人工智慧概論

0613413 蔡怡君

- 實驗目的：自建 Decision Tree，並且使用 random forest 去預測，CART 為 base tree。
- 資料來源：UCI Machine Learning Repository - wine dataset
- 實驗方法：

最一開始要將資料切割成 80% training 用、20% validation 用，因為要 random forest 每一顆 Tree 都需要長的不同，(tree bagging)所以取 80%中的 training = 64%的原始 data 去自建 class CART model，(attribute bagging)透過每一層給的 attribute(自訂原始 6 個)，看每一個 attribute 去跑 threshold 找到最佳的 gini index，再去挑選最好的 attribute 透過已經計算好的 gini index，跑的同時要去限制 minimum number 跟 Tree depth。

Random forest 中的 50 顆 CART，在這裡取 0.8*50 = 40 去 predict，而 0.2*50 = 10 顆為 out of bag (OOB)，跑 predict 的時候決定最後的 class 分類是由 40 顆去投票投出來的結果。

- 結果觀察：

在實驗的過程都是使用 correct classification rate 為 y 軸。實驗結果為跑 100 次的平均。

實驗變因：

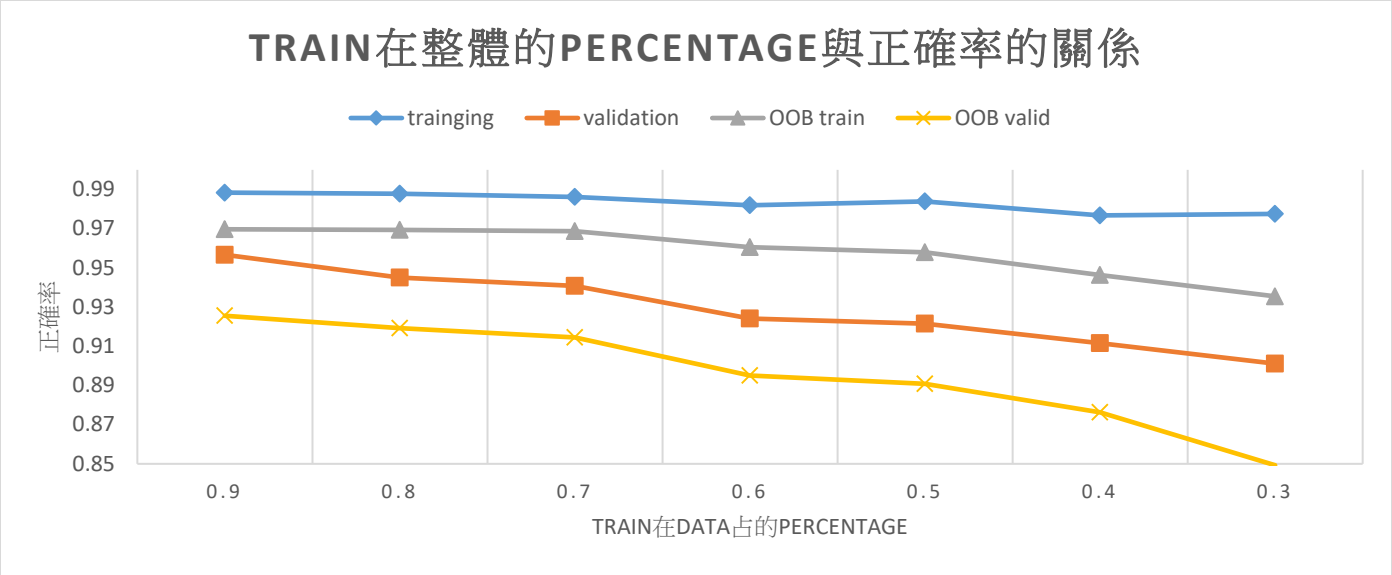
1. Size of Training and Validation
2. number of trees
3. how many attributes to consider at each node splitting. (當 attribute = 1 為特殊情形)
4. 樹的 max depth、樹的 minimum number

