Homework#6

B08902071 塗季芸

1. Answer: (b).

   The number of operations of $\delta_j^{(2)} = 1 \times 6$, and that of $\delta_j^{(1)} = 5 \times 6$. $6 + 30 = 36$.

2. Answer: (d).

```
#include <bits/stdc++.h>
using namespace std;
int mx=0;
int a[55];

void dfs(int cur,int L,int sum)
{
  //printf("cur=%d L=%d sum=%d\n",cur,L,sum);
  if(cur==L-1){

    a[cur]=sum+1;
    //int y=0;
    //for(int i=1;i<=L-1;i++) y+=a[i]+1;
    //printf("%d\n",y);
    //for(int i=1;i<=L-1;i++) a[i]++;
    int x=0;
    x+=20*a[1];
    for(int i=1;i<L-1;i++) x+=(a[i]+1)*a[i+1];
    x+=3*(a[L-1]+1);
    mx=max(mx,x);
    return;
  }
  for(int i=0;i<=sum;i++){
    //printf("cur=%d, i=%d, limit=%d\n",cur,i,sum-i-L+1+cur);
    a[cur]=i+1;
    dfs(cur+1,L,sum-i);
  }
}
int main()
{
  for(int L=2;L<=25;L++){
    printf("mx=%d\n",mx);
    dfs(1,L,50-2*L+2);
  }
}
```

3. Answer: (d).

$$\begin{aligned}
\frac{\partial err}{\partial s_k^{(L)}} &= \frac{\partial - \ln q_y}{\partial s_k^{(L)}} \\
&= \frac{\partial \ln(\sum_{k=1}^{K} \exp(s_k^{(L)})) - s_y^{(L)}}{\partial s_k^{(L)}} \\
&= \frac{\exp(s_k^{(L)})}{\sum_{k=1}^{K} \exp s_k^{(L)}} - v_k \\
&= q_k - v_k
\end{aligned}$$

4. Answer: (a).

   First iteration:

$x_i^{(l)}$

i / l	0	1	2
0	1	1	0
1	0	0	-
2	0	0	-
3	0	0	-
4	-	0	-

$\delta_j^{(l)}$

j / l	1	2
0	0	-2
1	0	-
2	0	-
3	0	-
4	0	-

$w_{ij}^{(l)}$

$(i,l)/j$	0	1	2	3	4
(0,1)	0	0	0	0	0
(1,1)	0	0	0	0	0
(2,1)	0	0	0	0	0
(3,1)	0	0	0	0	0
(0,2)	2	-	-	-	-
(1,2)	0	-	-	-	-
(2,2)	0	-	-	-	-
(3,2)	0	-	-	-	-
(4,2)	0	-	-	-	-

   Second iteration:

$x_i^{(l)}$

i / l	0	1	2
0	1	1	1
1	0	0	-
2	0	0	-
3	0	0	-
4	-	0	-

$\delta_j^{(l)}$

j / l	1	2
0	0	0
1	0	-
2	0	-
3	0	-
4	0	-

$w_{ij}^{(l)}$

$(i,l)/j$	0	1	2	3	4
(0,1)	0	0	0	0	0
(1,1)	0	0	0	0	0
(2,1)	0	0	0	0	0
(3,1)	0	0	0	0	0
(0,2)	2	-	-	-	-
(1,2)	0	-	-	-	-
(2,2)	0	-	-	-	-
(3,2)	0	-	-	-	-
(4,2)	0	-	-	-	-

   Third iteration is same as the second iteration.

5. Answer: (e).

   Our goal is to obtain the w_m^* such that $\sum_{i=n}^{N}(r_{im} - 2w_m^*)^2$ has the minimum. In high school, we learn that the solution of this problem is $2w_m^* = average(r_{im})$, which means w_m^* is half the average rating of the m-th movie.

6. Answer: (b).

$$\frac{\partial(r_{nm} - \mathbf{w}_m^T \mathbf{n}_n - a_m - b_n)^2}{\partial a_m} = -2(r_{nm} - \mathbf{w}_m^T \mathbf{n}_n - a_m - b_n)$$

$$a_m + \eta(r_{nm} - \mathbf{w}_m^T \mathbf{n}_n - a_m - b_n) = (1-\eta)a_m + \eta(r_{nm} - \mathbf{w}_m^T \mathbf{n}_n - b_n)$$

7. Answer: (d).

Define $a_0, a_1, ..., a_7$ as the probability of the following situations:

	a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7
$[\![g_1(\mathbf{x}) = f(\mathbf{x})]\!]$	1	1	1	0	1	0	0	0
$[\![g_2(\mathbf{x}) = f(\mathbf{x})]\!]$	1	1	0	1	0	1	0	0
$[\![g_3(\mathbf{x}) = f(\mathbf{x})]\!]$	1	0	1	1	0	0	1	0
$[\![G(\mathbf{x}) = f(\mathbf{x})]\!]$	1	1	1	1	0	0	0	0

