<Programming Assignment #2>

2017029716 박혜정

1. Environment

- OS : Windows

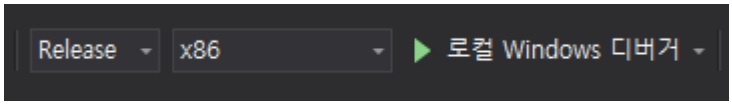
- Language : C++

- Tool : Visual Studio Community 2019

2. 컴파일 및 실행방법

1) 컴파일 방법

- Visual studio의 도구창에서 Debug->Release로 변환해주세요.



- Ctrl + F5 를 눌러 실행 시켜주시고, 실행된 파일은 종료 시켜주세요.

- 솔루션 폴더의 Release 폴더에 dt.exe가 생성된 걸 확인할 수 있습니다.

2) 실행 방법

```
C:₩Users₩HB₩Desktop₩dt₩Release>dt.exe dt_train.txt dt_test.txt result.txt
C:₩Users₩HB₩Desktop₩dt₩Release>dt.exe dt_train1.txt dt_test1.txt result1.txt
```

소스코드(dt.cpp)은 /decisionTree/dt 폴더 안에,

실행파일(dt.exe)과 데이터셋(dt_train.txt, dt_train1.txt, dt_test.txt, dt_test1.txt)는 /decisionTree/Release 폴더 안에 있습니다.

따라서 /decisionTree/Release 폴더에서 cmd창에

[실행파일명] [training file name] [test file name] [result file name]

를 입력하여 프로그램을 실행시킬 수 있습니다.

3. 코드 설명

1) Summary

맨 처음 파일을 읽어 attribute List와 data들을 읽어온 후, 각 attribute로 분류 시 entropy가 얼마나 줄어드는지 확인합니다. 가장 많이 줄어드는 attribute로 분류를 진행하며, 분류된 집합에도 같은 동작을 반복하며 decision tree를 만들어나갑니다. (information gain 방법 사용)

Test의 경우 data를 읽어와 tree를 따라가며 결과값을 예측합니다.

2) 자료 구조

```
typedef class Node {
public:
    vector<pair<string, bool> > attributes;
    vector<vector<string> > datas;
    vector<Node* > childNodes;

    int keyAttributeIndex;
    string value;
    double info;
    string classificationResult;
    bool isLeaf;

    Node() {
        classificationResult = "ROOT";
        isLeaf = false;
    }
```

Node는 다음과 같이 attribute List 와 데이터들을 담는 datas, 그리고 자식 노드를 가르키는 childNodes를 가진다. 또한 자식을 분류할 때 쓰는 attribute의 index를 저장하는 keyAttributeIndex, 자식 노드의 경우 분류될 때 사용된 attribute의 값을 저장하는 value, 노드의 information을 저장하는 info와 그 노드에서 멈출 경우 cateforical value와 말단 노드인지를 확인하는 isLeaf를 변수로 가지고 있다.

```
void makeResult() {
    vector<pair<string, int> > counts;
    int max, index = 0;
    for (int i = 0; i < datas.size(); ++i) {
        bool isInList = false;
        for (int j = 0; j < counts.size(); ++j) {
            if (counts[j].first.compare(datas[i][attributes.size() - 1]) == 0) {
                isInList = true;
                counts[j].second++;
            }
        }
        if (!isInList) counts.push_back(make_pair(datas[i][attributes.size() - 1], 1));
    }

    max = counts[0].second;
    for (int i = 1; i < counts.size(); ++i) {
        if (max < counts[i].second) {
            max = counts[i].second;
            index = i;
        }
    }
    classificationResult = counts[index].first;
}
```

또한 categorical value를 찾는 makeResult 메소드가 있으며,

```
void makeInfo() {
    vector<pair<string, int> > count;
    double result = 0.0;
    for (int i = 0; i < datas.size(); ++i) {
        bool isInList = false;
        for (int j = 0; j < count.size(); ++j) {
            if (datas[i][attributes.size() - 1].compare(count[j].first) == 0) {
                count[j].second++;
                isInList = true;
            }
        }
        if (!isInList) {
            count.push_back(make_pair(datas[i][attributes.size() - 1], 1));
        }
    }
    for (int i = 0; i < count.size(); ++i) {
        double p = (double)count[i].second / datas.size();
        result += -1.0 * p * log2(p) / log(2);
    }
    info = result;
}
```

Information을 계산하는 makeInfo 메소드도 가지고 있다.

