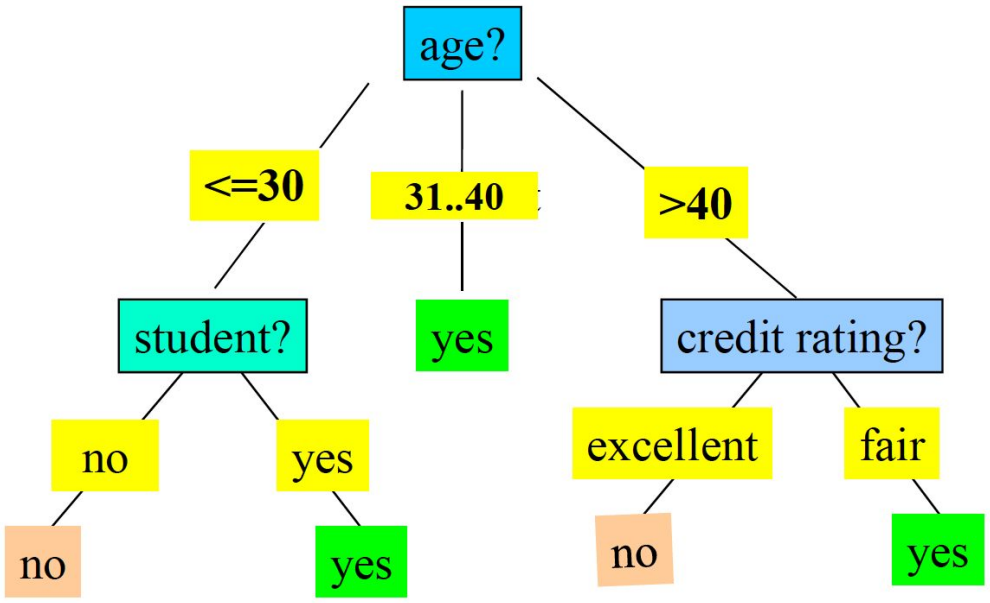
DataScience Class

Decision Tree

2016025096 강수진

1. Decision Tree algorithm



Decision Tree is one of supervised learning model for classification and regression. It separates datas based on specific standard(questions). (-> questions indicate 'Node')

1) Basic Algorithm

A greedy algorithm that constructs a decision tree in a top-down, recursive and divide-and-conquer manner.

- training time

a) All the training examples are at the root.
b) Change all attributes to be categorical. (continuous valued->discretized)
c) Select test attributs based on attribue selection measure.(information gain, gain ratio, gini index)
d) training examples are partitioned recursively based on the selected test attributes.

- test time

all test samples follow decision tree and find their appropriate labels.

2) Contitions for stopping the partitioning process

- All samples for a given node belong to the same class
- If there are no remaining attributes for further partitioning - majority voting is employed for classifying leaf

- If there are no samples left

3) Test attribute selection

- Information gain

a) Entropy (expected information) : more heterogeneous, the higher

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

b) Entropy after using A attribute to split data

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

c) Information gained by branching on attribute A
: The higher gain, the better attribute

$$Gain(A) = Info(D) - Info_A(D)$$

- Gain ratio

Information gain measure is biased towards attributes with a large number of values

a) Gain ratio : The higher gain ratio, the better attribute

$$GainRatio(A) = Gain(A) / SplitInfo_A(D)$$

b) Split Info

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2(\frac{|D_j|}{|D|})$$

- Gini index

a) Gini index

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

b) If attribute A splits data set into two subsets, gini index can be represented in the forms below.

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

c) Reduction in impurity : The larger value, the better attribute

$$\Delta gini(A) = gini(D) - gini_A(D)$$

2. Code implementation

(1) Code structure

To simplify my code, I listed all classes with their member variables and method without specifying data types or return values.

class DecisionTree

private:

Node root;//root node
int numOfAttr;//the number of attributes(except class label)
int numOfAllSamples;//the number of training samples
double threshold=0.001;//pre-pruning threshold
vector<vector<string>>trainSet;//training dataset
vector<vector<string>>testSet;//test dataset

public:

DecisionTree()//constructor
run()//method for running decision tree algorithm
training()//training step
test()//test step
testSamples()//return class for one test sample
getEntropy()//return entropy for given tuples
getEntropyAttr()//return entropy after using specified attribute
getSplitInfo()//get split info for given attribute
attrSelection()//test attribute selection -> gain ratio + pre-pruning
partition()//partition in top-down way
isEnd()//determine whether to stop partitioning or not

class Node

private:

int numOfSamples;//the number of training samples which correspond to this node
int numOfAttr;// the number of attributes
vector<vector<string>>tuples;//training samples
vector<Node>child;//child node
pair<int,string> nodeAttr;//<test attribute index, test attribute value>
vector<bool>attrChecker;//to check used attribute from the root node

public:

Node()//constructor
getTuples()//get training samples
getNumOfSamples()//get the number of training samples
getNumOfChildren()//get the number of child nodes of this node
setChild()//setter for initialize child node
getNodeAttr()//getter for nodeAttr pair
getChildren()//getter for all child nodes
getAttrChecker()//getter for attrChecker
checkAttr()//to specify attribute that I used
addChild()//function to add child node

class IO

private:

string trainFile;//train file name. ex)dt_train.txt
string testFile;//test file name. ex)dt_test.txt
string resultFile;//result file name. ex)dt_result.txt
string attrSet;//attribute name list
ifstream readFile;//input file stream
ofstream writeFile;//output file stream
vector<vector<string>>trainSet;//training dataset
vector<vector<string>>testSet;//test dataset

public:

IO()//constructor
fileRead()//read training file and test file
createOutputFile()//create result file
split()//split string by delimeter
getTrainSet()//getter for train set
getTestSet()//getter for test set

(2) Function description

I explained each member variables and functions(method) in details.

- **DecisionTree class**

DecisionTree class is for running decision tree algorithm.

member variables

```
private:
    Node root;
    int numOfAttr;
    int numOfAllSamples;
    double threshold=0.001;
    vector<vector<string>>trainSet;
    vector<vector<string>>testSet;
```

constructor

```
DecisionTree(vector<vector<string>> _trainSet, vector<vector<string>> _testSet):
trainSet(_trainSet), testSet(_testSet), root(_trainSet,make_pair(-1,"")){
    numOfAttr=int(trainSet[0].size()-1);
    numOfAllSamples=int(trainSet.size());
}
```

This is a constructor in DecisionTree class.

run()

```
void run(){
    training();
    test();
}
```

This is for running decision tree algorithm. In training step, decision tree is constructed based on test attribute. In test step, each test sample follows structured decision tree and get class result.

training()

```
void training(){
    root=partition(root);
}
```

For training step, root node calls partition function and each child node call partition function recursively based on decision tree algorithm.

test()

```
void test(){
    for(int i=0;i<testSet.size();i++){
        string ans=testSamples(testSet[i],root);
        testSet[i].push_back(ans);
    }
}
```

For test step, each test set calls testSamples function and get class label result.

testSamples()

```
string testSamples(vector<string>sample, Node node){
    if(node.getNumOfChildren()==0){
        int classIndx=int(numOfAttr);
        vector<vector<string>>tuple=node.getTuples();
        map<string,int>m;
        for(auto tup:tuple)
            m[tup[classIndx]]++;
        int numOfTuplesInClass=0;
        string labelName;
        for(pair<string,int>group:m){
            if(group.second>numOfTuplesInClass){
                numOfTuplesInClass=group.second;
                labelName=group.first;
            }
        }
        return labelName;
    }
    vector<Node>children=node.getChildren();
    for(auto childNode:children){
        pair<int,string>attrPair=childNode.getNodeAttr();
        if(sample[attrPair.first]==attrPair.second)
            return testSamples(sample, childNode);
    }
    return "None";
}
```

Until the node has no children, test sample follows the decision tree. If attrPair.second(attribute value) is same with sample's attribute value, this function recursively calls testSamples function and pass parameter for child node. If the node has no child nodes, the majority class label of tuples is the result for this node.

getTestResult()

```
vector<vector<string>>getTestResult(){
    return testSet;
}
```

Function to return test set.