原始資料：

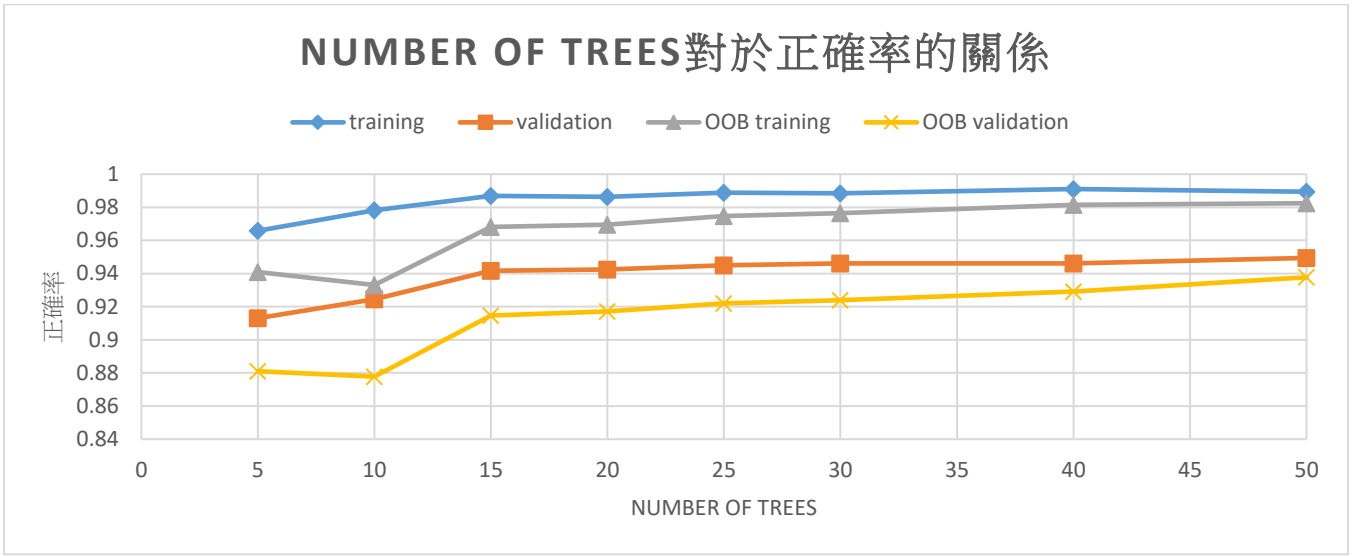
Percent of Training	Number of tree	Attribute(node split)	Max depth	Minimum number
0.8	50	6	5	5

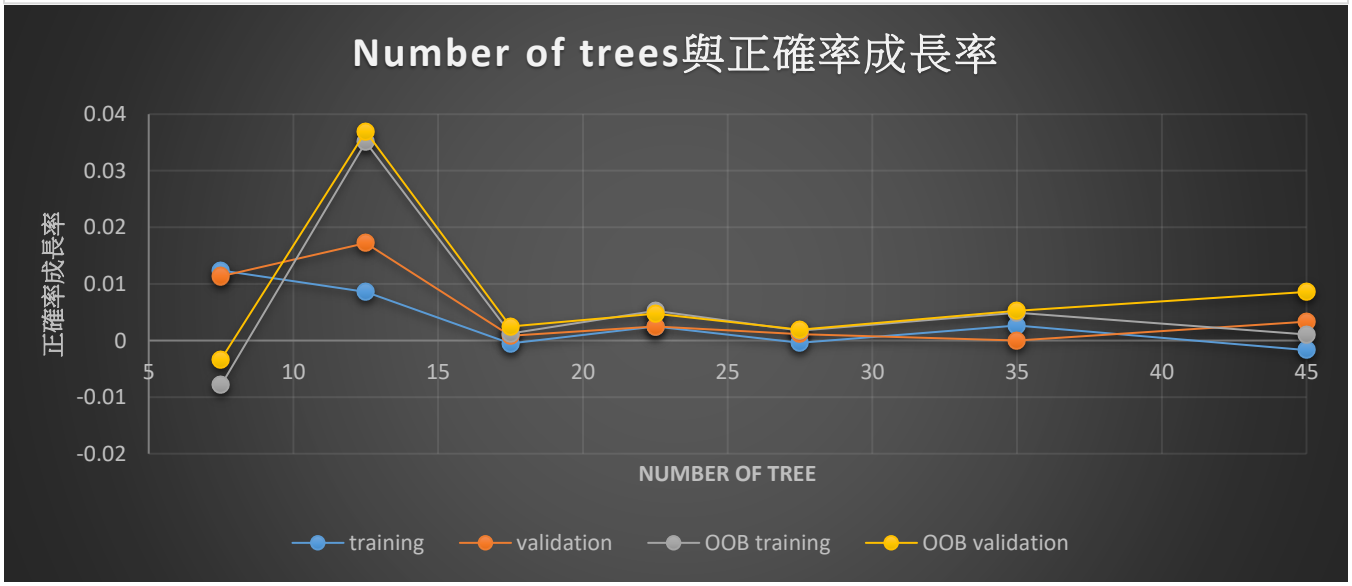
1. Size of Training and Validation 與 correct classification rates 的關係：

