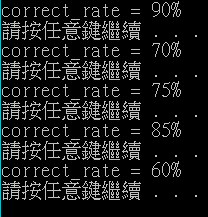
Experiments:先把資料都儲存到一個一維陣列中，接著對陣列分別用每個attribute做sorting，求出最佳的threshold，接著把陣列分成less、large兩個部分，重複以上動作，直到每個attribute皆使用過。

結果

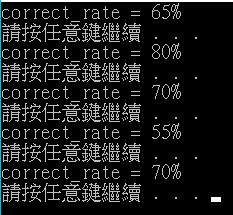
這邊以ellipse100.txt作為範例:

1. Results:首先是按照老師所要求之作法，做五次後所得結果如下圖:



可以看到正確率雖然有浮動，但大約落在70~80%左右。

1. Comparisons of out-of-bag errors and validation-set errors:

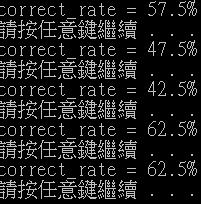


可以看到正確率有些微下降，且浮動率更大了

1. Relative sizes of the training and validation subsets:

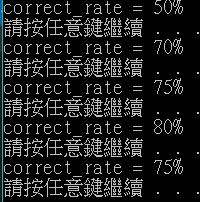
以上的數據是以80:20，也就是80筆data做training，20筆data做test，以下測試其他兩種情況:

a.60:40:



可以看到正確率下降不少。

b.10:90:

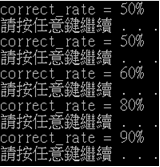


正確率浮動變大，但平均正確率和80:20相似。

4.Number of trees in the forest.

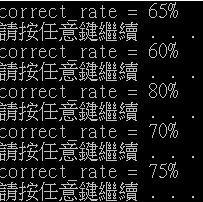
上述資料是以10棵tree所做成，以下測試兩種情況:

a.5棵tree:



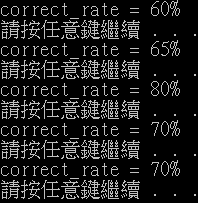
平均相似，只是正確率些微下降，幅度變大。

b.20棵tree:



相似。

5.Extremely random forest:



正確率些微下降，幅度些微變大。

Observations:我發現只有Relative sizes of the training and validation subsets有比較大的影響，其他方面而言，其實都差不多。

Interpretations:我認為實驗沒有太大的差距是由於樣本和可用的attribute太少了。

things you have learned:如何利用threshold分類。

remaining questions:在什麼情況下正確率才會大幅改變。

future investigation:可以嘗試更多attribute和資料量之training model。

Appendix

#include <iostream>

#include <fstream>

#include <string>

#include <time.h>

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

using namespace std;

class data\_set {

public:

float attr[4];

int classes;

};

class node {

public:

node \*large;

node \*less;

int test\_attr;

float threshold;

int classes;

};

class test\_data\_set {

public:

float attr[4];

int classes;

int record[20];

};

data\_set data\_in[205], training\_data[200];

test\_data\_set validation\_data[20];

int attr\_list[4];

void init() {

for (int i = 0; i < 205; i++) {

for (int j = 0; j < 4; j++) {

data\_in[i].attr[j] = 0;

training\_data[i].attr[j] = 0;

}

data\_in[i].classes = 0;

training\_data[i].classes = 0;

}

for (int i = 0; i < 20; i++) {

for (int j = 0; j < 4; j++) validation\_data[i].attr[j] = 0;

validation\_data[i].classes = 0;

}

for (int i = 0; i < 4; i++) attr\_list[i] = 0;

}

int read\_data(int num, int attr\_num) {

fstream file;

string name;

int i = 0;

char buf[200];

switch (num)

{

case 0:name = "cross200.txt"; break;

case 1:name = "ellipse100.txt"; break;

case 2:name = "iris.data"; break;

default:

break;

}

file.open(name, ios::in);

while (1) {

file >> data\_in[i].attr[0];

file >> data\_in[i].attr[1];

file >> data\_in[i].classes;

i++;

if (file.eof() != 0) break;

}

for (int j = 0; j < attr\_num; j++) attr\_list[j] = 1;

file.close();

return i - 1;

}

void random\_select(int data\_num, int train\_data\_num) {

int tmp, set[50];

srand((unsigned)time(NULL));

for (int i = 0; i < train\_data\_num;i++) {

tmp = rand() % data\_num;

for (int k = 0; k < 4; k++) validation\_data[i].attr[k] = data\_in[tmp].attr[k];

validation\_data[i].classes = data\_in[tmp].classes;

set[i] = tmp;

}

tmp = 0;

for (int i = 0; i < data\_num; i++) {

for (int j = 0; j < train\_data\_num; j++)

if (i == set[j]) continue;

training\_data[tmp] = data\_in[i];

tmp++;

}

}

void swap(data\_set &a, data\_set &b) {

data\_set temp = a;

a = b;

b = temp;

