

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 ($x \geq ?$ or $y \geq ?$)。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:

| | | |
|--|--|--|
| <pre>\$./a.out bag :5 forest: 10 0.525</pre> | <pre>\$./a.out bag :5 forest: 10 0.845</pre> | <pre>\$./a.out bag :5 forest: 10 0.635</pre> |
| <pre>\$./a.out bag :5 forest: 30 0.565</pre> | <pre>\$./a.out bag :5 forest: 30 0.77</pre> | <pre>\$./a.out bag :5 forest: 30 0.71</pre> |
| <pre>\$./a.out bag :5 forest: 50 0.6</pre> | <pre>\$./a.out bag :5 forest: 50 0.795</pre> | <pre>\$./a.out bag :5 forest: 50 0.655</pre> |
| <pre>\$./a.out bag :10 forest: 10 0.655</pre> | <pre>\$./a.out bag :10 forest: 10 0.815</pre> | <pre>\$./a.out bag :10 forest: 10 0.735</pre> |
| <pre>\$./a.out bag :10 forest: 50 0.81</pre> | <pre>\$./a.out bag :10 forest: 50 0.87</pre> | <pre>\$./a.out bag :10 forest: 50 0.915</pre> |
| <pre>\$./a.out bag :20 forest: 10 0.79</pre> | <pre>\$./a.out bag :20 forest: 10 0.885</pre> | <pre>\$./a.out bag :20 forest: 10 0.91</pre> |

```
$ ./a.out
bag :20
forest: 30
0.905
```

```
$ ./a.out
bag :20
forest: 30
0.935
```

```
$ ./a.out
bag :20
forest: 30
0.94
```

2. ellipse100.txt 的測試結果:

```
$ ./a.out
bag :5
forest: 10
0.51
```

```
$ ./a.out
bag :5
forest: 10
0.56
```

```
$ ./a.out
bag :5
forest: 10
0.64
```

```
$ ./a.out
bag :5
forest: 30
0.63
```

```
$ ./a.out
bag :5
forest: 30
0.66
```

```
$ ./a.out
bag :5
forest: 30
0.64
```

```
$ ./a.out
bag :10
forest: 10
0.74
```

```
$ ./a.out
bag :10
forest: 10
0.77
```

```
$ ./a.out
bag :10
forest: 10
0.86
```

```
$ ./a.out
bag :10
forest: 30
0.82
```

```
$ ./a.out
bag :10
forest: 30
0.83
```

```
$ ./a.out
bag :10
forest: 30
0.86
```

```
$ ./a.out
bag :20
forest: 10
0.88
```

```
$ ./a.out
bag :20
forest: 10
0.91
```

```
$ ./a.out
bag :20
forest: 10
0.94
```

```
$ ./a.out
bag :20
forest: 30
0.91
```

```
$ ./a.out
bag :20
forest: 30
0.95
```

```
$ ./a.out
bag :20
forest: 30
0.94
```

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 $x \geq 3$ 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)。

```
$ ./a.out
bag :1
forest: 100
0.5
```

01:55:35 ▶ b

v8.12.0 ▶

```
$ ./a.out
bag :1
forest: 100
0.5
```

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**)

```
$ ./a.out
bag :3
forest: 150
0.5
```

```
$ ./a.out
bag :3
forest: 150
0.72
```

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。

```
$ ./a.out
bag :50
forest: 3
0.905
```

```
$ ./a.out
bag :50
forest: 3
0.945
```

附錄 code:

```
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int bag = 50; // the number of point to train a decision tree
5  int forest = 3; // the number of the decision tree to vote
6  int datasize = 200; // how many point
7
8  struct node{
9      float x, y;
10     int category;
11 }; // point's x, y and class
12
13 struct Tree{
14     vector<node > data;
15     char direction;
16     float position;
17     int final;
18 }; // decision tree's node to store point and attribute
19
20 bool comparex(node a, node b){
21     if(a.x < b.x){
22         return true;
23     }else if(a.x == b.x){
24         return (a.y < b.y);
25     }else{
26         return false;
27     }
28 } // to sort x
29
30 bool comparey(node a, node b){
31     if(a.y < b.y){
32         return true;
33     }else if(a.y == b.y){
34         return (a.x < b.x);
35     }else{
36         return false;
37     }
38 } // to sort y
39
40 float findgini(vector<node > &data, char &direction){ // for decision tree's node to decide
41     //which attribute to choose
42     int total = data.size();
43     float min = 100;
44     float separete; // the threshold
45
46     sort(data.begin(),data.end(), comparex);
47     for(int i=0; i<total-1; i++){
48         float bipart = (data[i].x + data[i+1].x) / 2;
49         float left = 0;
50         float left1 = 0;
51         float left2 = 0;
52         float right = 0;
53         float right1 = 0;
54         float right2 = 0;
55         for(int j=0; j<total; j++){
56             if(data[j].x >= bipart) {
57                 right++;
58                 if(data[j].category == 1) right1++;
59                 else right2++;
60             }else{
61                 left++;
62                 if(data[j].category == 2) left1++;
63                 else left2++;
64             }
65         }
66         float ginileft = 1 - (left1/left)*(left1/left) - (left2/left)*(left2/left);
67         float giniright = 1 - (right1/right)*(right1/right) - (right2/right)*(right2/right);
68         float gini = ginileft*(left/total) + giniright*(right/total);
69         if(gini < min){
70             min = gini;
71             separete = bipart;
72             direction = 'x';
73         }
74     } // search threshold of x by gini
75 }
```



```

74     } // search threshold of x by gini
75
76     sort(data.begin(), data.end(), comparey);
77     for(int i=0; i<total-1; i++){
78         float bipart = (data[i].y + data[i+1].y) / 2;
79         float left = 0;
80         float left1 = 0;
81         float left2 = 0;
82         float right = 0;
83         float right1 = 0;
84         float right2 = 0;
85         for(int j=0; j<total; j++){
86             if(data[j].y >= bipart) {
87                 right++;
88                 if(data[j].category == 1) right1++;
89                 else right2++;
90             }else{
91                 left++;
92                 if(data[j].category == 2) left1++;
93                 else left2++;
94             }
95         }
96         float ginileft = 1 - (left1/left)*(left1/left) - (left2/left)*(left2/left);
97         float giniright = 1 - (right1/right)*(right1/right) - (right2/right)*(right2/right);
98         float gini = ginileft*(left/total) + giniright*(right/total);
99         if(gini < min){
100             min = gini;
101             separte = bipart;
102             direction = 'y';
103         }
104     } // search threshold of y by gini, choose the smallest one compare to x
105
106     return separte;
107 }