   變因：train 在整體的 percentage。

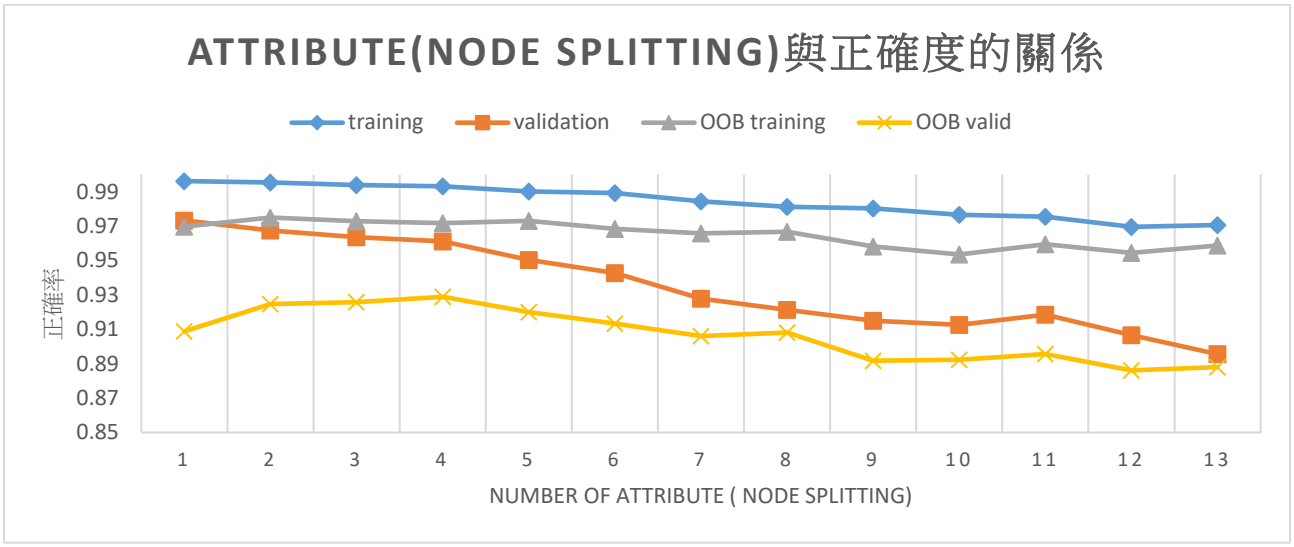


2. Number of trees in the forest 與 correct classification rates 的關係：



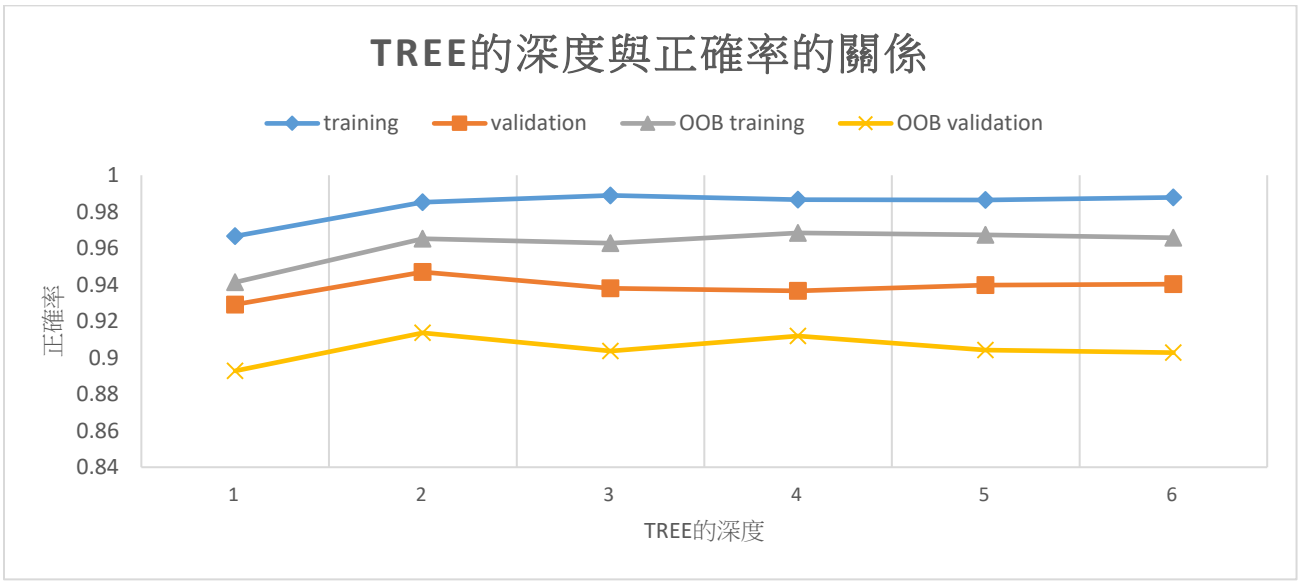


3. Number of attributes to consider at each node splitting 與 correct classification rates 的關係：



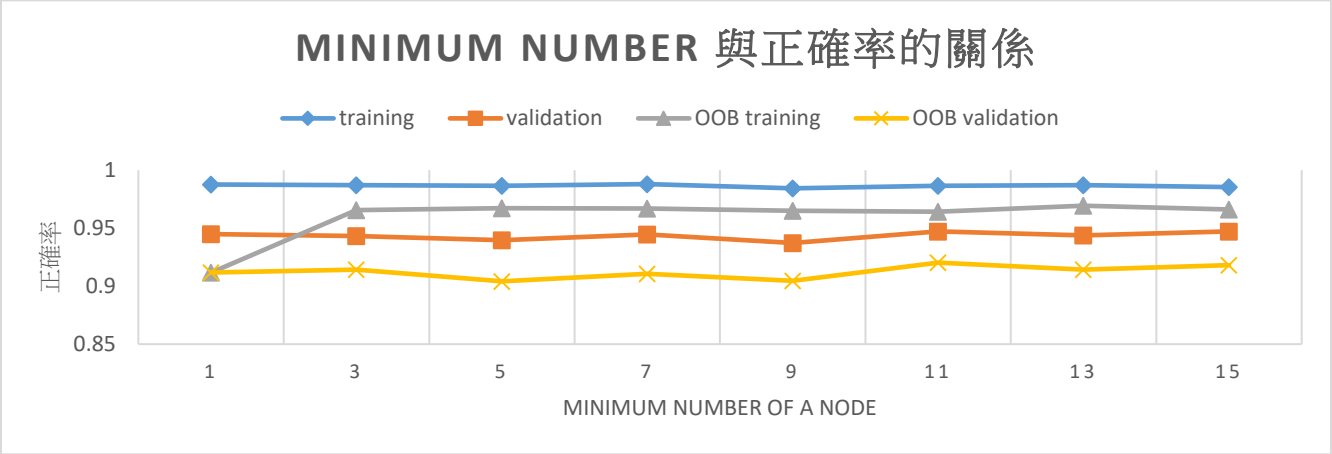
4. 樹的 max depth：

在檢查的過程發現其實跑到 depth 3 或是 4 的時候，樹就已經完整建立好了。

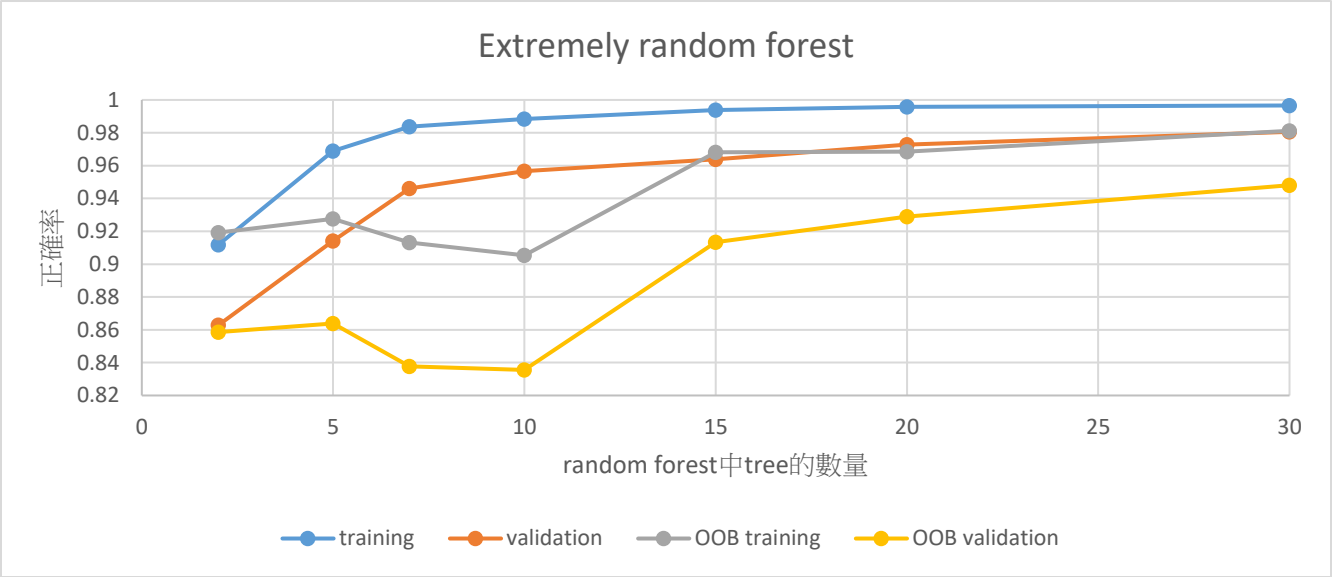



```
gini index:0.384335 [Proline] < 760 length:113
Left
  gini index:0.0990476 [OD280/OD315 of diluted wines] < 2.19 length:70
Left
    gini index:0.0641975 [Alcalinity of ash] < 17.15 length:30
Left
      depth 3. terminal class:2
Right
      depth 3. terminal class:3
Right