Then, because $E_{out}(G) = 0.2$, we have

$$\begin{cases} a_0 + a_1 + a_2 + a_3 = 0.8 \\ a_4 + a_5 + a_6 + a_7 = 0.2 \\ a_3 + a_5 + a_6 + a_7 = E_{out}(g_1) \\ a_2 + a_4 + a_6 + a_7 = E_{out}(g_2) \\ a_1 + a_4 + a_5 + a_7 = E_{out}(g_3) \end{cases}$$

$$E_{out}(g_1) + E_{out}(g_2) + E_{out}(g_3) = 0.8 - a_0 + 0.4 + a_7 \geq 0.4$$

Therefore, the only possibility is (d).

8. Answer: (c).

   Since the five random variable is independent,

   $E_{out}(G)$
   $= \text{three classifiers misclassified} + \text{four classifiers misclassified} + \text{five classifiers misclassified}$
   $= \binom{5}{3}(0.4)^3(0.6)^2 + \binom{5}{4}(0.4)^4(0.6) + \binom{5}{5}(0.4)^5$
   ≈ 0.32

9. Answer: (b).

$$\begin{aligned}
\lim_{N\to\infty} \frac{(N-1)^{\frac{1}{2}N}}{N^{\frac{1}{2}N}} &= \lim_{N\to\infty} \frac{\binom{\frac{1}{2}N}{0}N^{\frac{1}{2}N} - \binom{\frac{1}{2}N}{1}N^{\frac{1}{2}N-1} + \binom{\frac{1}{2}N}{2}N^{\frac{1}{2}N-2} - ...}{N^{\frac{1}{2}N}} \\
&= 1 - \left(\frac{1}{2}\right) + \left(\frac{1}{2}\right)^2 \frac{1}{2!} - \left(\frac{1}{2}\right)^3 \frac{1}{3!} + ... \\
&= e^{-\frac{1}{2}} \\
&\approx 0.607
\end{aligned}$$

10. Answer: (e).

Because $g_{+1,i,\theta}(\mathbf{x})g_{+1,i,\theta}(\mathbf{x\prime}) = g_{-1,i,\theta}(\mathbf{x})g_{-1,i,\theta}(\mathbf{x\prime})$,

$$\begin{aligned}
(\phi_{ds}(\mathbf{x}))^T(\phi_{ds}(\mathbf{x\prime})) &= g_{+1,1,2L+1}(\mathbf{x})g_{+1,1,2L+1}(\mathbf{x\prime}) + ... + g_{-1,d,2R-1}(\mathbf{x})g_{-1,d,2R-1}(\mathbf{x\prime}) \\
&= 2\left(g_{+1,1,2L+1}(\mathbf{x})g_{+1,1,2L+1}(\mathbf{x\prime}) + ... + g_{+1,d,2R-1}(\mathbf{x})g_{+1,d,2R-1}(\mathbf{x\prime})\right) \\
&= 2\left(d(R-L) - 2\frac{|\mathbf{x}-\mathbf{x\prime}|}{2}\right) \text{(there are } \frac{|\mathbf{x}-\mathbf{x\prime}|}{2} \text{ negative items)} \\
&= 2d(R-L) - 2|\mathbf{x}-\mathbf{x\prime}|
\end{aligned}$$

11. Answer: (a).

By the lecture slides,

$$\begin{cases} u_+^{(2)} = u_+^{(1)} \cdot a_t \\ u_-^{(2)} = u_-^{(1)} / a_t \end{cases}$$

where $a_t = \sqrt{\frac{1-0.05}{0.05}}$and $u_+^{(1)} = u_-^{(1)} = \frac{1}{N}$.

Therefore,

$$\frac{u_+^{(2)}}{u_-^{(2)}} = a_t^2 = 19$$

12. Answer: (d).

Let $\epsilon_t = \frac{\sum_{n=1}^{N} u_n^{(t)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t)}}$, $a_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$, then

$$U_{t+1} = \frac{1}{N} \left(\sum_{n=1}^{N} u_n^{(t)} [\![y_n = g_t(\mathbf{x}_n)]\!] / a_t + \sum_{n=1}^{N} u_n^{(t)} [\![y_n \neq g_t(\mathbf{x}_n)]\!] \cdot a_t \right)$$

, and

$$U_t = \frac{1}{N} \sum_{n=1}^{N} u_n^{(t)}$$

$$\begin{aligned} \frac{U_{t+1}}{U_t} &= (1-\epsilon_t)/a_t + \epsilon_t \cdot a_t \\ &= 2\sqrt{\epsilon_t(1-\epsilon_t)} \\ &\leq 2\sqrt{\epsilon(1-\epsilon)} \end{aligned}$$

$$\begin{aligned} U_{T+1} &= \frac{U_{T+1}}{U_T} \cdot \frac{U_T}{U_{T-1}} \cdot \ldots \cdot \frac{U_2}{U_1} \quad (U_1 = \frac{1}{N} \sum_{n=1}^{N} 1 = 1) \\ &= T \cdot 2\sqrt{\epsilon(1-\epsilon)} \\ &\leq \exp(-2T(\frac{1}{2} - \epsilon)^2) \end{aligned}$$