3)파일 읽기

```
void readFile(string filename, vector<pair<string, bool> >& attributes, vector<vector<string> >& datas) {
    ifstream openFile(filename.data());

    if (!openFile.eof()) {
        string line;

        getline(openFile, line);
        vector<string> attributesList;
        int pre = 0;
        for (int i = 0; i < line.size(); i++) {
            if (line[i] == '₩t') {
                string word = line.substr(pre, i - pre);
                attributesList.push_back(word);
                pre = i + 1;
            }
        }
        string word = line.substr(pre, line.size() - pre);
        attributesList.push_back(word);

        for (int i = 0; i < attributesList.size(); ++i) {
            attributes.push_back(make_pair(attributesList[i], false));
        }

        while (getline(openFile, line)) {
            vector<string> data;
            int pre = 0;
            for (int i = 0; i < line.size(); i++) {
                if (line[i] == '₩t') {
                    string word = line.substr(pre, i - pre);
                    data.push_back(word);
                    pre = i + 1;
                }
            }
            string word = line.substr(pre, line.size() - pre);
            data.push_back(word);

            datas.push_back(data);

        }
    }
}
```

첫 줄을 읽어와 attribute List를 만들고, 그 다음줄부터 파일이 끝날 때까지 읽어서 데이터를 형성합니다. 데이터는 '₩t'을 기준으로 끊어서 attribute의 값으로 저장합니다.

4) 분류 함수

```
vector< vector<pair<string, vector<vector<string> > > > > classify(Node* node) {
    //attribute 별로 값으로 분류된 벡터를 가지자.
    vector< vector<pair<string, vector<vector<string> > > > > classify(node->attributes.size());

    for (int attributeIndex = 0; attributeIndex < node->attributes.size() - 1; ++attributeIndex) {
        if (node->attributes[attributeIndex].second == true) continue;
        for (int datasIndex = 0; datasIndex < node->datas.size(); ++datasIndex) {
            bool isInList = false;
            for (int attributeValueIndex = 0; attributeValueIndex < classify[attributeIndex].size(); ++attributeValueIndex) {
                if (node->datas[datasIndex][attributeIndex].compare(classify[attributeIndex][attributeValueIndex].first) == 0) {
                    isInList = true;
                    classify[attributeIndex][attributeValueIndex].second.push_back(node->datas[datasIndex]);
                }
            }
            if (!isInList) {
                vector<vector<string> > temp;
                temp.push_back(node->datas[datasIndex]);
                classify[attributeIndex].push_back(make_pair(node->datas[datasIndex][attributeIndex], temp));
            }
        }
    }

    return classify;
}
```

주어진 데이터 셋을 특정 attribute의 값을 기준으로 하여 분류하는 함수입니다.

5) information 계산 함수

```
double calculateInfo(vector<pair<string, vector<vector<string> > > > datas, int classAttribute) {
    int all = 0;
    double info = 0;

    // 전체 갯수
    for (int i = 0; i < datas.size(); ++i) {
        vector<pair<string, int > > classify;
        all += datas[i].second.size();

        // class attribute value별로 갯수 세기
        for (int datasIndex = 0; datasIndex < datas[i].second.size(); ++datasIndex) {
            bool isInList = false;
            for (int j = 0; j < classify.size(); ++j) {
                if (classify[j].first.compare(datas[i].second[datasIndex][classAttribute]) == 0) {
                    isInList = true;
                    classify[j].second++;
                }
            }
            if (!isInList) {
                classify.push_back(make_pair(datas[i].second[datasIndex][classAttribute], 1));
            }
        }

        double temp = 0;
        for (int k = 0; k < classify.size(); ++k) {
            double p = (double)classify[k].second / datas[i].second.size();
            temp += -1.0 * p * log(p) / log(2);
        }
        info += datas[i].second.size() * temp;
    }
    info /= all;

    return info;
}
```

주어진 데이터 셋의 information을 계산하는 함수입니다.