    depth 2. terminal class:2
Right
  gini index:0.123137 [Flavanoids] < 2.165 length:43
Left
    depth 2. terminal class:3
Right
    depth 2. terminal class:1
```

5. 樹的 minimum



6. Extremely random forest 中 attribute 為 random 給的，去比較跟 random forest 中的 tree 數量成長有沒有影響到 correct rate：



- 結果分析：

1. 先講一下這次測試的 dataset - wine，可以從測試的時候去得知說是還蠻容易就學習的 dataset，連只給 30%data 去 train 就可以有 80%以上的成功率了，還有在建樹的過程中，只需要大約 3、4 層就可以建好了。
2. Train 在整體 data 占比例占的越少，正確率越低。但是這裡蠻訝異的一點就是不會有比例較大(0.9)，導致 validation data overfitting 的問題，我猜想應該是因為 dataset 太簡單太好預測了。
3. Number of tree in random forest 在 5~20 顆時，正確度成長率很快，差不多在近 20 顆時到達平衡，樹的多寡不會導致 overfitting 反而是正確率增加、錯誤率變小。
4. Number of attribute in node splitting 這邊當 attribute = 1 時為 extreme random

forest tree 特殊情形，可以看到 attribute 數量越多正確率越低，是因為 Attribute 給的越多，相對而言樹的 diversity 就越低，因為每一顆樹在選的時候都會選擇最好的 attribute 去長，所以使得最後投票的時候偏好類似，導致正確率下降。

5. 樹的 tree depth，正確率主要在 1~3 階段成長性增加而後穩定，因為在後圖可以看出其實因為這個 dataset 蠻簡單的，所以在建 tree 時主要在 3、4 就已經完成了。
6. Minimum number of a node 可以看出沒有太大的幅度，主要應該是因為 dataset 本身的資料 class 本來差異就蠻大。
7. Extremely random forest tree 可以看到大約正確率在~20 顆 tree 的時候大幅度的增加，就可以把 random 給 attribute 的正確率給恢復像 attribute 6 個時的正常情況，在只有 2~5 顆的時候可以看出正確率相較於 20 顆蠻低的。

- 心得：

這次自建 random forest + base tree CART 的實驗讓我收穫蠻多的，以前我有寫過 python 直接使用套建的結果，但那時候常常定義、參數還是會有點模糊，不是很清楚裡面套件裏頭主要是在做甚麼，透過這次的實驗，可以很清楚的知道每個步驟在做甚麼，而且在寫之前原本覺得應該會困難重重，結果還行，而且預測出來的模型也蠻佳的，所以還蠻開心的，這些概念對我未來使用套件會對於裏頭的參數有更多的了解，希望也可以預測模型更佳。

