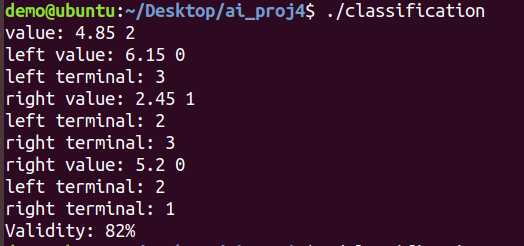
Assignment 4

1. Demo

The following fig is about executing.



There are no argument input, instead, user can modified some defined line (in classification.hpp ) to test what user want to test. After executing, the two number after “value” meaning the threshold and the number represent which attribute. And with the pre-order way to dump the tree on the screen. And the terminal represents what the tree lead a certain input data to. At last the screen would output the validity about the testing data. In the example, training data and testing data are same, so the validity is 100%.

1. Result

Because those data set on the website have no same format, so my experiment almost tested by the data set “iris”. Which have about 200 data inside, for kind of attribute and three class. Also, the first class is very easy to be classify by only one attribute.

* 1. Using only CART

The decision tree in this supervise training is necessary and the base structure in the experiment. Using only CART, is not called the forest. But it can use for test some case. Also, the result can be compare to other case, like the condition this assignment given.

* + 1. Testing data and train data rate

In the table, x axis is for training data take % data and y axis is used for testing validity. Each case takes 10 example. However, in the test we can’t find out the obvious different of changing testing data set size after the train data have 10% of total data, it shows that 10% of data is enough to train and the data have not obvious over-fitting situation happened thought there are only one tree.

In case the small data set and few class could affect the result I also try the “glass.data” to test.

And it shows the similar result to iris data set. According two result I guess the data with more attribute and less noise can validity higher.

* 1. Using random forest

To do the experiment, I still used “glass.data”. And used about 30% of total data for testing data. And I tried to modified the tree number to test. And the tree bagging is also use about 30% of training data. And still consider all attribute.

In the following fig. the x axis is for 2^x tree used to test, and y is the validity.

Although all of them are higher than 80% and the oscillation also exist, we can still observe that there is a trend raising, and it seem like there are an exponential relation between validity and tree number. Maybe if test more time can reduce the oscillation. However, the spending time is also increased a lot.

Also I tried modified the attribute number for all tree, I reduce the number to 6 (-3) and test it again. It seems that it has more oscillation at beginning. But when the tree number are raising, the oscillation are release a lot. And the perform seem same or even better than the upper one.

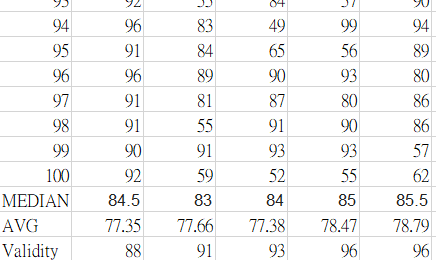
The case is I only choose one attribute; the performance is higher than I guess. After about 1000 tree, the validity is almost 100% present.

* 1. Extremely random forest

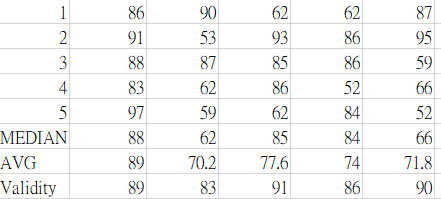
It is a tree that “At each node splitting, just randomly select an attribute”. So that if there are only one attribute it is always extremely random tree.

So I tried the upper example, and find the performance.

* 1. Out-of-bag errors and validation-set errors



The condition is same as 2.2. second test, and build 100 tree. And repeat 5 time. We can find out the validation-set from testing data is higher than average and median number.



Also, if there are only 5 tree, it also has same trend. And it seems intuitive because more tree can eliminate some useless tree’s imformation.

1. Other
   1. Remaining questions and ideas of future investigation

This program for the assignment is not very programmable, so that I am not sure when I testing those experiment all parameter is set absolutely right, thought I checked a lot of time. For example, the 2.4 part, testing the out-of-bag error, I modify the original function to reference and add some variable, and when I what to resume and test other part, it is possible that there are many problems I don’t come up.

Otherwise, I am a bit not sure those experience I tested and those data I get are coincident appear or even those data are “good” data set. And the data set on the web site all have different format, making it is more hard to once tested lot of different set and compare to each other, maybe in future I can solve the trouble and use them to try.

