DecisionTree 알고리즘 구현

컴퓨터소프트웨어학부

2016025996

조성진

내가 DecisionTree 를 구현한 방법은 decisionTree 구조체를 만들고, 해당 구조체를 루트로 가지는 한 노드를 생성한다. 우선 노드안에 모든 데이터를 집어넣고, 재귀적으로 트리의 자식들을 생성하는 방법으로 구현하였다. 노드를 재귀적으로 구현하면서 leaf 로 판단하는 경우가 크게 3 가지 있는데, 첫번째로 만약 노드안의 모든 데이터가 같은 class label 을 가지게 된다면, 해당 노드는 leaf 라고 판단한다. 두번째로 모든 attribute 를 분류하는데 이미 사용해서 더이상 사용할 수 있는 attribute 가 없다면 leaf 노드로 판단한다. 또한 분류된 노드에 데이터셋이 아무것도 존재하지 않는경우 leaf 노드로 판단하게 된다. 두번째 경우에 더이상 사용할 수 있는 attribute 가 없는경우, majority voting 을 통해 decision 을 정하게 되며, 세번째 경우 attribute 가 없는 경우 부모 노드의 majority voting 을 통해 decision 을 할 수 있도록 알고리즘을 구현하였다. 여기서 Attribute selection 기준에 따라 자식노드의 수가 달라지게 되는데, Information Gain 과 Gain ratio 의 경우 attribute 가 가지는 label 의 수 만큼 자식노드를 만들고, Gini Index 의 경우 2 개의 자식노드만 가지게 된다.

기능구현을 하기위해 내가 만든 함수들을 보면 우선 위에서 설명한 트리를 나누는 방법에 따라 두가지 함수로 나뉜다. Gini 가 뒤에 붙은 함수들은 Information Gain 방식에서도 사용하지만 매우 비슷한 형태로 Gini 구현에 사용된 함수들이다. 그리고 앞으로 나오는 모든 함수들에서 type 의 값에 따라 DecisionTree 가 다른방식으로 만들어지는데, type == 0 인경우 Information Gain, type==1 인 경우 Gain Ratio, type==2 인 경우 Gini 방식이 된다.

```
decisionTree makeDecisionTree(int type) {
    vector<bool> use(test_attr_size, false);
    decisionTree tree(db_table);
    if(type < 2)
        buildTree(tree, -1, use, type);
    else
        buildTreeGini(tree, -1);
    return tree;
}
```

1.
```
void buildTree(decisionTree& t, int attr_idx, vector<bool>& use, int type)
```
2.
```
void buildTreeGini(decisionTree& t, int attr_idx) {
```

이 두가지 함수는 위에 알고리즘 설명에 쓰인대로 자식노드를 만들거나 leaf 노드를 판단하고 class label을 predict하는데 사용된다. 다만 Information Gain방식에서 사용되는 함수는 attribute를 사용한 후 다시 사용하지 않아 use 라는 bool배열을 사용하지만, Gini 방식의 경우 사용된 attribute를 다시 사용할 수 있어 인자가 존재하지 않는다.

1. `int findNextAttrIdx(vector<vector<int> >& table, vector<bool>& use, int type) {`
2. `int findNextAttrIdxGini(vector<vector<int> >& table, vector<int>& sep) {`