- Reference：

只有讀檔 string 處理有使用到 source code。

```cpp
#include <iostream>
#include <string>
#include <stdio.h>
#include <stdlib.h>
#include <sstream>
#include <fstream>
#include <time.h> // time
#include <math.h>
#include <algorithm> // std::random_shuffle
#include <vector>
#define N 178
#define type 13

using namespace std;

float data[N][type+1];
float train[N][type+1];
float valid[N][type+1];

int TN,VN;

string printAttr[] = {"class",
 "Alcohol",
 "Malic acid",
 "Ash",
 "Alcalinity of ash",
 "Magnesium",
 "Total phenols",
 "Flavanoids",
 "Nonflavanoid phenols",
 "Proanthocyanins",
 "Color intensity",
 "Hue",
"OD280/OD315 of diluted wines",
"Proline"};

void print_data(float input[][type+1], int n){
    for (int j = 0 ; j < n ; j ++) {
        for ( int i = 0 ; i < type+1 ; i ++)
            cout << input[j][i] <<",";
        cout << endl;
    }
}
vector<string> _csv(string);
void load_data(string);

// split data to training / validation
void split_data(float p){
    int n = N * p;
    TN = n;
    VN = N - TN;
    //cout << "train N:" << FN << endl;
    //cout << "Valid N:" << VN << endl;
    random_shuffle(data,data+N);

    for (int i = 0 ; i < TN ; i ++)
        for (int j = 0 ; j < type+1 ; j ++)
            train[i][j] = data[i][j];

    int Vindex = 0;
    for (int i = TN ; i < N ; i ++) {
```

```
62.         for ( int j = 0 ; j < type+1 ; j ++)
63.             valid[Vindex][j] = data[i][j];
64.         Vindex++;
65.     }
66. }
67.
68. float gini_index(float input[][type+1],int n){
69.     int count[3]={0}; // 分別計算 class 1,2,3
70.     float gini = 0;
71.     for( int i = 0 ; i < n ; i ++) {
72.         count[ int(input[i][0]-1)] ++;
73.     }
74.     for (int i = 0 ; i < 3 ; i ++) {
75.         gini += pow(count[i]*1.0/n,2);
76.     }
77.     gini = 1.0 - gini;
78.     return gini;
79. }
80.
81. struct split_info{
82.     int attribute;
83.     float threshold;
84.     float gini;
85.     split_info(void){
86.         gini = 1.0;
87.     }
88. };
89.
90. class CART{
91.     private:
92.         float data[N][type+1];
93.         int length;
94.         int num_attr; // num of random attribute
95.     public:
96.         CART *Left;
97.         CART *Right;
98.         bool is_terminal;
99.         int end_class;
100.          struct split_info result; // save best attr, threshold, gini
101.
102.          CART(float[][type+1],int l); // constructor
103.          int get_length();
104.          void set_num_attr(int); // set num_attr
105.          float test_split(int,float);
106.          struct split_info select_threshold(int);  // find best value (lowest gini) according
    to threshold
107.          void find_best_attr(); // select the threshold with the lowest total impurity
108.          void get_split();
109.          bool same_class();
110.          int to_terminal();
111.  };
112.
113.  CART::CART(float input[][type+1],int l) {
114.      for (int i = 0 ; i < l ; i ++) {
115.          for(int j = 0 ; j < type+1 ; j ++)
116.              data[i][j] = input[i][j];
117.      }
118.      length = l;
119.      is_terminal = false;
120.  }
121.
122.  int CART::get_length() {
123.      return length;
124.  }
```

```
125.
126.  float CART::test_split(int attr,float threshold){
127.      float left[N][type+1];
128.      float right[N][type+1];
129.      int Lindex = 0;
130.      int Rindex = 0;
131.      for (int i = 0 ; i < length ; i ++) {
132.          if (data[i][attr] < threshold) {
133.              for ( int j = 0 ; j < type +1 ; j ++)
134.                  left[Lindex][j] = data[i][j];
135.              Lindex ++;
136.          }
137.          else {
138.              for (int j = 0 ; j < type+1 ; j ++)
139.                  right[Rindex][j] = data[i][j];
140.              Rindex ++;
141.          }
142.      }
143.      float gini = Lindex*1.0/length*gini_index(left,Lindex) + Rindex*1.0/length*gini_index(rig
    ht,Rindex);
144.      return gini;
145.  }
146.
147.  void CART::set_num_attr(int n) {