13. Answer: (d).

$$1 - |\mu_+ - \mu_-| = 1 - |(1 - \min(\mu_+, \mu_-)) - \min(\mu_+, \mu_-)|$$

Because $2\min(\mu_+, \mu_-) \leq 1$,

$$1 - |(1 - \min(\mu_+, \mu_-)) - \min(\mu_+, \mu_-)| = 1 - |1 - 2\min(\mu_+, \mu_-)| = 2\min(\mu_+, \mu_-)$$

14. Answer: (c). 0.166

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

const double INF = 1e18;

struct Instance{
  double x[10];
  double y;
  int cmp;
  bool operator < (const Instance & q) const{
    return x[cmp]<q.x[cmp];
  }
};

struct DecisionTree{
  int i;
  double theta;
  DecisionTree *left, *right;
  DecisionTree(int _i, double _theta){
    i=_i;
    theta=_theta;
    left=NULL;
    right=NULL;
  }
};

DecisionTree* root;

double f(int cnt0, int cnt1, int pos0, int pos1)
{
  return cnt0*(1.0-pow(1.0*pos0/cnt0,2)-pow(1.0-1.0*pos0/cnt0,2))\
  +cnt1*(1.0-pow(1.0*pos1/cnt1,2)-pow(1.0-1.0*pos1/cnt1,2));
}

bool myunic(Instance a, Instance b){
  return a.x[a.cmp]==b.x[b.cmp];
}

void build(DecisionTree* &u, vector<Instance> instances)
{
  //cout << "building...\n";
  // termination condition
  int equ=1;
  for(int i=0;i<instances.size()-1;i++) if(instances[i].y!=instances[i+1].y)
   {
    equ=0;
    break;
  }
  if(equ){
    if(instances[0].y>0) u=new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }
  else equ=1;
  for(int i=0;i<instances.size()-1;i++){
    for(int j=0;j<10;j++){
      if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
    }
    if(!equ) break;
  }
```

```
60  if(equ){
61    int cnt[2]={0};
62    for(auto X: instances){
63      if(X.y>0) cnt[1]++;
64      else cnt[0]++;
65    }
66    if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
67    else u=new DecisionTree(0,INF);
68    return;
69  }
70
71  // learn branching criteria
72  double mn=INF;
73  double theta_star;
74  int i_star;
75  for(int i=0;i<10;i++){
76    for(auto &X: instances) X.cmp = i;
77    sort(instances.begin(),instances.end());
78    vector<Instance> sorted_xi;
79    sorted_xi = instances;
80    auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
81      sorted_xi.resize( distance(sorted_xi.begin(),it) );
82
83      int cnt[2][1005];
84      //cnt[0]: number of positive instances count from the front(included)
85      //cnt[1]: number of positive instances count from the back(included)
86      cnt[0][0]=sorted_xi[0].y>0?1:0;
87      for(int j=1;j<sorted_xi.size();j++){
88        cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
89      }
90      cnt[1][sorted_xi.size()]=0;
91      for(int j=sorted_xi.size()-1;j>=0;j--){
92        cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
93      }
94
95      int N = sorted_xi.size();
96      double a;
97      if((a=f(0,N,0,cnt[1][0]))<mn){
98        mn=a;
99        theta_star=sorted_xi[0].x[i]-5.0; // ???
100       i_star = i;
101     }
102     for(int j=0;j<N;j++){
103       if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
104         mn=a;
105         theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; //
    ???
106         i_star = i;
107       }
108     }
109 }
110
111 u=new DecisionTree(i_star,theta_star);
112
113 vector<Instance> left, right;
114 for(auto &X: instances) X.cmp = i_star;
115 sort(instances.begin(),instances.end());
116 for(auto X: instances){
117   if(X.x[i_star] < theta_star) left.push_back(X);
118   else right.push_back(X);
119 }
```