```

```

109 void buildtree(vector<Tree> &DT){ // build decision tree by vector
110     for(int i=0; i<bag; i++){
111         if(DT[i].data.size() == 0){ // no data == no such child root
112             continue;
113         }else{
114             bool same = true;
115             for(int j=0; j<DT[i].data.size()-1; j++){
116                 if(DT[i].data[j].category != DT[i].data[j+1].category) same = false;
117             }
118             if(same){ // doesn't need to separte
119                 DT[i].final = DT[i].data[0].category;
120             }else{
121                 char direction;
122                 float separte;
123                 separte = findgini(DT[i].data, direction);
124                 if(direction == 'x'){
125                     for(int j=0; j<DT[i].data.size(); j++){
126                         if(DT[i].data[j].x >= separte){
127                             DT[2*i+2].data.push_back(DT[i].data[j]);
128                         }else{
129                             DT[2*i+1].data.push_back(DT[i].data[j]);
130                         }
131                     }
132                 }else{
133                     for(int j=0; j<DT[i].data.size(); j++){
134                         if(DT[i].data[j].y >= separte){
135                             DT[2*i+2].data.push_back(DT[i].data[j]);
136                         }else{
137                             DT[2*i+1].data.push_back(DT[i].data[j]);
138                         }
139                     }
140                 }
141                 DT[i].direction = direction;
142                 DT[i].position = separte;
143             } // use gini to separte and memorize the threshold
144         }
145     }
146 }

```

```

148 float depend(node &test, vector<vector<Tree> > &DT){ // use to vote a point is in class 1 or 2
149     int vote1 = 0;
150     int vote2 = 0;
151     for(int i=0; i<forest; i++){
152         for(int j=0; j<50; j++){
153             if(DT[i][j].final == 1){ // seems to be class 1
154                 vote1++;
155                 break;
156             }else if(DT[i][j].final == 2){ // seems to be class 2
157                 vote2++;
158                 break;
159             }else if(DT[i][j].direction == 'x'){
160                 if(test.x >= DT[i][j].position){ // right child
161                     j = j*2+2;
162                 }else{ // left child
163                     j = j*2+1;
164                 }
165             }else if(DT[i][j].direction == 'y'){
166                 if(test.y >= DT[i][j].position){ // right child
167                     j = j*2+2;
168                 }else{ // left child
169                     j = j*2+1;
170                 }
171             }else{ // can't decide (actually i think it's a kind of bug, but i have no idea about it)
172                 break;
173             }
174         }
175     }
176     if(vote1 >= vote2) return 1;
177     else return 2;
178 }

```

```

180 int main(){
181     srand( time(NULL)); // set random
182     //ifstream fin("ellipse100.txt", ios::in); // read data
183     ifstream fin("cross200.txt", ios::in); // read data
184     vector<node> nodelist; // store point
185     string str;
186     while(getline(fin, str)){
187         istringstream iss(str);
188         node temp;
189         iss >> temp.x;
190         iss >> temp.y;
191         iss >> temp.category;
192         nodelist.push_back(temp);
193     } // read data
194
195     vector<vector<Tree> > DT(forest, vector<Tree>(1000)); // decision tree forest
196
197     for(int i=0; i<forest; i++){
198         int random[datasize];
199         for(int j=0; j<datasize; j++){
200             random[j] = j;
201         }
202         for(int j=0; j<datasize; j++){
203             int pos = rand() % datasize;
204             int temp;
205             temp = random[j];
206             random[j] = random[pos];
207             random[pos] = temp;
208         } // get random number set
209
210
211         for(int j=0; j<bag; j++){
212             DT[i][0].data.push_back(nodelist[random[j]]);
213         } // random point at root for each decision tree
214
215         buildtree(DT[i]); // build it!!
216     }
217 }

```

```
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 }
```