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).

```cpp
#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,

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

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

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

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

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

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

```cpp
#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;
  }
```

```
if(equ){
  int cnt[2]={0};
  for(auto X: instances){
    if(X.y>0) cnt[1]++;
    else cnt[0]++;
  }
  if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
  else u=new DecisionTree(0,INF);
  return;
}

// learn branching criteria
double mn=INF;
double theta_star;
int i_star;
for(int i=0;i<10;i++){
  for(auto &X: instances) X.cmp = i;
  sort(instances.begin(),instances.end());
  vector<Instance> sorted_xi;
  sorted_xi = instances;
  auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
    sorted_xi.resize( distance(sorted_xi.begin(),it) );

    int cnt[2][1005];
    //cnt[0]: number of positive instances count from the front(included)
    //cnt[1]: number of positive instances count from the back(included)
    cnt[0][0]=sorted_xi[0].y>0?1:0;
    for(int j=1;j<sorted_xi.size();j++){
      cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
    }
    cnt[1][sorted_xi.size()]=0;
    for(int j=sorted_xi.size()-1;j>=0;j--){
      cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
    }

    int N = sorted_xi.size();
    double a;
    if((a=f(0,N,0,cnt[1][0]))<mn){
      mn=a;
      theta_star=sorted_xi[0].x[i]-5.0; // ???
      i_star = i;
    }
    for(int j=0;j<N;j++){
      if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
        mn=a;
        theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; // ???
        i_star = i;
      }
    }
}

u=new DecisionTree(i_star,theta_star);

vector<Instance> left, right;
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);
  }
  build(root, instances);

  int err=0, tot=0;
  while(getline(testfile,s)){
    tot++;
    Instance X;
    stringstream ss(s);
    for(int i=0;i<10;i++){
      ss >> X.x[i];
    }
    ss >> X.y;
    double y_pred = predict(root, X);
    if(y_pred!=X.y) err++;
  }
  cout << 1.0*err/tot << "\n";
}
```

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++){
      if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
    }
    if(!equ) break;
  }
  if(equ){
    int cnt[2]={0};
    for(auto X: instances){
      if(X.y>0) cnt[1]++;
      else cnt[0]++;
    }
    if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }

  // learn branching criteria
  double mn=INF;
  double theta_star;
  int i_star;
  for(int i=0;i<10;i++){
    for(auto &X: instances) X.cmp = i;
    sort(instances.begin(),instances.end());
    vector<Instance> sorted_xi;
    sorted_xi = instances;
    auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
      sorted_xi.resize( distance(sorted_xi.begin(),it) );

      int cnt[2][1005];
      //cnt[0]: number of positive instances count from the front(included)
      //cnt[1]: number of positive instances count from the back(included)
      cnt[0][0]=sorted_xi[0].y>0?1:0;
      for(int j=1;j<sorted_xi.size();j++){
        cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
      }
      cnt[1][sorted_xi.size()]=0;
      for(int j=sorted_xi.size()-1;j>=0;j--){
        cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
      }

      int N = sorted_xi.size();
      double a;
      if((a=f(0,N,0,cnt[1][0]))<mn){
        mn=a;
        theta_star=sorted_xi[0].x[i]-5.0; // ???
        i_star = i;
      }
      for(int j=0;j<N;j++){
        if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
          mn=a;
          theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; // ???
          i_star = i;
        }
      }
  }

  u=new DecisionTree(i_star,theta_star);

  vector<Instance> left, right;
  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();
    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

```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++){
      if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
    }
    if(!equ) break;
  }
  if(equ){
    int cnt[2]={0};
    for(auto X: instances){
      if(X.y>0) cnt[1]++;
      else cnt[0]++;
    }
    if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }

  // learn branching criteria
  double mn=INF;
  double theta_star;
  int i_star;
  for(int i=0;i<10;i++){
    for(auto &X: instances) X.cmp = i;
    sort(instances.begin(),instances.end());
    vector<Instance> sorted_xi;
    sorted_xi = instances;
    auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
      sorted_xi.resize( distance(sorted_xi.begin(),it) );

      int cnt[2][1005];
      //cnt[0]: number of positive instances count from the front(included)
      //cnt[1]: number of positive instances count from the back(included)
      cnt[0][0]=sorted_xi[0].y>0?1:0;
      for(int j=1;j<sorted_xi.size();j++){
        cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
      }
      cnt[1][sorted_xi.size()]=0;
      for(int j=sorted_xi.size()-1;j>=0;j--){
        cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
      }

      int N = sorted_xi.size();
      double a;
      if((a=f(0,N,0,cnt[1][0]))<mn){
        mn=a;
        theta_star=sorted_xi[0].x[i]-5.0; // ???
        i_star = i;
      }
      for(int j=0;j<N;j++){
        if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
          mn=a;
          theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; // ???
          i_star = i;
        }
      }
  }

  u=new DecisionTree(i_star,theta_star);

  vector<Instance> left, right;
  for(auto &X: instances) X.cmp = i_star;
  sort(instances.begin(),instances.end());
```