```
120
121   build(u->left,left);
122   build(u->right,right);
123 }
124
125 void dfs(DecisionTree* u)
126 {
127   if(u==NULL) return;
128   cout << "i=" << u->i << ", theta=" << u->theta << "\n";
129   dfs(u->left);
130   dfs(u->right);
131 }
132
133 double predict(DecisionTree* u, Instance X)
134 {
135   if(X.x[u->i] < u->theta){
136     if(!u->left) return -1.0;
137     else return predict(u->left,X);
138   } else {
139     if(!u->right) return 1.0;
140     else return predict(u->right,X);
141   }
142 }
143
144 int main()
145 {
146   ios::sync_with_stdio(false); cin.tie(0);
147   fstream trainfile("hw6_train.dat"), testfile("hw6_test.dat");
148   vector<Instance> instances;
149   string s;
150   while(getline(trainfile,s)){
151     Instance X;
152     stringstream ss(s);
153     for(int i=0;i<10;i++){
154       ss >> X.x[i];
155     }
156     ss >> X.y;
157     instances.push_back(X);
158   }
159   build(root, instances);
160
161   int err=0, tot=0;
162   while(getline(testfile,s)){
163     tot++;
164     Instance X;
165     stringstream ss(s);
166     for(int i=0;i<10;i++){
167       ss >> X.x[i];
168     }
169     ss >> X.y;
170     double y_pred = predict(root, X);
171     if(y_pred!=X.y) err++;
172   }
173   cout << 1.0*err/tot << "\n";
174 }
```

15. Answer: (d). 0.23

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

const double INF = 1e18;
const int trees = 2000;
default_random_engine generator;

struct Instance{
  double x[10];
  double y;
  int cmp;
  bool operator < (const Instance & q) const{
    return x[cmp]<q.x[cmp];
  }
};

struct DecisionTree{
  int i;
  double theta;
  DecisionTree *left, *right;
  DecisionTree(int _i, double _theta){
    i=_i;
    theta=_theta;
    left=NULL;
    right=NULL;
  }
};

DecisionTree* roots[trees];

double f(int cnt0, int cnt1, int pos0, int pos1)
{
  return cnt0*(1.0-pow(1.0*pos0/cnt0,2)-pow(1.0-1.0*pos0/cnt0,2))\
  +cnt1*(1.0-pow(1.0*pos1/cnt1,2)-pow(1.0-1.0*pos1/cnt1,2));
}

double sign(double x) {return x<=0?-1.0:1.0;}

bool myunic(Instance a, Instance b){
  return a.x[a.cmp]==b.x[b.cmp];
}

void build(DecisionTree* &u, vector<Instance> instances)
{
  //cout << "building...\n";
  // termination condition
  int equ=1;
  for(int i=0;i<instances.size()-1;i++) if(instances[i].y!=instances[i+1].y)
    {
    equ=0;
    break;
  }
  if(equ){
    if(instances[0].y>0) u=new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }
  else equ=1;
  for(int i=0;i<instances.size()-1;i++){
    for(int j=0;j<10;j++){
```

```
60        if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
61      }
62      if(!equ) break;
63    }
64    if(equ){
65      int cnt[2]={0};
66      for(auto X: instances){
67        if(X.y>0) cnt[1]++;
68        else cnt[0]++;
69      }
70      if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
71      else u=new DecisionTree(0,INF);
72      return;
73    }
74
75    // learn branching criteria
76    double mn=INF;
77    double theta_star;
78    int i_star;
79    for(int i=0;i<10;i++){
80      for(auto &X: instances) X.cmp = i;
81      sort(instances.begin(),instances.end());
82      vector<Instance> sorted_xi;
83      sorted_xi = instances;
84      auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
85        sorted_xi.resize( distance(sorted_xi.begin(),it) );
86
87        int cnt[2][1005];
88        //cnt[0]: number of positive instances count from the front(included)
89        //cnt[1]: number of positive instances count from the back(included)
90        cnt[0][0]=sorted_xi[0].y>0?1:0;
91        for(int j=1;j<sorted_xi.size();j++){
92          cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
93        }
94        cnt[1][sorted_xi.size()]=0;
95        for(int j=sorted_xi.size()-1;j>=0;j--){
96          cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
97        }
98
99        int N = sorted_xi.size();
100       double a;
101       if((a=f(0,N,0,cnt[1][0]))<mn){
102         mn=a;
103         theta_star=sorted_xi[0].x[i]-5.0; // ???
104         i_star = i;
105       }
106       for(int j=0;j<N;j++){
107         if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
108           mn=a;
109           theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; //
      ???
110           i_star = i;
111         }
112       }
113   }
114
115   u=new DecisionTree(i_star,theta_star);
116
117   vector<Instance> left, right;
118   for(auto &X: instances) X.cmp = i_star;
119   sort(instances.begin(),instances.end());
```

```
120   for(auto X: instances){
121     if(X.x[i_star] < theta_star) left.push_back(X);
122     else right.push_back(X);
123   }
124
125   build(u->left,left);
126   build(u->right,right);
127 }
128
129 void dfs(DecisionTree* u)
130 {
131   if(u==NULL) return;
132   cout << "i=" << u->i << ", theta=" << u->theta << "\n";
133   dfs(u->left);
134   dfs(u->right);
135 }
136
137 double predict(DecisionTree* u, Instance X)
138 {
139   if(X.x[u->i] < u->theta){
140     if(!u->left) return -1.0;
141     else return predict(u->left,X);
142   } else {
143     if(!u->right) return 1.0;
144     else return predict(u->right,X);
145   }
146 }
147
148 int main()
149 {
150   ios::sync_with_stdio(false); cin.tie(0);
151   fstream trainfile("hw6_train.dat"), testfile("hw6_test.dat");
152   vector<Instance> instances;
153   string s;
154   while(getline(trainfile,s)){
155     Instance X;
156     stringstream ss(s);
157     for(int i=0;i<10;i++){
158       ss >> X.x[i];
159     }
160     ss >> X.y;
161     instances.push_back(X);
162   }
163
164   vector<Instance> tests;
165   while(getline(testfile,s)){
166     Instance X;
167     stringstream ss(s);
168     for(int i=0;i<10;i++){
169       ss >> X.x[i];
170     }
171     ss >> X.y;
172     tests.push_back(X);
173   }
174
175   double E_out=0.0;
176   for(int t=0;t<trees;t++){
177     cout << "trees: " << t << endl;
178     vector<Instance> sample;
179     int N=instances.size();
180     uniform_int_distribution<int> distribution(0,N);
```

```
    for(int i=0;i<N/2;i++){
      int num = distribution(generator);
      sample.push_back(instances[num]);
    }
    build(roots[t], sample);
  }

   int err=0;
  for(auto X: tests){
    double sum=0.0;
    for(int i=0;i<trees;i++){
      double y_pred = predict(roots[i], X);
      sum+=y_pred;
    }
    if(sign(sum)!=X.y) err++;
  }

  /*int err=0;
  for(auto X: instances){
    double sum=0.0;
    for(int i=0;i<trees;i++){
      double y_pred = predict(roots[i], X);
      sum+=y_pred;
    }
    if(sign(sum)!=X.y) err++;
  }*/

  cout << err*1.0/instances.size() << endl;

}
```