getEntropy()

```
double getEntropy(vector<vector<string>>tuple){
    map<string,int>m;
    int classIndx=int(numOfAttr);
    double info=0.0;
    for(auto tup:tuple){
        m[tup[classIndx]]++;
    }
    for(pair<string,int> p:m){
        int num=int(p.second);
        double prob=(double)num/(double)tuple.size();
        info-=prob*log2(prob);
    }
    return info;
}
```

Function to return entropy for given tuples(vector<<vector<string>>tuple).

getEntropyAttr()

```
double getEntropyAttr(int attrIndx,vector<vector<string>> tuple){
    double infoAttr=0.0;
    map<string,vector<vector<string>>>m;
    for(auto tup:tuple){
        m[tup[attrIndx]].push_back(tup);
    }
    for(auto group:m){
        vector<vector<string>>groupedTuples=group.second;
        infoAttr+=((double)groupedTuples.size()/(double)tuple.size())*getEntropy(groupedTuples);
    }
    return infoAttr;
}
```

Function to get entropy after splitting data based on specified attribute(attrIndx).

getSplitInfo()

```
double getSplitInfo(int attrIndx, vector<vector<string>> tuple){
    double splitInfo=0.0;
    double numOfTuples=double(tuple.size());
    map<string,int>m;
    for(auto tup:tuple){
        m[tup[attrIndx]]++;
    }
    for(auto group:m){
        double prob=((double)group.second)/numOfTuples;
        splitInfo-=prob*log2(prob);
    }
    return splitInfo;
}
```

Function to return split info for given attribute(attrIndx).

attrSelection()

```
int attrSelection(Node node){
    double info=getEntropy(node.getTuples());
    int attrSelected=-1;
    double gainRatioMax=0.0;
    vector<bool>attrChecker=node.getAttrChecker();
    for(int i=0;i<numOfAttr;i++){
        if(attrChecker[i])continue;
        vector<vector<string>>tuples=node.getTuples();
        double gain=info-getEntropyAttr(i,tuples);
        double splitInfo=getSplitInfo(i, tuples);
        if(gain/splitInfo>gainRatioMax){
            attrSelected=i;
            gainRatioMax=gain/splitInfo;
        }
    }
    return attrSelected;
}
```

Test attribute selection function. I used gain ratio measure to select the best attribute. For unused attribute, I compared their gain ratio to get maximum gain ratio attribute.

partition()

```
Node partition(Node node){
    if(isEnd(node)==true)return node;
    int attrIndx=attrSelection(node);
    node.checkAttr(attrIndx);
    map<string,vector<vector<string>>>m;
    for(auto tup:node.getTuples())
        m[tup[attrIndx]].push_back(tup);
    double childAverageSamples=(double)node.getNumOfSamples()/(double)m.size();
    if(childAverageSamples/numOfAllSamples>=threshold){
        for(auto group:m){
            node.addChild(m[group.first],make_pair(attrIndx,group.first));
        }
        vector<Node>child=node.getChildren();
        int num=node.getNumOfChildren();
        for(int i=0;i<num;i++){
            node.setChild(i, partition(child[i]));
        }
    }
    return node;
}
```

In partition process, I select attribute by using attrSelection function(gain ratio). And then, I used pre-pruning by comparing (average samples of splitted groups/the number of training samples) with threshold. I found the best value for threshold by trying this process many times and that was 0.001. If the value exceeds threshold, this function recursively calls partition and set child node info.

isEnd()

```
bool isEnd(Node node){
    //no samples left
    int numOfSamples=node.getNumOfSamples();
    if(numOfSamples==0)return true;
    //belong to the same class
    int classIndx=int(numOfAttr);
    vector<vector<string>>tuples=node.getTuples();
    map<string,int>m;
    for(auto tup:tuples)
        m[tup[classIndx]]++;
    if(m.size()==1)return true;
    //no remaining attributes
    vector<bool>attrChecker=node.getAttrChecker();
    int num=0;
    for(int i=0;i<numOfAttr;i++){
        if(attrChecker[i]==false)num++;
    }
    if(num==0)return true;
    return false;
}
```

This function decided whether to split the node further or not. There are 3 conditions for stopping partitioning process.