해당 두 함수는 다음 Attribute의 index를 찾는데 사용되는 함수들인데, 각각의 attribute selection기준에 따라 cost가 최대 또는 최소가 되는 경우를 인덱스로 구한다. 여기서 Gini의 경우 나뉘는 인덱스 뿐만 아니라 실제 나뉘어지는 attribute의 label의 조합도 중요하기 때문에 sep vector을 사용하며 해당 내용을 저장한다.

```
double getInfoGain(vector<vector<int> >& count_table, int total, int idx, int type) {
    double ret = 0.0;
    vector<int> sum_cols(category_size[idx], 0);
    for(int i = 0; i < count_table.size(); ++i) {
        for(int j = 0; j < count_table[i].size(); ++j) {
            sum_cols[i] += count_table[i][j];
        }
    }
    vector<int> sum_rows(category_size[attr_size-1], 0);
    for(int i = 0; i < count_table.size(); ++i) {
        for(int j = 0; j < count_table[i].size(); ++j) {
            sum_rows[j] += count_table[i][j];
        }
    }


    double InfoD = getInfoD(sum_rows, total);
    for(int i = 0; i < count_table.size(); ++i) {
        double tmp = 0.0;
        for(int j = 0; j < count_table[i].size(); ++j) {
            tmp += infoGain(count_table[i][j], sum_cols[i]);
        }
        tmp *= (double)(sum_cols[i]) / total;
        ret += tmp;
    }
    ret = InfoD - ret;
    if(type == 1) {
        double split = splitInfo(count_table, total, idx);
        ret /= split;
    }
    return ret;
}
```

이 함수는 Information Gain을 계산하는 함수이다.

```
double getInfoD(vector<int>& count, int total) {
    int size = count.size();
    double ret = 0;
    for(int i = 0; i < count.size(); ++i) {
        if(count[i] == 0) continue;
        ret += infoGain(count[i], total);
    }
    return ret;
}
```

이 함수는 가지고 있는 Data에 대한 전체 Entropy를 계산하는 방법이다.

```
double gini(vector<vector<int> >& count_table, int total, int attr_idx, vector<int>& part) {
    double ret = 0.0;
    ret = binarySplit(count_table, total, attr_idx, part);
    return ret;
}
```

이 함수는 Gini값을 계산하는 함수이며, 위에 적은 sep vector의 역할을 part 벡터가 한다.

```
double binarySplit(vector<vector<int> >& count_table, int total, int attr_idx, vector<int>& part) {
    vector<int> allAttr;
    vector<int> subset;
    set<int> s;
    for(int i = 0; i < count_table.size() ;++i) {
        for(int j = 0; j < count_table[i].size(); ++j) {
            if(count_table[i][j] > 0) {
                s.insert(i);
            }
        }
    }
    for(auto it = s.begin(); it != s.end(); ++it) {
        allAttr.push_back(*it);
    }
    double minVal = 1e9;
    double ret = search(count_table, allAttr, 0, subset, part, minVal);
    return ret;
}
```

```
double search(vector<vector<int> >& count_table, vector<int> allAttr, int pos, vector<int>& subset, vector<int>& part, double& minVal) {
    if(pos == allAttr.size()) {
        if(subset.size() == 0 || subset.size() == allAttr.size() /*|| subset.size() < allAttr.size() / 2*/) {
            return 1e9;
        } else {
            vector<int> v(allAttr.size() + subset.size());
            vector<int>::iterator it;
            it = set_difference(allAttr.begin(), allAttr.end(), subset.begin(), subset.end(), v.begin());
            v.resize(it - v.begin());
            int cnt1 = 0, cnt2 = 0;
            for(int i = 0; i < v.size(); ++i) {
                for(int j = 0; j < count_table[v[i]].size(); ++j) {
                    cnt1 += count_table[v[i]][j];
                }
            }
            for(int i = 0; i < subset.size(); ++i) {
                for(int j = 0; j < count_table[subset[i]].size(); ++j) {
                    cnt2 += count_table[subset[i]][j];
                }
            }
            int total = cnt1 + cnt2;
            double ret = (double)cnt1 / total * getGini(count_table, v, cnt1, allAttr.size()) + (double)cnt2 / total * getGini(count_table, subset, cnt2, allAt
            if(minVal > ret) {
                minVal = ret;
                part.clear();
                part.assign(subset.size(), 0);
                copy(subset.begin(), subset.end(), part.begin());
            }
            return ret;
        }
    }
    subset.push_back(allAttr[pos]);
    search(count_table, allAttr, pos+1, subset, part, minVal);
    subset.pop_back();
    search(count_table, allAttr, pos+1, subset, part, minVal);
    return minVal;
}
```