16. Answer: (a). 0.016

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

const double INF = 1e18;
const int trees = 2000;
default_random_engine generator;

struct Instance{
  double x[10];
  double y;
  int cmp;
  bool operator < (const Instance & q) const{
    return x[cmp]<q.x[cmp];
  }
};

struct DecisionTree{
  int i;
  double theta;
  DecisionTree *left, *right;
  DecisionTree(int _i, double _theta){
    i=_i;
    theta=_theta;
    left=NULL;
    right=NULL;
  }
};

DecisionTree* roots[trees];

double f(int cnt0, int cnt1, int pos0, int pos1)
{
  return cnt0*(1.0-pow(1.0*pos0/cnt0,2)-pow(1.0-1.0*pos0/cnt0,2))\
  +cnt1*(1.0-pow(1.0*pos1/cnt1,2)-pow(1.0-1.0*pos1/cnt1,2));
}

double sign(double x) {return x<=0?-1.0:1.0;}

bool myunic(Instance a, Instance b){
  return a.x[a.cmp]==b.x[b.cmp];
}

void build(DecisionTree* &u, vector<Instance> instances)
{
  //cout << "building...\n";
  // termination condition
  int equ=1;
  for(int i=0;i<instances.size()-1;i++) if(instances[i].y!=instances[i+1].y)
    {
    equ=0;
    break;
  }
  if(equ){
    if(instances[0].y>0) u=new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }
  else equ=1;
  for(int i=0;i<instances.size()-1;i++){
    for(int j=0;j<10;j++){
```

```
60        if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
61      }
62      if(!equ) break;
63    }
64    if(equ){
65      int cnt[2]={0};
66      for(auto X: instances){
67        if(X.y>0) cnt[1]++;
68        else cnt[0]++;
69      }
70      if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
71      else u=new DecisionTree(0,INF);
72      return;
73    }
74
75    // learn branching criteria
76    double mn=INF;
77    double theta_star;
78    int i_star;
79    for(int i=0;i<10;i++){
80      for(auto &X: instances) X.cmp = i;
81      sort(instances.begin(),instances.end());
82      vector<Instance> sorted_xi;
83      sorted_xi = instances;
84      auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
85        sorted_xi.resize( distance(sorted_xi.begin(),it) );
86
87        int cnt[2][1005];
88        //cnt[0]: number of positive instances count from the front(included)
89        //cnt[1]: number of positive instances count from the back(included)
90        cnt[0][0]=sorted_xi[0].y>0?1:0;
91        for(int j=1;j<sorted_xi.size();j++){
92          cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
93        }
94        cnt[1][sorted_xi.size()]=0;
95        for(int j=sorted_xi.size()-1;j>=0;j--){
96          cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
97        }
98
99        int N = sorted_xi.size();
100       double a;
101       if((a=f(0,N,0,cnt[1][0]))<mn){
102         mn=a;
103         theta_star=sorted_xi[0].x[i]-5.0; // ???
104         i_star = i;
105       }
106       for(int j=0;j<N;j++){
107         if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
108           mn=a;
109           theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; //
      ???
110           i_star = i;
111         }
112       }
113   }
114
115   u=new DecisionTree(i_star,theta_star);
116
117   vector<Instance> left, right;
118   for(auto &X: instances) X.cmp = i_star;
119   sort(instances.begin(),instances.end());
```

