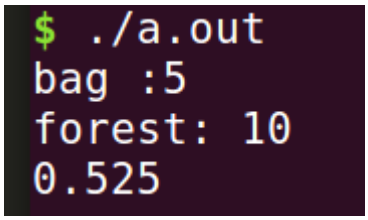
0616223 劉柏宇
人工智慧概論 程式作業 2 – CART

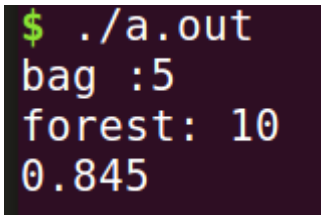
1.大致作法:

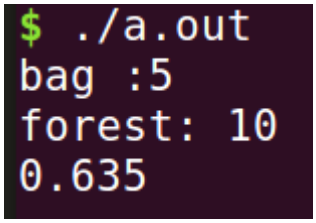
對原始 data 隨機選取點產生 decision tree, 重複並產生 forest。在選擇 decision tree 的 attribute 時, 用 gini's impurity 來決定要取哪個 threshold (x >= ? or y >= ?)。產生 forest 後對每個原始資料進行錯誤率計算, 把每個點都放進 forest 來投票, 決定該點是 class 1 還是 class 2, 再與正確資料相比, 即可得到正確率。

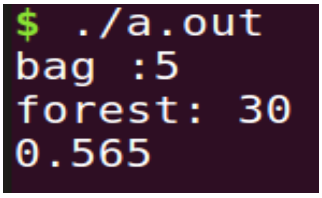
2. 實驗結果:

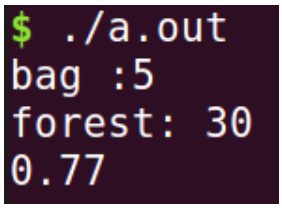
1. cross200.txt 的測試結果:

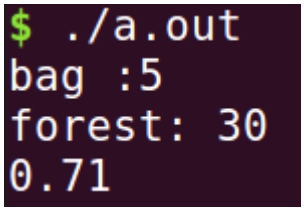


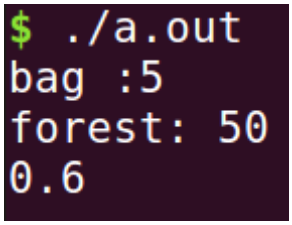


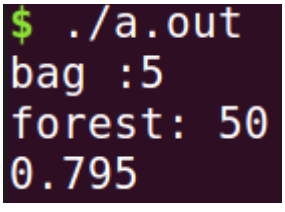


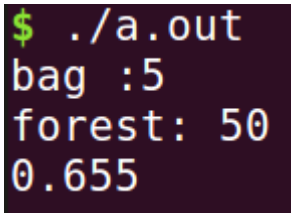


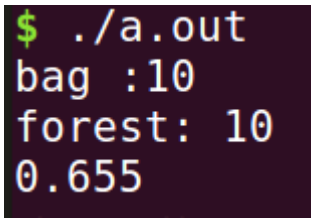


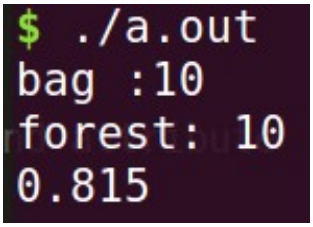


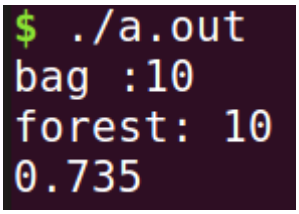


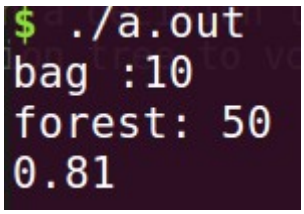


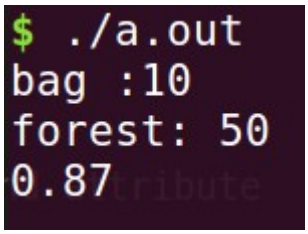


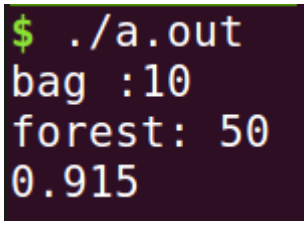


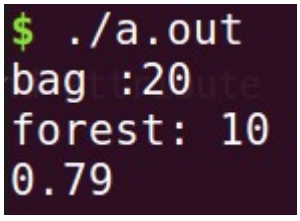


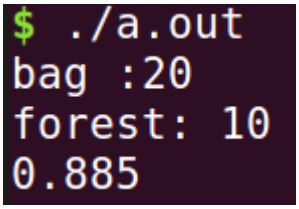


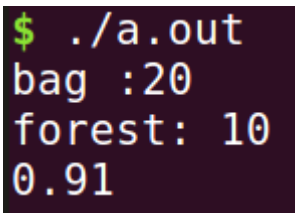


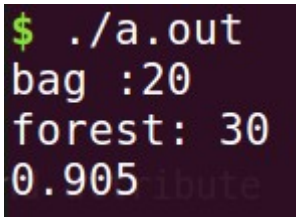


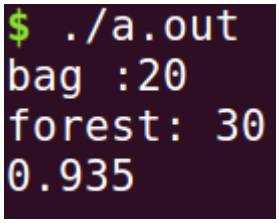


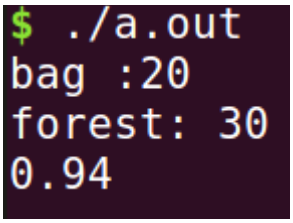




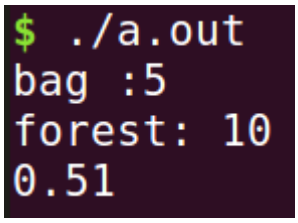


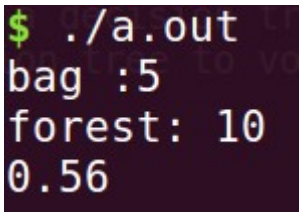


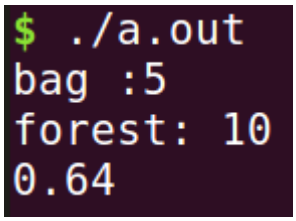


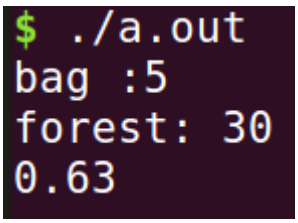


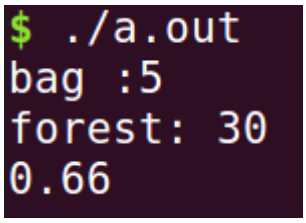
2. ellipse100.txt 的測試結果:

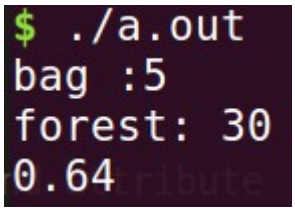


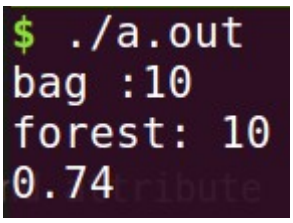


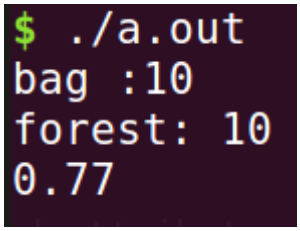


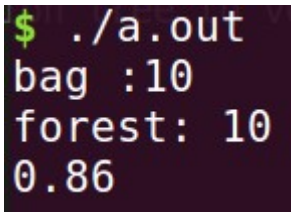


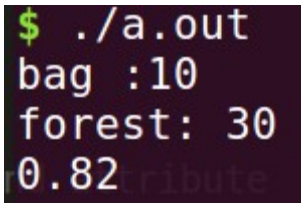


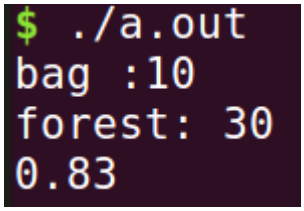


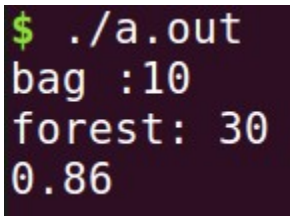


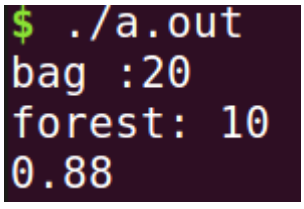


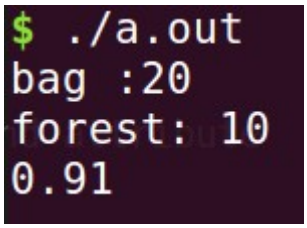


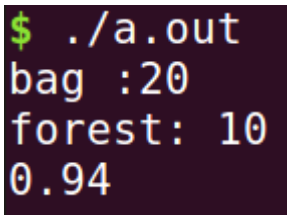


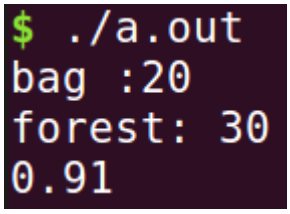


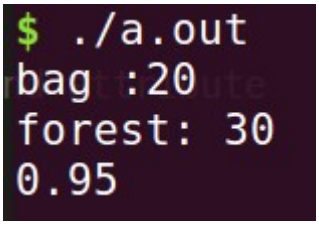


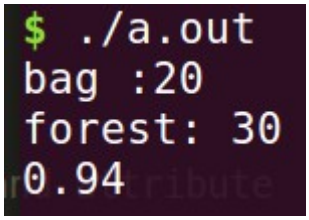












3. 實驗探討:

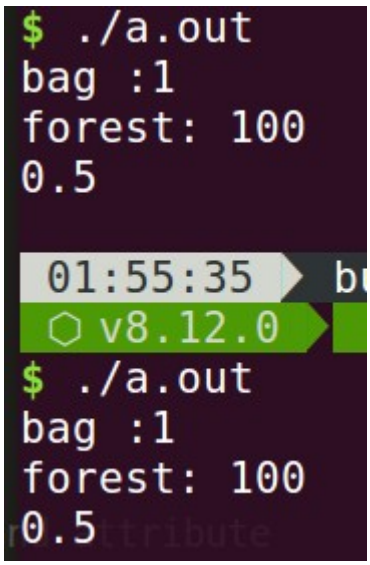
從實驗結果互相比對, 提高 forest 的 tree 的數量, 會讓正確率些微上升, 且整體的正確率範圍會比較集中(較穩定)(很多次紀錄的結果, 但這樣圖太多了無法截圖)。而提高每次做 decision tree 的 point 數, 可以有效提高正確率。

而我計算正確率的方式, 是每次隨機取 bag(ex: bag = 10)個點, 去 train 一棵 decision tree, 因此每棵 tree 的 validation 都不一樣, 因此採取 forest 投票的方式, 來決定所有的 point 是屬於哪個 class, 藉此來判斷正確率。

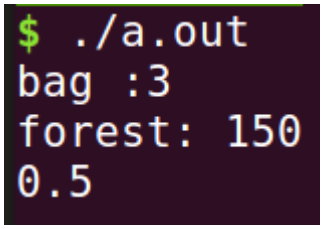
attribute 的選擇是挑選 x 或 y 的邊界值來決定要在左邊還是右邊, 例如 x >= 3 的都在右邊, 其他就跑到左邊, 所以如果可以跑圖形出來應該會是階梯狀的分隔線。如果把 forest 的分隔線全部加起來平均, 形成曲線, 跑在圖上的點的分佈的正確率, 應該會跟投票出來的結果一樣（因為投票結果造成的影響, 應該跟平均後的那個點的位置一樣, 也就是如果投票完是 51:49 , 那那個點應該會在離分隔線不遠的位置但在分隔線之上, 如果是 80:20, 那就毫無疑問是在上面）。