And there is some condition I want to test and because of the program is hard to modify in some part. One of case I want to test is changing some condition about extremely random forest and I found that there are many paper compare between Random Forest and Extra tree.

About out-of-bag part, thought compute out-of-bag error is easy to compute, but I don’t know where the error testing can be used. Maybe it can be used to compute every tree’s weight with a certain way and function. And I only test few condition, maybe there are not condition I don’t consider have different result.

Code

Main.cpp

#include "classification.hpp"

int main(int argc, char\*\* argv){

    srand((unsigned int)time(NULL) + getpid());

    vector<Example> dataSet = readData();

    vector<Example>& trainDataSet = dataSet;

    int treeNum = TREE\_NUM;

    vector<Node\*> forest;

    vector<Example> testDataSet = apart\_data(trainDataSet);

    while(treeNum--){

        vector<Example> randomDataSet  = random\_part(dataSet);

        if(randomDataSet.size() > 1) forest.push\_back(CART\_build(randomDataSet));

    }

    Node\* finialTree = vote(forest);

    dump\_screen(finialTree);

    test\_vaildity(finialTree, testDataSet);

    return 0;

}

Classification.cpp

#include "classification.hpp"

vector<Example> readData(){

    vector<Example> dataSet;

    ifstream iFile(FILEDIR);

    while(!iFile.eof()){

        string tmp;

        getline(iFile, tmp);

        if(!tmp.size()) continue;

        char cstr[tmp.length() + 1];

        strcpy(cstr, tmp.c\_str());

        Example correntExp;

        char\* tok = strtok(cstr, ",");

        while(1){

            double readValue = atof(tok);

            correntExp.tag.push\_back(readValue);

            char\* tmp\_ptr = strtok(NULL, ",");

            if(!tmp\_ptr){

                if(!strcmp(tok, "Iris-setosa")) correntExp.classType = 1;

                else if(!strcmp(tok, "Iris-versicolor")) correntExp.classType = 2;

                else if(!strcmp(tok, "Iris-virginica")) correntExp.classType = 3;

                break;

            }

            tok = tmp\_ptr;

        }

        dataSet.push\_back(correntExp);

    }

    return dataSet;

}

#include "classification.hpp"

extern int rate;

//read data, obviously

vector<Example> readData(){

    vector<Example> dataSet;

    ifstream iFile(FILEDIR);

    while(!iFile.eof()){

        string tmp;

        getline(iFile, tmp);

        if(!tmp.size()) continue;

        char cstr[tmp.length() + 1];

        strcpy(cstr, tmp.c\_str());

        Example correntExp;

        char\* tok = strtok(cstr, ",");

        tok = strtok(NULL, ",");

        while(1){

            double readValue = atof(tok);

            correntExp.tag.push\_back(readValue);

            char\* tmp\_ptr = strtok(NULL, ",");

            if(!tmp\_ptr){

//this part is used for different format data input.

                /\*

                if(!strcmp(tok, "Iris-setosa")) correntExp.classType = 1;

                else if(!strcmp(tok, "Iris-versicolor")) correntExp.classType = 2;

                else if(!strcmp(tok, "Iris-virginica")) correntExp.classType = 3;

                break;\*/

                correntExp.classType = atoi(tok);

                break;

            }

            tok = tmp\_ptr;

        }

        dataSet.push\_back(correntExp);

    }

    return dataSet;

}

//usiing for tree bagging

vector<Example> random\_part(vector<Example> dataSet){

    //This part is used for only one CART tree

    //return dataSet;

    vector<Example> partedData;

    for(int i=0; i<(dataSet.size()/TRAIN\_DATA\_RATE); i++)

        partedData.push\_back(dataSet[rand() % dataSet.size()]);

    //This part is usd for reduce attribute to a tree

    //notUsedNum can reduce such number of attribute

    bool arr[partedData[0].tag.size()] = {0};

    int notUsedNum = 5;

    while(notUsedNum){

        int randAttribute = rand() % partedData[0].tag.size();

        if(arr[randAttribute] == 0) {

            arr[randAttribute] = 1;

            notUsedNum --;

        }

    }

    for(int n=0; n<partedData[0].tag.size(); n++){

        if(arr[n]){

            for(int i=0; i<partedData.size(); i++){

                partedData[i].tag[n] = 0.0;