```
120   for(auto X: instances){
121     if(X.x[i_star] < theta_star) left.push_back(X);
122     else right.push_back(X);
123   }
124
125   build(u->left,left);
126   build(u->right,right);
127 }
128
129 void dfs(DecisionTree* u)
130 {
131   if(u==NULL) return;
132   cout << "i=" << u->i << ", theta=" << u->theta << "\n";
133   dfs(u->left);
134   dfs(u->right);
135 }
136
137 double predict(DecisionTree* u, Instance X)
138 {
139   if(X.x[u->i] < u->theta){
140     if(!u->left) return -1.0;
141     else return predict(u->left,X);
142   } else {
143     if(!u->right) return 1.0;
144     else return predict(u->right,X);
145   }
146 }
147
148 int main()
149 {
150   ios::sync_with_stdio(false); cin.tie(0);
151   fstream trainfile("hw6_train.dat"), testfile("hw6_test.dat");
152   vector<Instance> instances;
153   string s;
154   while(getline(trainfile,s)){
155     Instance X;
156     stringstream ss(s);
157     for(int i=0;i<10;i++){
158       ss >> X.x[i];
159     }
160     ss >> X.y;
161     instances.push_back(X);
162   }
163
164   vector<Instance> tests;
165   while(getline(testfile,s)){
166     Instance X;
167     stringstream ss(s);
168     for(int i=0;i<10;i++){
169       ss >> X.x[i];
170     }
171     ss >> X.y;
172     tests.push_back(X);
173   }
174
175   double E_out=0.0;
176   for(int t=0;t<trees;t++){
177     cout << "trees: " << t << endl;
178     vector<Instance> sample;
179     int N=instances.size();
180     uniform_int_distribution<int> distribution(0,N);
```

```
    for(int i=0;i<N/2;i++){
      int num = distribution(generator);
      sample.push_back(instances[num]);
    }
    build(roots[t], sample);
  }

  /*int err=0;
  for(auto X: tests){
    double sum=0.0;
    for(int i=0;i<trees;i++){
      double y_pred = predict(roots[i], X);
      sum+=y_pred;
    }
    if(sign(sum)!=X.y) err++;
  }*/

  int err=0;
  for(auto X: instances){
    double sum=0.0;
    for(int i=0;i<trees;i++){
      double y_pred = predict(roots[i], X);
      sum+=y_pred;
    }
    if(sign(sum)!=X.y) err++;
  }

  cout << err*1.0/instances.size() << endl;