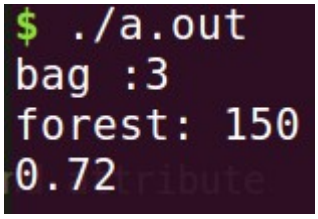
4. 特殊情況實驗

1. 當 bag 只有 1, 正確率是 0.5（畢竟是經過挑選的 attribute, 0.5 為分一種的最高正確率）。

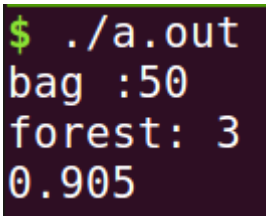


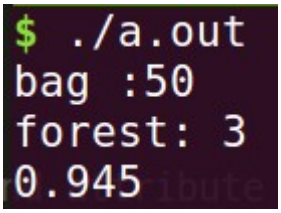
2. 當 forest >> bag, 實驗出來的結果浮動值變得很大, 並沒有因為 forest 增加而更精準(overfit)





3. 當 bag >> forest, 好像就只是單純的沒那麼準確, 沒有 50 個 bag 應該有的水準。





附錄 code:

```cpp
#include<bits/stdc++.h>
using namespace std;

int bag = 50; // the number of point to train a decision tree
int forest = 3; // the number of the decision tree to vote
int datasize = 200; // how many point

struct node{
  float x, y;
  int category;
};  // point's x, y and class

struct Tree{
  vector<node > data;
  char direction;
  float position;
  int final;
};  // decision tree's node to store point and attribute

bool comparex(node a, node b){
    if(a.x < b.x){
      return true;
    }else if(a.x == b.x){
      return (a.y < b.y);
    }else{
      return false;
    }
} // to sort x

bool comparey(node a, node b){
    if(a.y < b.y){
      return true;
    }else if(a.y == b.y){
      return (a.x < b.x);
    }else{
      return false;
    }
} // to sort y
```

```cpp
float findgini(vector<node > &data, char &direction){ // for decision tree's node to decide
                                                      //which attribute to choose
  int total = data.size();
  float min = 100;
  float seperate; // the threshold

  sort(data.begin(),data.end(), comparex);
  for(int i=0; i<total-1; i++){
    float bipart = (data[i].x + data[i+1].x) / 2;
    float left = 0;
    float left1 = 0;
    float left2 = 0;
    float right = 0;
    float right1 = 0;
    float right2 = 0;
    for(int j=0; j<total; j++){
      if(data[j].x >= bipart) {
        right++;
        if(data[j].category == 1) right1++;
        else right2++;
      }else{
        left++;
        if(data[j].category == 2) left1++;
        else left2++;
      }
    }
    float gini left = 1 - (left1/left)*(left1/left) - (left2/left)*(left2/left);
    float giniright = 1 - (right1/right)*(right1/right) - (right2/right)*(right2/right);
    float gini = ginileft*(left/total) + giniright*(right/total);
    if(gini < min){
      min = gini;
      seperate = bipart;
      direction = 'x';
    }
  } // search threshold of x by gini
```

```
  } // search threshold of x by gini

  sort(data.begin(),data.end(), comparey);
  for(int i=0; i<total-1; i++){
    float bipart = (data[i].y + data[i+1].y) / 2;
    float left = 0;
    float left1 = 0;
    float left2 = 0;
    float right = 0;
    float right1 = 0;
    float right2 = 0;
    for(int j=0; j<total; j++){
      if(data[j].y >= bipart) {
        right++;
        if(data[j].category == 1) right1++;
        else right2++;
      }else{
        left++;
        if(data[j].category == 2) left1++;
        else left2++;
      }
    }
    float ginileft = 1 - (left1/left)*(left1/left) - (left2/left)*(left2/left);
    float giniright = 1 - (right1/right)*(right1/right) - (right2/right)*(right2/right);
    float gini = ginileft*(left/total) + giniright*(right/total);
    if(gini < min){
      min = gini;
      seperate = bipart;
      direction = 'y';
    }
  } // search threshold of y by gini, choose the smallist one compare to x

  return seperate;
}
```

```
void buildtree(vector<Tree > &DT){  // build decision tree by vector
  for(int i=0; i<bag; i++){
    if(DT[i].data.size() == 0){ // no data == no such child root
      continue;
