Solution 1.

(a) $E(Y) = f(p_\alpha, p_\beta)$

we know $p_\alpha + p_\beta + p_\gamma = 1$

$\Rightarrow p_\gamma = 1 - p_\alpha - p_\beta$

$$E(Y) = -\sum_{x \in \{\alpha,\beta,\gamma\}} P(X=x) \log_2 P(x=x)$$

$$= -p_\alpha \log_2 p_\alpha - p_\beta \log_2 p_\beta - p_\gamma \log_2 p_\gamma$$

$$= -p_\alpha \log_2 p_\alpha - p_\beta \log_2 p_\beta - (1-p_\alpha-p_\beta) \log_2 (1-p_\alpha-p_\beta)$$

$$= -p_\alpha \log_2 p_\alpha + p_\alpha \log_2 (1-p_\alpha-p_\beta - p_\beta \log_2 p_\beta + p_\beta \log (1-p_\alpha-p_\beta) - \log_2 (1-p_\alpha-p_\beta)$$

$$= -[p_\alpha [\log_2 p_\alpha - \log_2 (1-p_\alpha-p_\beta)] + p_\beta (\log p_\beta - \log_2 (1-p_\alpha-p_\beta) + \log_2 (1-p_\alpha-p_\beta)$$

$$= -\left[p_\alpha \cdot \log_2 \frac{p_\alpha}{1-p_\alpha-p_\beta} + p_\beta \log_2 \frac{p_\beta}{1-p_\alpha-p_\beta} + \log_2 (1-p_\alpha-p_\beta) \right]$$

(2) For 2 value case, $p_\alpha + p_\beta = 1$

$$E(Y) = -[p_\alpha \log_2 p_\alpha + p_\beta \log_2 p_\beta]$$

$$= -[p_\alpha \log_2 p_\alpha + (1-p_\alpha) \log_2 (1-p_\alpha)]$$

$$\frac{d E(Y)}{d p_\alpha} = -[\log_2 p_\alpha + 1 - \log_2 (1-p_\alpha) - 1]$$

$$= -\left[\log_2 \frac{p_\alpha}{1-p_\alpha} \right]$$

The value is maximal when

$$\frac{d\,E(Y)}{d\,p_\alpha} = 0$$

$$\Rightarrow \quad \log \frac{p_\alpha}{1-p_\alpha} = 0$$

$$\Rightarrow \quad \frac{p_\alpha}{1-p_\alpha} = 1$$

$$\Rightarrow \quad p_\alpha = 1 - p_\alpha$$

$$\Rightarrow \quad p_\alpha = 1/2 \Rightarrow p_\beta = 1/2.$$

$E(Y)$ is maximal when $p_\alpha = p_\beta = 1/2$.

(3) We do not need a formal proof as we know that the entropy of the system (uncertainty of the system) is highest when all values have equal probability of occuring.