- **Node class**

Node class is for storing information in each node.

member variable

```
private:
    int numOfSamples;
    int numOfAttr;
    vector<vector<string>>tuples;//tuples
    vector<Node>child;//child node
    pair<int,string> nodeAttr;
    vector<bool>attrChecker;
```

Node()

```
Node(vector<vector<string>>_tuples, pair<int,string>attrInfo){
    tuples = _tuples;
    nodeAttr = attrInfo;
    numOfSamples=int(tuples.size());
    numOfAttr=int(tuples[0].size()-1);
    attrChecker.resize(numOfAttr,false);
}
```

Constructor for node class.

getTuples()

```
vector<vector<string>> getTuples(){
    return tuples;
}
```

getter for tuples

getNumOfSamples()

```
int getNumOfSamples(){
    return numOfSamples;
}
```

getter for numOfSamples variable

getNumOfChildren()

```
int getNumOfChildren(){
    return int(child.size());
}
```

getter for number of child node size

setChild()

```
void setChild(int i,Node childNode){
    child[i]=childNode;
}
```

setter for child node

getNodeAttr()

```
pair<int,string> getNodeAttr(){
    return nodeAttr;
}
```

getter for nodeAttr pair

getChildren()

```
vector<Node> getChildren(){
    return child;
}
```

getter for child nodes

getAttrChecker()

```
vector<bool> getAttrChecker(){
    return attrChecker;
}
```

getter for attribute checker

checkAttr()

```
void checkAttr(int indx){
    attrChecker[indx]=true;
}
```

Function to check attribute which I used for test attribute from the root to that node.

addChild()

```
void addChild(vector<vector<string>>tuple,pair<int,string>attr){
    child.push_back(Node(tuple,attr));
}
```

Function to add child node in this node.

- **IO class**

IO class has two roles of reading input file and writing output file.

member variable

```
private:
    string trainFile;
    string testFile;
    string resultFile;
    string attrSet;
    ifstream readFile;
    ofstream writeFile;
    vector<vector<string>>trainSet;
    vector<vector<string>>testSet;
```

constructor

```
IO(string train, string test, string result){
    trainFile = train;
    testFile = test;
    resultFile = result;
}
```

This is a constructor for io class.

fileRead()

```
void fileRead(){
    readFile.open(trainFile);
    if(readFile.is_open()) {
        string s;
        getline(readFile,s);
        attrSet=s;
        while(getline(readFile, s)){
            vector<string> parse = split(s,'\t');
            if(parse.size()>0)
                trainSet.push_back(parse);
        }
        readFile.close();
    }
    else{
        cout << "Train file is not opened\n";
        exit(0);
    }
    readFile.open(testFile);
    if(readFile.is_open()) {
        string s;
        getline(readFile,s);
        while(getline(readFile, s)){
            vector<string> parse = split(s,'\t');
            if(parse.size()>0)
                testSet.push_back(parse);
        }
        readFile.close();
    }
    else{
        cout << "Test file is not opened\n";
        exit(0);
    }
}
```

This is the method for reading train file and test file.

createOutputFile()

```
void createOutputFile(vector<vector<string>>testFile){
    writeFile.open(resultFile);
    if(writeFile.is_open()) {
        writeFile<<attrSet;
        for(vector<string> row:testFile){
            for(int i=0;i<row.size();i++){
                if(i==row.size()-1)
                    writeFile<<row[i];
                else
                    writeFile<<row[i]<<'\t';
            }
            writeFile<<'\n';
        }
        writeFile.close();
    }
    else{
        cout << "Ouput file is not opened\n";
        exit(0);
    }
}
```

This is the method for writing output file.

split()

```
vector<string> split(string str, char delimiter) {
    vector<string> ret;
    string str2=str.erase(str.find_last_not_of("\n"));
    stringstream ss(str2);
    string temp;

    while (getline(ss, temp, delimiter))
        ret.push_back(temp);

    return ret;
}
```

For spliting string by tab delimiter, I added split method.

getTrainSet()

```
vector<vector<string>>getTrainSet(){
    return trainSet;
}
```

getter for train set

getTestSet()

```
vector<vector<string>>getTestSet(){
    return testSet;
}
```

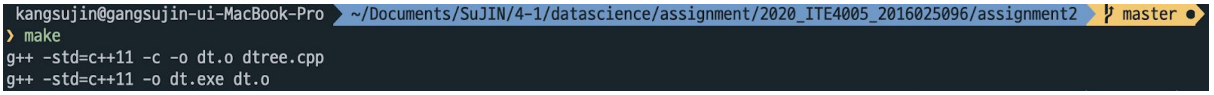
getter for test set

3. Instructions for compilation

Environment
OS : Mac OS
Language : C++

I created Makefile and followed two steps for compilation below.