    }else{
      bool same = true;
      for(int j=0; j<DT[i].data.size()-1; j++){
        if(DT[i].data[j].category != DT[i].data[j+1].category) same = false;
      }
      if(same){ // doesn't need to seperate
        DT[i].final = DT[i].data[0].category;
      }else{
        char direction;
        float seperate;
        seperate = findgini(DT[i].data, direction);
        if(direction == 'x'){
          for(int j=0; j<DT[i].data.size(); j++){
            if(DT[i].data[j].x >= seperate){
              DT[2*i+2].data.push_back(DT[i].data[j]);
            }else{
              DT[2*i+1].data.push_back(DT[i].data[j]);
            }
          }
        }else{
          for(int j=0; j<DT[i].data.size(); j++){
            if(DT[i].data[j].y >= seperate){
              DT[2*i+2].data.push_back(DT[i].data[j]);
            }else{
              DT[2*i+1].data.push_back(DT[i].data[j]);
            }
          }
        }
        DT[i].direction = direction;
        DT[i].position = seperate;
      } // use gini to seperate and memorize the threshold
    }
  }
}
```

```
float depend(node &test, vector<vector<Tree > > &DT){ // use to vote a point is in class 1 or 2
  int vote1 = 0;
  int vote2 = 0;
  for(int i=0; i<forest; i++){
    for(int j=0; j<50 ;){
      if(DT[i][j].final == 1){ // semms to be class 1
        vote1++;
        break;
      }else if(DT[i][j].final == 2){  //seems to be class 2
        vote2++;
        break;
      }else if(DT[i][j].direction == 'x'){
        if(test.x >= DT[i][j].position){  // right child
          j = j*2+2;
        }else{  // left child
          j = j*2+1;
        }
      }else if(DT[i][j].direction == 'y'){
        if(test.y >= DT[i][j].position){  // right child
          j = j*2+2;
        }else{  // left child
          j = j*2+1;
        }
      }else{  // can't decide (actually i think it's a kind of bug, but i have no idea about it)
        break;
      }
    }
  }
  if(vote1 >= vote2) return 1;
  else return 2;
}
```

```
int main(){
  srand( time(NULL)); //  set random
  //ifstream fin("ellipse100.txt", ios::in);  // read data
  ifstream fin("cross200.txt", ios::in);  // read data
  vector<node > nodelist; // store point
  string str;
  while(getline(fin, str)){
    istringstream iss(str);
    node temp;
    iss >> temp.x;
    iss >> temp.y;
    iss >> temp.category;
    nodelist.push_back(temp);
  }   // read data

  vector<vector<Tree > > DT(forest, vector<Tree>(1000)); // decision tree forest

  for(int i=0; i<forest; i++){
    int random[datasize];
    for(int j=0; j<datasize; j++){
      random[j] = j;
    }
    for(int j=0; j<datasize; j++){
      int pos = rand() % datasize;
      int temp;
      temp = random[j];
      random[j] = random[pos];
      random[pos] = temp;
    } // get random number set


    for(int j=0; j<bag; j++){
      DT[i][0].data.push_back(nodelist[random[j]]);
    } // random point at root for each decision tree

    buildtree(DT[i]); // build it!!
  }
```

```
218     float fitrate;
219     float fitnode = 0;
220     for(int i=0; i<datasize; i++){
221       if(nodelist[i].category == depend(nodelist[i], DT)) fitnode++;
222     } // test each node to see if it fit the result of vote
223     fitrate = fitnode / datasize; // compute error rate
224     cout << "bag :" << bag << endl;
225     cout << "forest: " << forest << endl;
226     cout << fitrate << endl;
227 
228     return 0;
229 }
```