            }

        }

    }

    return partedData;

}

//using for part testing data and training data

vector<Example> apart\_data(vector<Example>& dataSet){

    vector<Example> testDataSet;

    for(vector<Example>::iterator it=dataSet.begin(); it < dataSet.end(); ){

        if((rand() % TEST\_DATA\_RATE\_TOTAL\_PART) < TEST\_DATA\_RATE){

            testDataSet.push\_back(\*it);

            it = dataSet.erase(it);

        }else it++;

    }

    return testDataSet;

}

//To built a CART tree

Node\* CART\_build(vector<Example> dataSet){

    if(dataSet.size() < 2) return NULL;

    int tagNum = dataSet[0].tag.size();

    vector<vector<double>> allThreshold;

    int selectAttribute = rand() % tagNum;

    for (int i=0; i<tagNum; i++){

        vector<double> appearNum;

        for(int j=0; j<dataSet.size(); j++){

            appearNum.push\_back(dataSet[j].tag[i]);

        }

        sort(appearNum.begin(), appearNum.end());

        vector<double> useableThreshold;

        double firstNum, secondNum = appearNum[0];

        for(int j=1; j<appearNum.size(); j++){

            if(appearNum[j] == secondNum) continue;

            firstNum = secondNum;

            secondNum = appearNum[j];

            useableThreshold.push\_back((firstNum + secondNum)/2);

        }

//This part is used for extra tree but there are some problem exist

//      if(selectAttribute == i)

            allThreshold.push\_back(useableThreshold);

//      else {

//          vector<double> tmp;

//          allThreshold.push\_back(tmp);

//      }

    }

//choosing the best node

    double minImpurity = 1.5;

    Node\* bestNode = new Node;

    vector<Example> refFirstSet;

    vector<Example> refSecondSet;

    double refFirstGini = 1.0, refSecondGini = 1.0;

    for(int j=0; j<allThreshold.size(); j++){

        for(int k=0; k<allThreshold[j].size(); k++){

            vector<Example> firstSet, secondSet;

            for (int l=0; l<dataSet.size(); l++){

                if(dataSet[l].tag[j] > allThreshold[j][k]) firstSet.push\_back(dataSet[l]);

                else secondSet.push\_back(dataSet[l]);

            }

            double firstGini = Gini\_impurity(firstSet) \* firstSet.size() / dataSet.size();

            double secondGini = Gini\_impurity(secondSet) \* secondSet.size() / dataSet.size();

            double correntImpurity = firstGini + secondGini;

            if(correntImpurity < minImpurity){

                minImpurity = correntImpurity;

                bestNode->value = allThreshold[j][k];

                bestNode->tagNum = j;

                refFirstSet = firstSet;

                refSecondSet = secondSet;

                refFirstGini = firstGini;

                refSecondGini = secondGini;

            }

        }

    }

    if(refFirstGini > 0.00001) bestNode->leftNode = CART\_build(refFirstSet);

    else {

    bestNode->leftNode = NULL;

        bestNode->leftTerminalTag = refFirstSet[0].classType;

    }

    if(refSecondGini > 0.00001) bestNode->rightNode = CART\_build(refSecondSet);

    else {

    bestNode->rightNode = NULL;

        bestNode->rightTerminalTag = refSecondSet[0].classType;

    }

    return bestNode;

}

//comput Gini's impurity

double Gini\_impurity(vector<Example> dataSet){

    const int classNum = CLASS\_NUM;

    int arr[classNum + 1] = {0};

    for (int i=0; i<dataSet.size(); i++){

        arr[dataSet[i].classType]++;

    }

    double sum = 0.0;

    for (int i=1; i<classNum+1; i++) sum += arr[i]\* arr[i] \* 1.0;

    sum /= (dataSet.size() \* dataSet.size());

    return 1.0 - sum;

}

//Used for sorting

bool compareFunc (SimpleNode node1, SimpleNode node2) {

    return (node1.value + node1.tagNum\*1000 < node2.value + node2.tagNum\*1000);

}

//Vote the node

Node\* vote(vector<Node\*> forest){

    if(!forest.size()) return NULL;

    vector<SimpleNode> allNode;

    for(int i=0; i<forest.size(); i++){

        SimpleNode correntNode;

        correntNode.value = forest[i]->value;

        correntNode.tagNum = forest[i]->tagNum;

        allNode.push\_back(correntNode);