}
```

17. Answer: (d). 0.152

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

const double INF = 1e18;
const int trees = 2000;
default_random_engine generator;

struct Instance{
  double x[10];
  double y;
  int cmp;
  bool operator < (const Instance & q) const{
    return x[cmp]<q.x[cmp];
  }
};

struct DecisionTree{
  int i;
  double theta;
  DecisionTree *left, *right;
  DecisionTree(int _i, double _theta){
    i=_i;
    theta=_theta;
    left=NULL;
    right=NULL;
  }
};

DecisionTree* roots[trees];

double f(int cnt0, int cnt1, int pos0, int pos1)
{
  return cnt0*(1.0-pow(1.0*pos0/cnt0,2)-pow(1.0-1.0*pos0/cnt0,2))\
  +cnt1*(1.0-pow(1.0*pos1/cnt1,2)-pow(1.0-1.0*pos1/cnt1,2));
}

double sign(double x) {return x<=0?-1.0:1.0;}

bool myunic(Instance a, Instance b){
  return a.x[a.cmp]==b.x[b.cmp];
}

void build(DecisionTree* &u, vector<Instance> instances)
{
  //cout << "building...\n";
  // termination condition
  int equ=1;
  for(int i=0;i<instances.size()-1;i++) if(instances[i].y!=instances[i+1].y)
    {
    equ=0;
    break;
  }
  if(equ){
    if(instances[0].y>0) u=new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }
  else equ=1;
  for(int i=0;i<instances.size()-1;i++){
    for(int j=0;j<10;j++){
```

```
60        if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
61      }
62      if(!equ) break;
63    }
64    if(equ){
65      int cnt[2]={0};
66      for(auto X: instances){
67        if(X.y>0) cnt[1]++;
68        else cnt[0]++;
69      }
70      if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
71      else u=new DecisionTree(0,INF);
72      return;
73    }
74
75    // learn branching criteria
76    double mn=INF;
77    double theta_star;
78    int i_star;
79    for(int i=0;i<10;i++){
80      for(auto &X: instances) X.cmp = i;
81      sort(instances.begin(),instances.end());
82      vector<Instance> sorted_xi;
83      sorted_xi = instances;
84      auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
85        sorted_xi.resize( distance(sorted_xi.begin(),it) );
86
87        int cnt[2][1005];
88        //cnt[0]: number of positive instances count from the front(included)
89        //cnt[1]: number of positive instances count from the back(included)
90        cnt[0][0]=sorted_xi[0].y>0?1:0;
91        for(int j=1;j<sorted_xi.size();j++){
92          cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
93        }
94        cnt[1][sorted_xi.size()]=0;
95        for(int j=sorted_xi.size()-1;j>=0;j--){
96          cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
97        }
98
99        int N = sorted_xi.size();
100       double a;
101       if((a=f(0,N,0,cnt[1][0]))<mn){
102         mn=a;
103         theta_star=sorted_xi[0].x[i]-5.0; // ???
104         i_star = i;
105       }
106       for(int j=0;j<N;j++){
107         if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
108           mn=a;
109           theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; //
      ???
110           i_star = i;
111         }
112       }
113   }
114
115   u=new DecisionTree(i_star,theta_star);
116
117   vector<Instance> left, right;
118   for(auto &X: instances) X.cmp = i_star;
119   sort(instances.begin(),instances.end());
```

```
120   for(auto X: instances){
121     if(X.x[i_star] < theta_star) left.push_back(X);
122     else right.push_back(X);
123   }
124
125   build(u->left,left);
126   build(u->right,right);
127 }
128
129 void dfs(DecisionTree* u)
130 {
131   if(u==NULL) return;
132   cout << "i=" << u->i << ", theta=" << u->theta << "\n";
133   dfs(u->left);
134   dfs(u->right);
135 }
136
137 double predict(DecisionTree* u, Instance X)
138 {
139   if(X.x[u->i] < u->theta){
140     if(!u->left) return -1.0;
141     else return predict(u->left,X);
142   } else {
143     if(!u->right) return 1.0;
144     else return predict(u->right,X);
145   }
146 }
147
148 int main()
149 {
150   ios::sync_with_stdio(false); cin.tie(0);
151   fstream trainfile("hw6_train.dat"), testfile("hw6_test.dat");
152   vector<Instance> instances;
153   string s;
154   while(getline(trainfile,s)){
155     Instance X;
156     stringstream ss(s);
157     for(int i=0;i<10;i++){
158       ss >> X.x[i];
159     }
160     ss >> X.y;
161     instances.push_back(X);
162   }
163
164   vector<Instance> tests;
165   while(getline(testfile,s)){
166     Instance X;
167     stringstream ss(s);
168     for(int i=0;i<10;i++){
169       ss >> X.x[i];
170     }
171     ss >> X.y;
172     tests.push_back(X);
173   }
174
175   double E_out=0.0;
176   for(int t=0;t<trees;t++){
177     cout << "trees: " << t << endl;
178     vector<Instance> sample;
179     int N=instances.size();
180     uniform_int_distribution<int> distribution(0,N);
```

```
    for(int i=0;i<N/2;i++){
      int num = distribution(generator);
      sample.push_back(instances[num]);
    }
    build(roots[t], sample);
  }

  /*int err=0;
  for(auto X: tests){
    double sum=0.0;
    for(int i=0;i<trees;i++){
      double y_pred = predict(roots[i], X);
      sum+=y_pred;
    }
    if(sign(sum)!=X.y) err++;
  }*/

  int err=0;
  for(auto X: instances){
    double sum=0.0;
    for(int i=0;i<trees;i++){
      double y_pred = predict(roots[i], X);
      sum+=y_pred;
    }
    if(sign(sum)!=X.y) err++;
  }

  cout << err*1.0/instances.size() << endl;

}
```

18. Answer: (b). 0.075

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

const double INF = 1e18;
const int trees = 2000;
default_random_engine generator;

struct Instance{
  double x[10];
  double y;
  int cmp;
  bool operator < (const Instance & q) const{
    return x[cmp]<q.x[cmp];
  }
};

struct DecisionTree{
  int i;
  double theta;
  DecisionTree *left, *right;
  DecisionTree(int _i, double _theta){
    i=_i;
    theta=_theta;
    left=NULL;
    right=NULL;
  }
};

int A[1005][2005];
DecisionTree* roots[trees];

double f(int cnt0, int cnt1, int pos0, int pos1)
{
  return cnt0*(1.0-pow(1.0*pos0/cnt0,2)-pow(1.0-1.0*pos0/cnt0,2))\
  +cnt1*(1.0-pow(1.0*pos1/cnt1,2)-pow(1.0-1.0*pos1/cnt1,2));
}

double sign(double x) {return x<=0?-1.0:1.0;}

bool myunic(Instance a, Instance b){
  return a.x[a.cmp]==b.x[b.cmp];
}

void build(DecisionTree* &u, vector<Instance> instances)
{
  //cout << "building...\n";
  // termination condition
  int equ=1;
  for(int i=0;i<instances.size()-1;i++) if(instances[i].y!=instances[i+1].y)
    {
    equ=0;
    break;
  }
  if(equ){
    if(instances[0].y>0) u=new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }
  else equ=1;
  for(int i=0;i<instances.size()-1;i++){
```

