp_tuple)
        {   //保存属性值为1的属性的ID
            int pos = tmp_tuple.find(':');
            string s1;
            s1 = tmp_tuple.substr(0, pos);
            TRAINING_DATA[count][atoi(s1.c_str())] = true;
            ZERO_ATTR[atoi(s1.c_str())] = 1; //ZERO_ATTR中属
性为1的在选取属性构造决策树时才会被选中
        }
   }
```

create thread:

```
cin >> TREE_PER_THREAD >> ATTRIBUTION_SELECT_AMOUNT;  //输入每个线程计算多少树，每棵树有多少个属性
for(int i = 0; i < THREAD_AMOUNT; i++)
{
    threads.push_back(thread(thread_function));
}

for(auto& t: threads)
{
    t.join();
}
```

具体函数

```
void thread_function()
{
    //每个线程构造一些树，同时有一个字节的seed
    srandom(time(0));

    for(int i = 0; i < TREE_PER_THREAD; i++)
    {
        build_tree();
        cout << "tree complete once" << endl;
    }
}
```

thread：
先选取样例和属性

```
void build_tree()
{
    //构造树，这里不是递归的，可直接获取一个根节点，首先会计算出随机的example_id和attribution_id给根节点
    vector<int> example_id = getRandom(EXAMPLE_AMOUNT, EXAMPLE_SELECT_AMOUNT);
    vector<int> attribution_id = getRandom2(ATTRIBUTION_AMOUNT, ATTRIBUTION_SELECT_AMOUNT);

    node* tree = create_tree(example_id, attribution_id);

    trees.push_back(tree);
}
```

然后根据选取的样例和属性进行创建

```
node* create_tree(vector<int> example_id, vector<int> attribution_id)
{
    cout << "create_tree" << endl;
    //构造树是递归的，先创建属于自己的节点，判断自己是不是叶子节点，如果不是，那么计算最优的用于划分的属性，然后根据属性分派自己的到的example_id和attribution_id到左子节点和右子节点。如果是叶子节点则返回
    node* root = new node();
    root->example_id = example_id;
    root->attribution_id = attribution_id;
    if(root->is_leaf_node()) return root;
    root->attribution_number = root->caculate_id();

    vector<int> left_example, right_example;

    //划分训练数据
    for(auto c: example_id)
    {
    if(TRAINING_DATA[c].find(root->attribution_number) != TRAINING_DATA[c].end()) left_example.push_back(c);
    else right_example.push_back(c);
    }

    //这里需要加一个判断，就是划分到左子节点或者右子节点的训练数据的个数是否为0，如果为0则也是叶子节点
    if(left_example.size() == 0 || right_example.size() ==
```

```
0)
    {
        root->leaf_node = true;
        int one_number = 0;
        int zero_number = 0;
        for(auto c: example_id)
        {
            if(TRAINING_REAULT[c] == true) one_number++;
            else zero_number++;
        }

    if(one_number >= zero_number) root->leaf_result = 1;
    else root->leaf_result = 0;
    return root;
    }
    //重新计算属性
     auto it = attribution_id.begin();
    while(it != attribution_id.end())
    {
        if((*it) == root->attribution_number)attribution_id
.erase(it);
        else it++;
    }

    vector<int> new_attr = attribution_id;
    root->left_child = create_tree(left_example, new_attr);
    root->right_child = create_tree(right_example, new_attr
);
    cout << "" << endl;
    if(root->leaf_node) cout << root->leaf_result << endl;
    else cout << root->attribution_number << endl;
    return root;
}
```

output:

```
    ofstream output("output.txt");
    output << "id,label" << endl;