148.      num_attr = n;
149.  }
150.
151.  // if threshold = 1 代表 Alcohol
152.  struct split_info CART::select_threshold(int attr){
153.      // th 存放這 attribute 所有值
154.      struct split_info best;
155.      best.gini = 1.0;
156.      vector<float>th;
157.      for (int i = 0 ; i < length ; i ++)
158.          th.push_back(data[i][attr]);
159.      sort(th.begin(),th.begin()+length);
160.
161.      for (int i = 0 ; i < length-1 ; i ++) {
162.          float threshold = (th[i] + th[i+1]) / 2.0;
163.          float resultgini = test_split(attr,threshold);
164.          if(resultgini < best.gini){
165.              best.threshold = threshold;
166.              best.gini = resultgini;
167.          }
168.      }
169.      best.attribute = attr;
170.      return best;
171.  }
172.
173.  //select attribute from attr. bag
174.  void CART::find_best_attr(){
175.      int select [type+1];
176.      for (int i = 0 ; i < type; i ++)
177.          select[i] = i+1;
178.      random_shuffle(select,select+type);
179.
180.      for (int i = 0 ; i < num_attr ; i ++) {
181.          struct split_info tmp;
182.          //select threshold of the attribute
183.          tmp = select_threshold(select[i]);
184.          //cout<<"attribute:"<<i<<endl;
185.          if (tmp.gini < result.gini) {
186.              result = tmp;
187.          }
```

```
    }
}

// check whether data in this class are the same class;
bool CART::same_class() {
    int first = data[0][0];
    for (int i = 0 ; i < length ; i ++) {
        if (data[i][0] != first)
            return false;
    }
    is_terminal = true;
    end_class = first;
    return true;
}

// find the best attribute -> produce left & rihgt CART
void CART::get_split(){
    set_num_attr(6);
    find_best_attr();
    float left[N][type+1];
    float right[N][type+1];
    int Lindex = 0;
    int Rindex = 0;
    for (int i = 0 ; i < length ; i ++) {
        if (data[i][result.attribute] < result.threshold) {
            for ( int j = 0 ; j < type +1 ; j ++)
                left[Lindex][j] = data[i][j];
            Lindex ++;
        }
        else {
            for (int j = 0 ; j < type+1 ; j ++)
                right[Rindex][j] = data[i][j];
            Rindex ++;
        }
    }
    Left = new CART(left,Lindex);
    Right = new CART(right,Rindex);
}


int CART::to_terminal() {
    int count[3] = {0}; // calculate type
    for (int i = 0 ; i < length ; i ++) {
        count[ int(data[i][0]-1) ] ++;
    }
    int maxx = count[0];
    int ctype = 1;
    for (int i = 1 ; i < 3 ; i ++) {
        if(count[i] > maxx) {
            ctype = i+1;
            maxx = count[i];
        }
    }
    is_terminal = true;
    end_class = ctype;
    return ctype;
}

void split(CART *r, int max_depth, int min_size , int depth) {
    //cout<<" Root split "<<endl;
    r->get_split();
    // check for a no split
    if(r->result.gini == 0) {
        r->to_terminal();
```

```
252.          return;
253.      }
254.      // check for max depth
255.      if (depth >= max_depth) {
256.          r->Left->to_terminal();
257.          r->Right->to_terminal();
258.          return;
259.      }
260.      // process left child
261.      if (r->Left->get_length() <= min_size)
262.          r->Left->to_terminal();
263.      else {
264.          r->Left->get_split();
265.          split(r->Left, max_depth, min_size, depth+1);
266.      }
267.      // process Right child
268.      if (r->Right->get_length() <= min_size)
269.          r->Right->to_terminal();
270.      else {
271.          r->Right->get_split();
272.          split(r->Right, max_depth, min_size, depth+1);
273.      }
274.  }
275.
276.  void print_tree(CART *root,int depth){
277.      if ( root->is_terminal == false) {
278.          for(int i = 0 ; i < depth ; i ++)
279.              cout<<"  ";
280.          cout<<"gini index:"<< root->result.gini;
281.          cout <<" ["<< printAttr[root->result.attribute]<<"] < " << root-
    >result.threshold << " length:" << root->get_length() << endl;