```
60      for(int j=0;j<10;j++){
61        if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
62      }
63      if(!equ) break;
64    }
65    if(equ){
66      int cnt[2]={0};
67      for(auto X: instances){
68        if(X.y>0) cnt[1]++;
69        else cnt[0]++;
70      }
71      if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
72      else u=new DecisionTree(0,INF);
73      return;
74    }
75
76    // learn branching criteria
77    double mn=INF;
78    double theta_star;
79    int i_star;
80    for(int i=0;i<10;i++){
81      for(auto &X: instances) X.cmp = i;
82      sort(instances.begin(),instances.end());
83      vector<Instance> sorted_xi;
84      sorted_xi = instances;
85      auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
86        sorted_xi.resize( distance(sorted_xi.begin(),it) );
87
88        int cnt[2][1005];
89        //cnt[0]: number of positive instances count from the front(included)
90        //cnt[1]: number of positive instances count from the back(included)
91        cnt[0][0]=sorted_xi[0].y>0?1:0;
92        for(int j=1;j<sorted_xi.size();j++){
93          cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
94        }
95        cnt[1][sorted_xi.size()]=0;
96        for(int j=sorted_xi.size()-1;j>=0;j--){
97          cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
98        }
99
100       int N = sorted_xi.size();
101       double a;
102       if((a=f(0,N,0,cnt[1][0]))<mn){
103         mn=a;
104         theta_star=sorted_xi[0].x[i]-5.0; // ???
105         i_star = i;
106       }
107       for(int j=0;j<N;j++){
108         if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
109           mn=a;
110           theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; //
      ???
111           i_star = i;
112         }
113       }
114   }
115
116   u=new DecisionTree(i_star,theta_star);
117
118   vector<Instance> left, right;
119   for(auto &X: instances) X.cmp = i_star;
```

```
  sort(instances.begin(),instances.end());
  for(auto X: instances){
    if(X.x[i_star] < theta_star) left.push_back(X);
    else right.push_back(X);
  }

  build(u->left,left);
  build(u->right,right);
}

void dfs(DecisionTree* u)
{
  if(u==NULL) return;
  cout << "i=" << u->i << ", theta=" << u->theta << "\n";
  dfs(u->left);
  dfs(u->right);
}

double predict(DecisionTree* u, Instance X)
{
  if(X.x[u->i] < u->theta){
    if(!u->left) return -1.0;
    else return predict(u->left,X);
  } else {
    if(!u->right) return 1.0;
    else return predict(u->right,X);
  }
}

int main()
{
  ios::sync_with_stdio(false); cin.tie(0);
  fstream trainfile("hw6_train.dat"), testfile("hw6_test.dat");
  vector<Instance> instances;
  string s;
  while(getline(trainfile,s)){
    Instance X;
    stringstream ss(s);
    for(int i=0;i<10;i++){
      ss >> X.x[i];
    }
    ss >> X.y;
    instances.push_back(X);
  }

  vector<Instance> tests;
  while(getline(testfile,s)){
    Instance X;
    stringstream ss(s);
    for(int i=0;i<10;i++){
      ss >> X.x[i];
    }
    ss >> X.y;
    tests.push_back(X);
  }

  double E_out=0.0;
  for(int t=0;t<trees;t++){
    cout << "trees: " << t << endl;
    vector<Instance> sample;
    int N=instances.size();
```

```
181     uniform_int_distribution<int> distribution(0,N);
182     for(int i=0;i<N/2;i++){
183       int num = distribution(generator);
184       A[num][t]=1;
185       sample.push_back(instances[num]);
186     }
187     build(roots[t], sample);
188   }
189
190   int err=0;
191   for(int i=0;i<instances.size();i++){
192     double sum=0.0;
193     int check=0;
194     for(int j=0;j<trees;j++) if(!A[i][j]){
195       check=1;
196       double y_pred = predict(roots[j],instances[i]);
197       sum+=y_pred;
198     }
199     double tot_pred;
200     if(!check) tot_pred=-1.0;
201     else tot_pred=sign(sum);
202     if(tot_pred!=instances[i].y) err++;
203   }
204
205   cout << err*1.0/instances.size() << endl;
206
207 }
```

19. Answer: (d).

    I like the decision tree and random forest the most, because it utilizes the very basic concepts to achieve an excellent performance. When implementing on my own, I'm amazed that I can develop a training model on my own, and use it to predict some data. It uses a very simple idea to obtain good predictions, that why I like it the most.

20. Answer: (a).

    I think I like it the least because I don't understand it very much. For me, it's just an amazing package which can classify the data not badly. The limitation on the data size also limits the usage of the model (at least I don't use it in my final project). Maybe one day, I'll find it very interesting too if I understand it more.