    }

    sort(allNode.begin(), allNode.end(), compareFunc);

    int arr[allNode.size()] = {1};

    for(int i=1; i<allNode.size(); i++){

        if(allNode[i].tagNum == allNode[i-1].tagNum &&

            allNode[i].value == allNode[i-1].value){

            arr[i] = arr[i-1] + 1;

        }else arr[i] = 1;

    }

    int maxCount = 0;

    int seq = -1;

    for(int i=0; i<allNode.size(); i++){

        if(arr[i] > maxCount){

            maxCount = arr[i];

            seq = i;

        }

    }

    SimpleNode selectNodeInfo = allNode[seq];

    Node\* selectNode = new Node;

    selectNode->value = selectNodeInfo.value;

    selectNode->tagNum = selectNodeInfo.tagNum;

    vector<Node\*> subLeftForest;

    vector<Node\*> subRightForest;

    for(int i=0; i<forest.size(); i++){

        if(forest[i]->value == selectNodeInfo.value &&

            forest[i]->tagNum == selectNodeInfo.tagNum ){

            if(forest[i]->leftNode){

                subLeftForest.push\_back(forest[i]->leftNode);

            }else selectNode->leftTerminalTag = forest[i]->leftTerminalTag;

            if(forest[i]->rightNode){

                subRightForest.push\_back(forest[i]->rightNode);

            }else selectNode->rightTerminalTag = forest[i]->rightTerminalTag;

        }else {

//Can reduce space using but not really need

        //  deleteTree(forest[i]);

        }

    }

    selectNode->leftNode = vote(subLeftForest);

    selectNode->rightNode = vote(subRightForest);

    return selectNode;

}

//out put to screen

void dump\_screen(Node\* corrent){

    cout << "value: " << corrent->value << " " << corrent->tagNum << endl;

    cout << "left ";

    if(corrent->leftNode) dump\_screen(corrent->leftNode);

    else cout << "terminal: " << corrent->leftTerminalTag << endl;

    cout << "right ";

    if(corrent->rightNode) dump\_screen(corrent->rightNode);

    else cout << "terminal: " << corrent->rightTerminalTag << endl;

}

//testing vaildity

void test\_vaildity(Node\* tree, vector<Example> dataSet){

    int correctNum = 0;

    for(int i=0; i<dataSet.size(); i++){

        Node\* traveling = tree;

        int correntClassType;

        while(traveling){

            if(dataSet[i].tag[traveling->tagNum] > traveling->value){

                correntClassType = traveling -> leftTerminalTag;

                traveling = traveling -> leftNode;

            }else{

                correntClassType = traveling -> rightTerminalTag;

                traveling = traveling->rightNode;

            }

        }

        if(correntClassType == dataSet[i].classType) correctNum++;

    }

    ofstream outFile;

    outFile.open("dataOutput.txt", fstream::out | fstream::app);

    outFile << (correctNum \* 100) / dataSet.size() << endl;

    outFile.close();

    cout << "Validity: " << correctNum \* 100 / dataSet.size() << "%" << endl;

}

Classification.hpp

#include <unistd.h>

#include <iostream>

#include <fstream>

#include <vector>

#include <unordered\_map>

#include <algorithm>

#include <cstring>

#include <cmath>

#include <ctime>

#include <cstdlib>

#define FILEDIR "dataset/glass.data"

#define TEST\_DATA\_RATE 33

#define TEST\_DATA\_RATE\_TOTAL\_PART 100

#define TRAIN\_DATA\_RATE 5

#define TREE\_NUM 1

#define CLASS\_NUM 7

using namespace std;

class Example{

    public:

        int classType;

        vector<double> tag;

};

class Node{

    public:

        Node \*leftNode, \*rightNode;

        int leftTerminalTag, rightTerminalTag;

        double value;

        int tagNum;

};

class SimpleNode{

    public:

        double value;

        int tagNum;

};

vector<Example> readData();

Node\* CART\_build(vector<Example>);

vector<Example> random\_part(vector<Example>);

vector<Example> apart\_data(vector<Example> &);

double Gini\_impurity(vector<Example>);

Node\* vote(vector<Node\*>);

void dump\_screen(Node\*);

bool compareFunc (SimpleNode, SimpleNode);

void test\_vaildity(Node\*, vector<Example>);

Makefile

all:

    g++ -g main.cpp classification.cpp -o classification

clear:

    rm classification