    for(int i = 0; i < OUTPUT_AMOUNT; i++)
    {
        int one_count = 0, zero_count = 0, out = 0;
        for(auto tree: trees)
        {
            int tmp_thread = tree->bianli2(i);
            if(tmp_thread == 1) one_count++;
            else zero_count++;
        }
        if(one_count >= zero_count) out = 1;
        output << i << ',' << out << endl;
    }
```

测试方法

测试环境

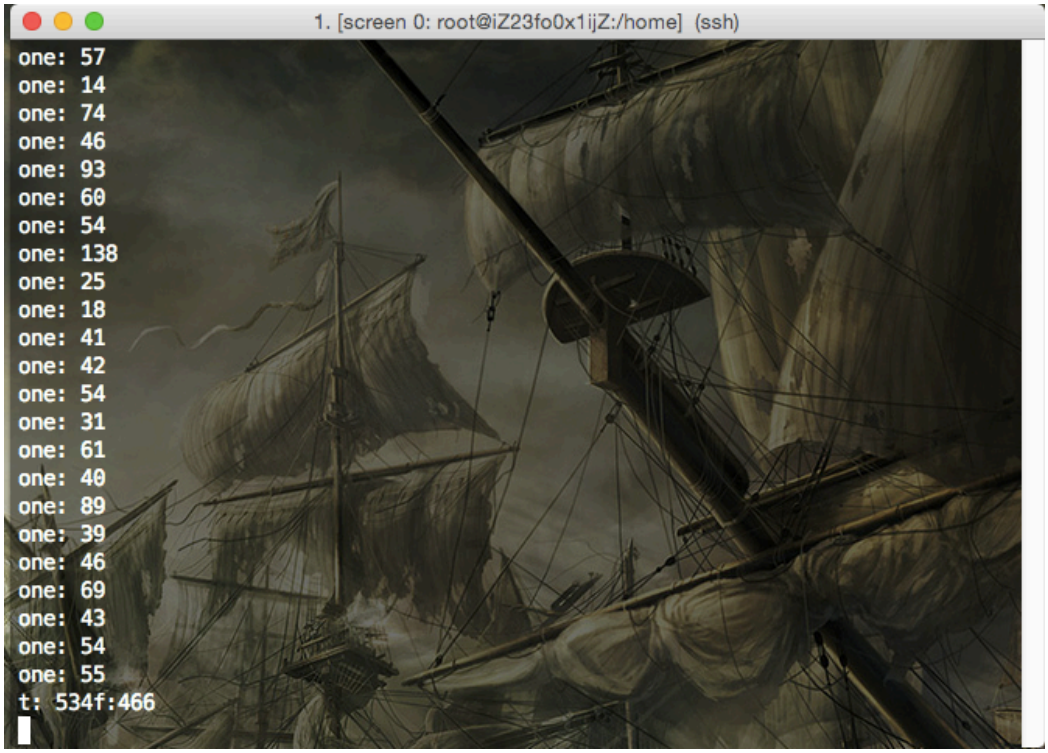
详见配置文件，主要为Macbook Air

测试思路

1.根据已有程序，检验程序的正确性
2.根据少部分的样例，判断树的数量和准确度的关系
3.根据少部分样例，判断树的属性数量和准确度的关系
4.估计执行速度，计算算法的时间复杂度，估算执行时间以安排合适的配置

测试用例

测试用例示例：

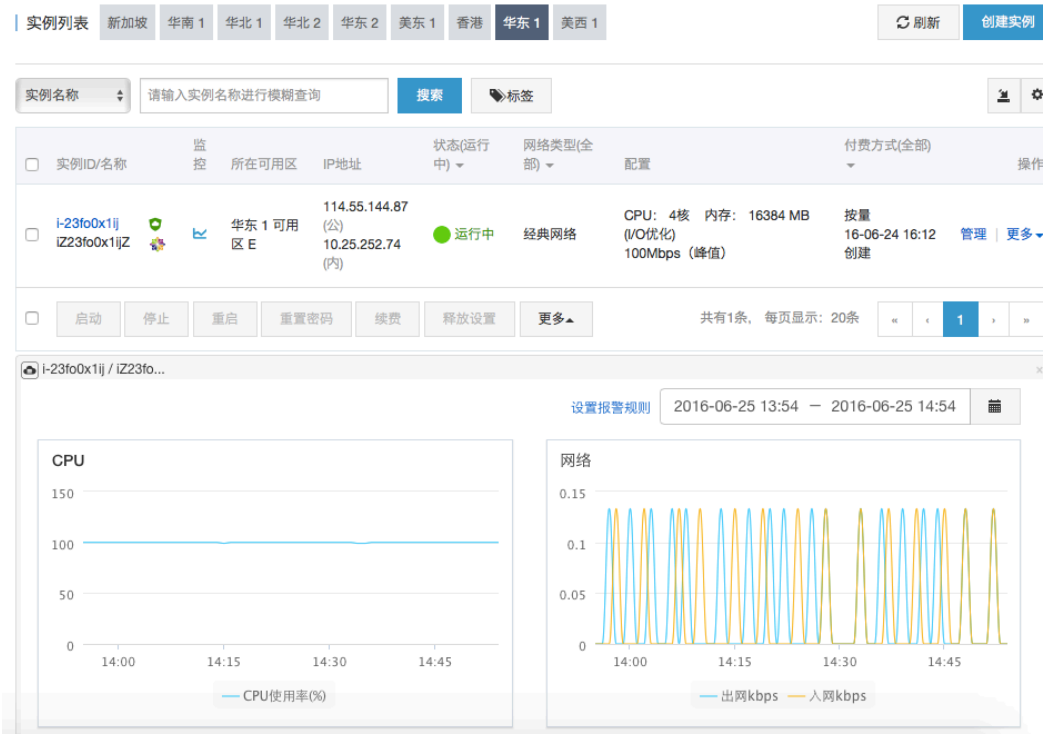


测试结果

从结果上来看，200个属性是比较合适的，关于属性的时间复杂度为 O(n^2) 树

的个数越多越好（暂时无法测试过拟合），之后再实际环境中进行测试

实际环境：



在选取树的个数为1600时，取得了不错的成果

结论

通过本次试验，我较为深入系统的了解到决策树的实现原理，具体的实现方法和测试技巧，了解到随机森林的长处和多线程构造随机森林的具体方法。很有收获