6) decision tree 만들기

```
void chooseAttribute(Node** node) {

    vector< vector<pair<string, vector<vector<string> > > > > classified = classify(*node);
    double maxGain = -1;
    double nowInfo = (*node)->info;
    int classifiedAttribute = 0;
    bool isAttributesAllChecked = true;

    for (int i = 0; i < (*node)->attributes.size(); ++i) {
        if ((*node)->attributes[i].second == false) isAttributesAllChecked = false;
    }

    if (isAttributesAllChecked) {
        (*node)->makeResult();
        (*node)->isLeaf = true;
        return;
    }
    if (nowInfo == 0) {
        (*node)->makeResult();
        (*node)->isLeaf = true;
        return;
    }
    if ((*node)->datas.size() == 0) {
        (*node)->isLeaf = true;
        (*node)->makeResult();
        return;
    }

    for (int attributeIndex = 0; attributeIndex < classified.size(); ++attributeIndex) {
        double gain = nowInfo - calculateInfo(classified[attributeIndex], classified.size() - 1);
        if (gain > maxGain) {
            maxGain = gain;
            classifiedAttribute = attributeIndex;
        }
    }

    (*node)->keyAttributeIndex = classifiedAttribute;
```

```
for (int i = 0; i < classified[classifiedAttribute].size(); ++i) {
    if (classified[classifiedAttribute][i].second.size() != 0) {
        Node* nNode = new Node();
        nNode->datas = classified[classifiedAttribute][i].second;
        nNode->attributes = (*node)->attributes;
        nNode->attributes[classifiedAttribute].second = true;
        nNode->value = classified[classifiedAttribute][i].first;
        nNode->makeInfo();
        nNode->makeResult();
        (*node)->childNodes.push_back(nNode);
    }
}

for (int i = 0; i < (*node)->childNodes.size(); ++i) {
    chooseAttribute(&((*node)->childNodes[i]));
}
```

위의 분류함수와 information 계산 함수를 이용하여, 각 attribute를 분류 기준으로 삼았을 때의

information gain을 측정하고, information gain이 가장 큰 attribute로 partition을 진행합니다.

자식노드들에도 재귀적으로 이 과정을 반복하며, 분류된 데이터가 모두 같은 class에 속하거나, 더 이상 데이터가 없거나, 모든 attribute를 분류 기준으로 사용한 경우 분류과정을 멈춥니다.

7) 결과 예측 함수

```
string guess(vector<string> data, Node* rootNode) {
    Node* cur = rootNode;
    string result = cur->classificationResult;
    while (!cur->isLeaf) {
        for (int i = 0; i < cur->childNodes.size(); ++i) {
            if (data[cur->keyAttributeIndex].compare(cur->childNodes[i]->value) == 0) {
                cur = cur->childNodes[i];
                result = cur->classificationResult;
                i = 0;
                if (cur->childNodes.size() == 0) return result;
            }
        }
        cur = cur->childNodes[0];
    }
    return result;
}
```

만들어진 decision tree를 따라가며 주어진 data의 categorical value를 예측합니다.

8) 테스트 함수

```
void test(string testFileName, Node* rootNode, string outputFileName) {
    Node* testNode = new Node();
    readFile(testFileName, testNode->attributes, testNode->datas);

    string str = "";
    ofstream writeFile;
    writeFile.open(outputFileName);

    for (int i = 0; i < rootNode->attributes.size() - 1; ++i) {
        str += rootNode->attributes[i].first + '₩t';
    }
    str += rootNode->attributes[rootNode->attributes.size() - 1].first + '₩n';
    writeFile.write(str.c_str(), str.size());
    str = "";

    for (int i = 0; i < testNode->datas.size(); ++i) {
        string result = guess(testNode->datas[i], rootNode);
        for (int j = 0; j < testNode->datas[i].size(); ++j) {
            str += testNode->datas[i][j] + '₩t';
        }
        str += result + '₩n';
        writeFile.write(str.c_str(), str.size());
        str = "";
    }

    writeFile.close();
}
```

테스트 파일을 읽어와 그 데이터의 결과값을 예측하여 파일로 저장합니다.

4. dt_test.exe 결과

```
C:₩Users₩HB₩Desktop₩dt₩Release>dt_test dt_answer.txt result.txt
5 / 5

C:₩Users₩HB₩Desktop₩dt₩Release>dt_test dt_answer1.txt result1.txt
304 / 346
```

다음과 같은 결과를 확인할 수 있었습니다.