}

void sort(int start, int end, int attr\_num) {

for(int i = start; i < end;i++)

for (int j = start; j < end - i - 1; j++)

if (training\_data[j].attr[attr\_num] > training\_data[j + 1].attr[attr\_num])

swap(training\_data[j], training\_data[j + 1]);

}

float com\_impurity(int start\_point, int split\_point, int end\_point) {

float a, b, Ga, Gb, G, na, nb, total;

total = end\_point - start\_point;

na = (split\_point - start\_point)/total; nb = (end\_point - split\_point) / total;

a = 0; b = 0;

for (int i = 0; i < split\_point; i++) {

if (training\_data[i].classes == 1) a++;

else b++;

}

Ga = 1 - pow((a / (a + b)), 2) - pow((b / (a + b)), 2);

a = 0; b = 0;

for (int i = split\_point; i < end\_point; i++) {

if (training\_data[i].classes == 1) a++;

else b++;

}

Gb = 1 - pow((a / (a + b)), 2) - pow((b / (a + b)), 2);

G = na\*Ga + nb\*Gb;

return G;

}

int max(int a, int b, int c) {

if (a >= b && a >= c) return a;

if (b >= a && b >= c) return b;

if (c >= a && c >= b) return c;

}

node\* train(int start, int end) {

int attr\_num = 0;

float impurity, min\_impurity = 10;

int split\_point = 0, split\_attr = 0;

int one, two, three;

node \*new\_node = new node;

//cout << "ss" << endl;

one = 0; two = 0; three = 0; //check end condition

for (attr\_num = 0; attr\_num < 4; attr\_num++) {

if (attr\_list[attr\_num] != 0) break;

else if (attr\_num == 3) {

for (int i = 0; i < end; i++) {

if (training\_data[i].classes == 1) one++;

else if (training\_data[i].classes == 2) two++;

else if (training\_data[i].classes == 3) three++;

}

if (max(one, two, three) == one) new\_node->classes = 1;

else if (max(one, two, three) == two) new\_node->classes = 2;

else if (max(one, two, three) == three) new\_node->classes = 3;

return new\_node;

}

}

//cout << attr\_num << " " << attr\_list[attr\_num] << endl;

/\*attr\_num = rand() % 2; //Extremely random forest

if (attr\_list[attr\_num] == 0) attr\_num = 1 - attr\_num;\*/

for (; attr\_num < 4; attr\_num++) {

if (attr\_list[attr\_num] != 0) {

sort(start, end, attr\_num); //split

for (int j = start; j < end; j++) {

impurity = com\_impurity(start, j, end);

if (min\_impurity > impurity) {

min\_impurity = impurity;

split\_attr = attr\_num;

split\_point = j;

}

}

}

}

attr\_list[split\_attr] = 0; //recursive

new\_node -> threshold = (training\_data[split\_point].attr[split\_attr] + training\_data[split\_point + 1].attr[split\_attr]) / 2;

new\_node -> test\_attr = split\_attr;

new\_node -> classes = 0;

new\_node->less = new node; new\_node->large = new node;

new\_node -> less = train(start, split\_point);

new\_node -> large = train(split\_point, end);

attr\_list[split\_attr] = 1;

return new\_node;

}

void free\_tree(node \*tree) {

if (tree->classes == 0) {

free\_tree(tree->large);

free\_tree(tree->less);

}

delete tree;

return;

}

void test(node \*tree, int times, int train\_data\_num) {

int correct = 0;

for (int i = 0; i < train\_data\_num ; i++) {

node \*next = tree;

while (next->classes == 0) {

if (validation\_data[i].attr[next->test\_attr] <= next->threshold) next = next->less;

else next = next->large;

}

validation\_data[i].record[times] = next->classes;

}

free\_tree(tree);

}

void print(int train\_data\_num) {

int one = 0, two = 0, three = 0;

float result = 0, correct = 0;

for (int i = 0; i < train\_data\_num; i++) {

for (int j = 0; j < 10; j++) {

if (validation\_data[i].record[j] == 1) one++;

else if (validation\_data[i].record[j] == 2) two++;

else if (validation\_data[i].record[j] == 3) three++;

}

if (max(one, two, three) == one) result = 1;

else if (max(one, two, three) == two) result = 2;

else if (max(one, two, three) == three) result = 3;

if (validation\_data[i].classes == result) correct++;

one = 0; two = 0; three = 0;

}

cout << "correct\_rate = " << (correct / train\_data\_num)\*100 << "%" << endl;

return;

}

int main(void) {

int data\_num, file\_num, total\_attr\_num, train\_data\_num;

float correct\_rate = 0;

node \*tree;

file\_num = 0;

train\_data\_num = 20; //train\_data\_num

if(file\_num == 0 || file\_num == 1) total\_attr\_num = 2;

else total\_attr\_num = 3;

init();

data\_num = read\_data(file\_num, total\_attr\_num);

for (int j = 0; j < 5; j++) {

for (int i = 0; i < 10; i++) { //forest number

random\_select(data\_num, train\_data\_num);

tree = train(0, data\_num - train\_data\_num);

test(tree, i, train\_data\_num);

}

print(train\_data\_num);

system("pause");

}

system("pause");

return 0;

}