```cpp
  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();
    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++){
      if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
    }
    if(!equ) break;
  }
  if(equ){
    int cnt[2]={0};
    for(auto X: instances){
      if(X.y>0) cnt[1]++;
      else cnt[0]++;
    }
    if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }

  // learn branching criteria
  double mn=INF;
  double theta_star;
  int i_star;
  for(int i=0;i<10;i++){
    for(auto &X: instances) X.cmp = i;
    sort(instances.begin(),instances.end());
    vector<Instance> sorted_xi;
    sorted_xi = instances;
    auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
      sorted_xi.resize( distance(sorted_xi.begin(),it) );

      int cnt[2][1005];
      //cnt[0]: number of positive instances count from the front(included)
      //cnt[1]: number of positive instances count from the back(included)
      cnt[0][0]=sorted_xi[0].y>0?1:0;
      for(int j=1;j<sorted_xi.size();j++){
        cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
      }
      cnt[1][sorted_xi.size()]=0;
      for(int j=sorted_xi.size()-1;j>=0;j--){
        cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
      }

      int N = sorted_xi.size();
      double a;
      if((a=f(0,N,0,cnt[1][0]))<mn){
        mn=a;
        theta_star=sorted_xi[0].x[i]-5.0; // ???
        i_star = i;
      }
      for(int j=0;j<N;j++){
        if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
          mn=a;
          theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; // ???
          i_star = i;
        }
      }
  }

  u=new DecisionTree(i_star,theta_star);

  vector<Instance> left, right;
  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();
    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

```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;
  }
};

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++){
    for(int j=0;j<10;j++){
      if(instances[i].x[j]!=instances[i+1].x[j]) {equ=0; break;}
    }
    if(!equ) break;
  }
  if(equ){
    int cnt[2]={0};
    for(auto X: instances){
      if(X.y>0) cnt[1]++;
      else cnt[0]++;
    }
    if(cnt[1]>cnt[0]) u= new DecisionTree(0,-INF);
    else u=new DecisionTree(0,INF);
    return;
  }

  // learn branching criteria
  double mn=INF;
  double theta_star;
  int i_star;
  for(int i=0;i<10;i++){
    for(auto &X: instances) X.cmp = i;
    sort(instances.begin(),instances.end());
    vector<Instance> sorted_xi;
    sorted_xi = instances;
    auto it = unique (sorted_xi.begin(), sorted_xi.end(), myunic);
      sorted_xi.resize( distance(sorted_xi.begin(),it) );

      int cnt[2][1005];
      //cnt[0]: number of positive instances count from the front(included)
      //cnt[1]: number of positive instances count from the back(included)
      cnt[0][0]=sorted_xi[0].y>0?1:0;
      for(int j=1;j<sorted_xi.size();j++){
        cnt[0][j]=sorted_xi[j].y>0?cnt[0][j-1]+1:cnt[0][j-1];
      }
      cnt[1][sorted_xi.size()]=0;
      for(int j=sorted_xi.size()-1;j>=0;j--){
        cnt[1][j]=sorted_xi[j].y>0?cnt[1][j+1]+1:cnt[1][j+1];
      }

      int N = sorted_xi.size();
      double a;
      if((a=f(0,N,0,cnt[1][0]))<mn){
        mn=a;
        theta_star=sorted_xi[0].x[i]-5.0; // ???
        i_star = i;
      }
      for(int j=0;j<N;j++){
        if((a=f(j+1,N-j-1,cnt[0][j],cnt[1][j+1]))<mn){
          mn=a;
          theta_star=(double)(sorted_xi[j].x[i]+sorted_xi[j+1].x[i])/2.0; // ???
          i_star = i;
        }
      }
  }

  u=new DecisionTree(i_star,theta_star);

  vector<Instance> left, right;
  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();
    uniform_int_distribution<int> distribution(0,N);
    for(int i=0;i<N/2;i++){
      int num = distribution(generator);
      A[num][t]=1;
      sample.push_back(instances[num]);
    }
    build(roots[t], sample);
  }

  int err=0;
  for(int i=0;i<instances.size();i++){
    double sum=0.0;
    int check=0;
    for(int j=0;j<trees;j++) if(!A[i][j]){
      check=1;
      double y_pred = predict(roots[j],instances[i]);
      sum+=y_pred;
    }
    double tot_pred;
    if(!check) tot_pred=-1.0;
    else tot_pred=sign(sum);
    if(tot_pred!=instances[i].y) err++;
  }

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

}
```

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.