Data Science

Programming Assignment #2

컴퓨터전공

2013011695 정태화

1. Goal

Build a decision tree, and then classify the test set using this tree.

2. Summary of algorithm

This program goes through following procedure :

1. Read the entire training data and also test data from the input text files.
2. Start building decision tree with training data using Gain Ratio measure.
3. Now data sets will be partitioned by attributes with maximum Gain Ratio value until it gets to the leaf node.
4. After building tree, try classifying the test data set.
5. Write this classification result into output text file.

3. Detailed description of code

A. `class Node`

I implemented this Node class to use as a tree. Each Node has its name, and also map data to point next node with divided attribute values.

B. `vector< vector< pair<string,string> > > readTraining(string)`

This function reads data from given input file, and parameter 'string' is for input file path. These input text files have carriage return '\r', so I erased it.

Reading each line as data, this function parses every line with delimiter '\t' and inserts every items in each data into set.

And then it pushes each set into vector so that this function can return the entire data set as vector format.

C. `string deleteCR(string);`

Erases carriage return in each string.

D. `Node* initTree(vector< vector< pair<string,string> > > trainingData)`

Initiate process to build decision tree. This function transforms training data vector into the form of map to use it properly.

Also makes attributes' name set and map which contains values of each attribute.

Returns the function 'learning' which starts to build tree.

E. `Node* learning(string, vector< map<string, string> >, set<string>, map<string, set<string> > &)`

Starts decision tree building process using training data set.

We have to set the criteria for choosing appropriate attribute for partitioning database. So I used Gain Ratio measure, and it is implemented as function 'getGainRatio'.

This function is implemented recursively so that it can traverse through the tree and point next node properly.

Function will stop when it meets 2 conditions. First one is when all class labels are same in current node, and the second one is when there are no more attributes left to partition.

Returns the top(head) Node.

F. `double getInfo(string, vector<map<string,string>> &)`

```
double getSelectedInfo(string, vector<map<string,string>> &, string)

double getSplitInfo(string, vector<map<string,string>> &, string)

string getGainRatio(string, vector<map<string,string>> &, set<string>)
```

These are functions for calculating Gain Ratio value. Function 'getGainRatio' contains other 3 functions and returns finally selected attribute which has maximum Gain Ratio.

Function 'getInfo' is to calculate expected information(entropy) needed to classify a tuple.

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

Function 'getSelectedInfo' is to calculate expected information needed after using specific attribute to split current node.

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

You can get Information gained value by

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

Further, you can calculate gain ratio using SplitInfo so that it can normalize the information gain.

Function 'getSplitInfo' is to get the value for choosing the best split point with the minimum information requirement.

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

Finally, you can have Gain Ratio.

$$GainRatio(A) = {}^{Gain(A)}\!\big/_{SplitInfo(A)}$$

G. `pair<bool,string> isAllSame(string, vector<map<string,string>> &)`

Checks if current node's every data's class labels are same and returns True or False with the name of Class label.

H. `string majorityVote(string, vector< map<string, string> > &)`

While partitioning, if there are no remaining attributes for further partitioning, use majority voting with leaf node data set's class label for classifying the leaf.

I. `void makeResult(Node* cur, vector< vector< pair<string,string> > > &testData, string className, string path)`

Function to write out classification result data to output text file. Classification for test data set 'classLabel' is executed in this procedure.

J. `string classLabel(Node* cur, map<string,string> data)`

Classify class label for every test data. Each data traverses through pre-built decision tree and the leaf node's name is its class label. This function can check leaf node when there are no more child in specific node.

Sometimes data can lose its way while traversing through the tree, because some attributes do not have all the values given. In this case, I used majority voting to classify this node's label.

4. Result

By testing "dt_result1.txt" with "dt_answer1.txt", the score was **320/346**.

This means that I got 320 correct predictions out of 346 answers. In the form of percentage, it shows **92.49%** accuracy.

5. Instructions for compiling this code

- This project contains 'decision_tree.cpp', 'Makefile', 'dt_train1.txt' , and 'dt_test1.txt'.

- In project folder path, just type 'make' in terminal, or please type below line. This will generate executable file 'dt' for linux.

```
$ g++ -O2 -o dt decision_tree.cpp --std=c++11
```

- Now you will be able to execute this file with specified arguments.

```
$ ./dt 5 dt_train1.txt dt_test1.txt dt_result1.txt
```

6. Any other specifications

- This code is written in C++11.

- Compiler must support C++11 standard.

- This program is compiled with g++ and xcode.

- This program compilation is tested on macOS High Sierra.