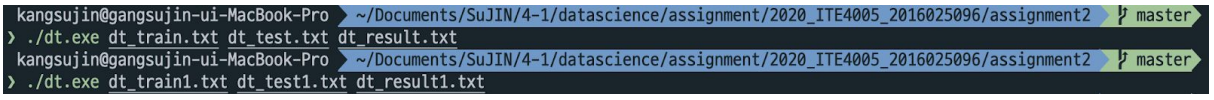
1. command : make



in Makefile, two commands are written.
g++ -std=c++11 -c -o dt.o dtree.cpp
g++ -std=c++11 -o dt.exe dt.o

=> dt.o, dt.exe files are created by using 'make' command

2. command : ./dt.exe dt_train.txt dt_test.txt dt_result.txt



If you use ./dt.exe dt_train.txt dt_test.txt dt_result.txt command, you have to put 1) dt_train.txt, 2)dt_test.txt files in same folder(with dt.exe file)
=> Then, dt.result.txt file is created in this folder.

4. Result

I used given testing program in windows

1) dt_result.txt

dt_result.txt

```
age      income  student credit_rating   Class:buys_computer
<=30     low     no      fair    no
<=30     medium  yes     fair    yes
31...40  low     no      fair    yes
>40      high    no      fair    yes
>40      low     yes     excellent       no
```

By using testing program, I got 5/5.

```
C:₩Users₩강수진>dt_test.exe dt_answer.txt dt_result.txt
5 / 5
```

2) dt_result1.txt

dt_result1.txt

buying	maint	doors	persons	lug_boot	safety	car_evaluation
med	vhigh	2	4	med	med	acc
low	high	4	4	small	low	unacc
high	vhigh	4	4	med	med	acc
high	vhigh	4	more	big	low	unacc
low	high	3	more	med	low	unacc
med	high	2	more	small	high	acc
vhigh	low	3	2	med	high	unacc
med	high	2	4	small	low	unacc
med	low	5more	4	small	med	good
med	low	5more	2	big	med	unacc
med	low	4	more	big	high	vgood
low	low	4	2	big	high	unacc
low	low	3	more	med	low	unacc
high	med	2	2	big	high	unacc
high	low	4	more	small	low	unacc
med	vhigh	3	4	med	med	acc
low	low	3	more	small	high	good
vhigh	med	2	more	med	med	acc
vhigh	low	4	more	big	high	acc
vhigh	low	2	2	small	high	unacc
high	high	5more	2	big	med	unacc
high	med	5more	2	med	low	unacc
med	med	3	2	big	high	unacc
low	vhigh	5more	2	small	low	unacc
vhigh	vhigh	3	more	small	high	unacc
low	high	2	more	big	med	acc
vhigh	vhigh	5more	more	small	high	unacc
high	low	5more	4	small	med	unacc
low	med	4	2	big	low	unacc
med	vhigh	3	2	small	med	unacc
med	high	4	more	small	med	unacc
vhigh	low	4	4	big	low	unacc
high	low	5more	4	big	med	acc
low	vhigh	3	more	small	med	unacc
vhigh	low	5more	2	small	low	unacc
low	vhigh	2	2	med	med	unacc
med	med	4	4	big	low	unacc
med	high	4	more	big	high	acc
high	high	5more	4	big	high	acc
med	vhigh	4	more	med	low	unacc
low	med	3	4	small	high	good
vhigh	med	3	2	med	low	unacc
med	high	5more	more	big	med	acc
med	med	2	4	small	high	acc
high	high	3	2	med	med	unacc
high	med	4	4	med	high	acc
vhigh	high	5more	2	small	high	unacc

By using testing program, I got 320/346.

```
C:₩Users₩강수진>dt_test.exe dt_answer1.txt dt_result1.txt
320 / 346
```