Gini 값을 구하기 위해 한 attribute를 binary split해야 하는데, search 함수와 함께 모든 부분집합을 조사하여 Gini값의 최솟값을 찾아낸다.

```
double infoGain(int c, int total) {
    if(c == 0 || total == 0) return 0;
    double x = (double)c / total;
    double ret = -(x * log2(x));
    return ret;
}
```

Information Gain을 구하는 함수이다.

```
double splitInfo(vector<vector<int> >& count_table, int total, int attr_idx) {
    double ret = 0.0;
    vector<int> count(category_size[attr_idx], 0);
    for(int i = 0; i < count_table.size(); ++i) {
        for(int j = 0; j < count_table[i].size(); ++j) {
            count[i] += count_table[i][j];
        }
    }
    for(int i = 0; i < count.size(); ++i) {
        ret += infoGain(count[i], total);
    }
    return ret;
}
```

Gain Ratio를 구할때 사용되는 값으로, 분모에 divide되는 값이다.

```
int getPredictedIdx(decisionTree& t, vector<int>& line, int type) {
    if(t.leaf == true) return t.predict;
    int ret = -1;
    if(type < 2) {
        ret = getPredictedIdx(t.childs[line[t.test_attribute]], line, type);
    } else {
        ret = getPredictedIdx(t.childs[t.hashTable[line[t.test_attribute]]], line, type);
    }
    return ret;
}
```

각 type에 따른 DecisionTree마다 예상되는 class label을 가져온다. Int값으로 리턴하여 Idx2Word를 통해 실제 class label을 구한다.

```
void read_trainData(ifstream& ifs, string& train_name)
void read_testData(ifstream& ifs, string& test_name)
void write_resultData(ofstream& ofs, string& result_name)
```

다음 세 함수는 인풋을 읽는 두가지 함수와 출력을 파일로 남기는 한가지 함수이다. 단순히 파일입출력을 진행하며, 입력하는 함수의 경우 Idx2Word, Word2Idx등 벡터들을 만들며 데이터를 integer type으로 바꿔주는 역할을 수행한다. 출력하는 경우에 integer 을 다시 string으로 바꿔준다.

```
vector<vector<int> > db_table;
vector<string> attribute_name;
vector<map<string, int> > Word2Idx;
vector<map<int, string> > Idx2Word;
vector<int> category_size;
int attr_size = 0;
int test_attr_size = 0;

vector<vector<int> > test_db_table;
vector<int> result;
```

다음은 구현을 하면서 사용한 전역변수들 목록이다.

해당 알고리즘을 cpp로 구현했기 때문에 Makefile을 작성하였다.

```
~/Class/DataScience/Assignment/2
❯ make
c++ -std=c++11   -c -o assignment2.o assignment2.cpp
g++ -o assignment2.out assignment2.o
```

Make로 실행파일을 만든후

```
~/Class/DataScience/Assignment/2
❯ ./assignment2.out dt_train1.txt dt_test1.txt dt_result1.txt
```

만약 make로 컴파일이 실패한다면

```
~/Class/DataScience/Assignment/2
❯ g++ -o assignment2.out assignment2.cpp -std=c++11
```

다음과 같은 방법으로 컴파일 할 수 있다.

실행방법의 경우 ./실행파일명 train_name test_name result_name와 같은 형태로 실행하면된다(Linux, MacOS)

나의 컴파일 환경은 다음과 같다.

```
❯ g++ --version
Configured with: --prefix=/Library/Developer/CommandLineTools/usr --with-gxx-include-dir=/Library/Developer/CommandLineTools/SDKs/MacOSX.sdk/usr/include/c++/4.2.1
Apple clang version 12.0.0 (clang-1200.0.32.29)
Target: x86_64-apple-darwin20.3.0
Thread model: posix
InstalledDir: /Library/Developer/CommandLineTools/usr/bin
```

Information Gain, Gain Ratio, Gini 세가지 방법 모두 구현했고, 정확도를 확인해보니 각각 315/346, 318/346, 335/346이 나와 Gini가 가장 정확도가 높았고, 서로 조합을 해봐도 Gini보다 높은 정확도를 얻지 못해 Gini방법을 사용한 파일을 과제로 제출한다.