282.          cout << "Left" << endl;
283.          print_tree(root->Left,depth+1);
284.          cout << "Right" << endl;
285.          print_tree(root->Right,depth+1);
286.      }
287.      else{
288.          for(int i = 0 ; i < depth ; i ++)
289.              cout<<"  ";
290.          cout << "depth " << depth <<". ";
291.          cout << "terminal class:" <<  root->end_class << endl;
292.      }
293.  }
294.
295.  CART build_tree(float TRAIN[][type+1],int l, int max_depth, int min_size) {
296.      CART root(TRAIN,l);
297.      split(&root,max_depth,min_size,0);
298.      return root;
299.  }
300.
301.  int predict(CART *root, float input[type+1]) {
302.      CART *cursor = root;
303.      while ( !cursor->is_terminal ) {
304.          // < go to left
305.          if( input[cursor->result.attribute] < cursor->result.threshold)
306.              cursor = cursor->Left;
307.          else
308.              cursor = cursor->Right;
309.      }
310.      return cursor->end_class;
311.  }
312.
313.  void print_result(vector<CART> RF,float input[][type+1],int n) {
314.      int success = 0;
```

```
315.      for(int i = 0 ; i < n ; i ++) {
316.          int classCount[3] = {0};
317.          for (int j = 0 ; j < RF.size() ; j ++) {
318.              CART current = RF[j];
319.              classCount[ predict(¤t,input[i]) - 1 ] ++;
320.          }
321.          // select most count to be the answer
322.          int maxx = 0;
323.          int finalclass;
324.          for(int a = 0 ; a < 3 ; a ++) {
325.              if(classCount[a] > maxx ) {
326.                  maxx = classCount[a];
327.                  finalclass = a+1; // 1 ~ 3
328.              }
329.          }
330.          if(input[i][0] == finalclass)
331.              success++;
332.      }
333.      cout << "success rate:" << success*1.0/n <<endl;
334.  }
335.
336.  int main()
337.  {
338.      // loading data
339.      load_data("wine.data");
340.      // divide dataset into training & validation
341.      split_data(0.7);
342.
343.      // build K tree
344.      int tree_num = 200;
345.      vector<CART> random_forest;
346.      for(int i = 0 ; i < tree_num ; i ++) {
347.          // split train to 0.8
348.          random_shuffle(train,train+TN);
349.          int IN = TN * 0.8;
350.          float in[N][type+1];
351.          for (int a = 0 ; a < IN ; a ++ )
352.              for (int b = 0 ; b < type+1 ; b ++)
353.                  in[a][b] = train[a][b];
354.          // build each tree
355.          CART tmp = build_tree(in,IN,5,5);
356.          random_forest.push_back(tmp);
357.      }
358.      // Tree bagging
359.      int TB = 0.8*random_forest.size();
360.      vector<CART> selected(random_forest.begin(),random_forest.begin()+TB);
361.      vector<CART> OOB(random_forest.begin()+TB,random_forest.end());
362.
363.      // testing by using OOB err. or validation data set
364.      print_result(selected,train,TN);
365.      print_result(selected,valid,VN);
366.      cout<<"------------------OOB-----------------"<<endl;
367.      print_result(OOB,train,TN);
368.      print_result(OOB,valid,VN);
369.
370.      //print_tree(&root,0);
371.      system("pause");
372.      return 0;
373.  }
374.
375.  vector<string> _csv(string s){
376.      vector<string> arr;
377.      istringstream delim(s);
378.      string token;
```

```
379.      int c = 0;
380.      while (getline(delim,token,',')) {
381.          arr.push_back(token);
382.          c ++;
383.      }
384.      return arr;
385.  }
386.
387.  void load_data(string f) {
388.      ifstream inFile("wine.data");
389.      if (!inFile){
390.          cout << "檔案無法開啟\n";
391.          exit(1);
392.      }
393.      string line;
394.      int index = 0;
395.      while (getline(inFile,line)) {
396.          vector<string> a = _csv(line);
397.          for ( int i = 0 ; i < a.size() ; i ++)
398.              data[index][i] = atof(a[i].c_str());
399.          index ++;
400.      }
401.  }
```