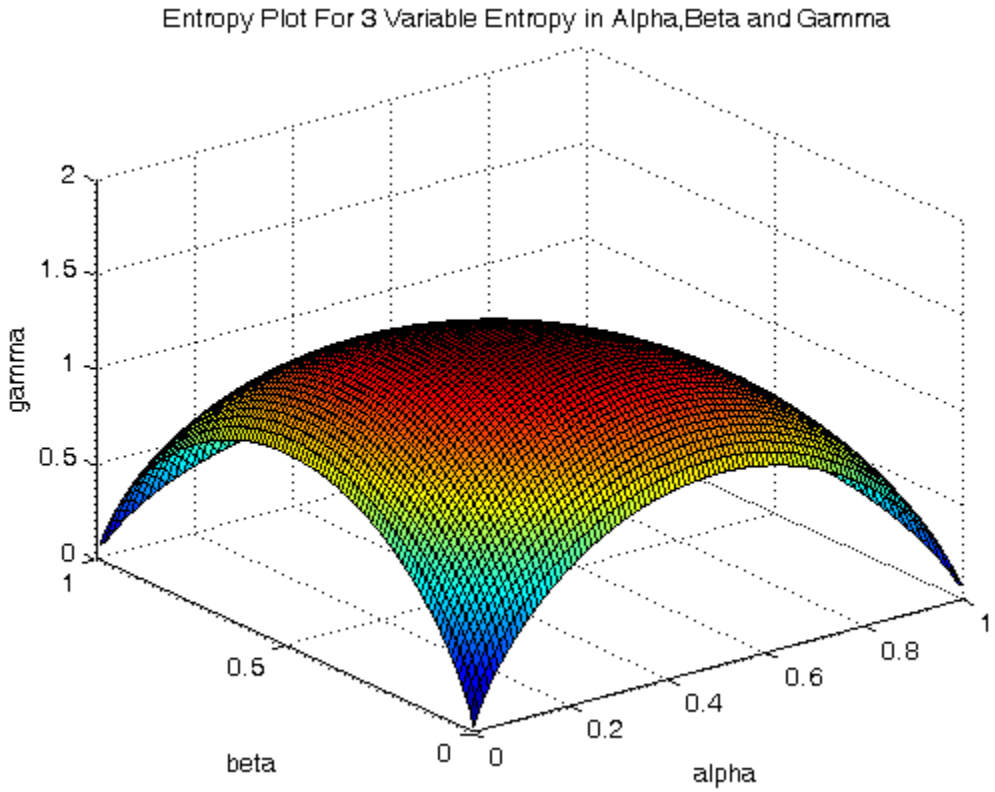
Problem 2

```
function calculateEyAndPlot()
   [x,y]=meshgrid(0.001:0.01:1);
   size=length(x);
   for(i=1:size)
      for(j=1:size)
         if(x(i,j)+y(i,j)>1)
            y(i,j)=1-x(i,j);
         end
         if(x(i,j)+y(i,j)<=1)
           z(i,j)=1-x(i,j)-y(i,j);
         end
      end
   end
   for(i=1:size)
      for(j=1:size)
         alpha=x(i,j);
         beta=y(i,j);
         gamma=z(i,j);
         if(~alpha) alpha=1; end
         if (~beta) beta=1; end
         if(~gamma) gamma=1; end

        c(i,j)=-(alpha*log2(alpha)+beta*log2(beta)+gamma*log2(gamma));

      end
   end
   surf(x,y,real(c));
   title('Entropy Plot For 3 Variable Entropy in Alpha,Beta and Gamma')
   xlabel('alpha')
   ylabel('beta')
   zlabel('gamma')

end
calculateEyAndPlot()
```


Entropy Plot For 3 Variable Entropy in Alpha,Beta and Gamma
gamma
2
1.5
1
0.5
0
beta
1
0.5
0
alpha
0
0.2
0.4
0.6
0.8
1

Problem 3

Solution 3

(a) For two variables, $p_\alpha + p_\beta = 1$,
$E(y)$ is max at $p_\alpha = p_\beta = 1/2$

For 3 variables, $p_\alpha + p_\beta + p_\gamma = 1$
$E(y)$ is max at $p_\alpha = p_\beta = p_\gamma = .33$.

So for a 4 variable system in $\{a, b, c, d\}$

$E(y)$ is maximum at $p_a = p_b = p_c = p_d = 1/4$

(b) $$E(y) = -\sum_{x \in \{a,b,c,d\}} P(x=x) \log_2 P(x=x)$$

$$= -\left(4 \times \frac{1}{4} \log_2 \frac{1}{4}\right)$$

$$= 2.$$

Problem 4

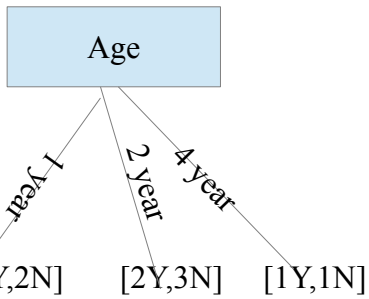
From the data set we calculate the overall entropy of the data set

Overall Entropy

Number of positive data samples = 6 .
Number of negative data samples = 6
Total Samples = 12

Entropy = -6/12*log6/12 – 6/12log6/12
= -1/2log1/2 – 1/2log1/2
=1/2log2 +1/2log2
=log 2
= 1

<u>Gain From Age as the attribute</u>



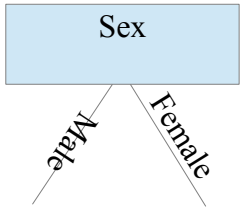
E[Age = 1] = - [3/5log3/5 +2/5log2/5]
= 3/5log5/3 +2/5log5/2
= 0.9710

E[Age=2]= -2/5log2/5 – 3/5log3/5
= 2/5log5/2+ 3/5log5/3
= 0.9710

E[Age=3] = -1/2log1/2 -1/2log1/2
= log 2 = 1

Gain(Data,Age) = 1 - (5/12*(.9710) + 5/12*(9710) + 2/12*(1))
=0.0242

<u>Gain From Selecting Attribute as Sex</u>



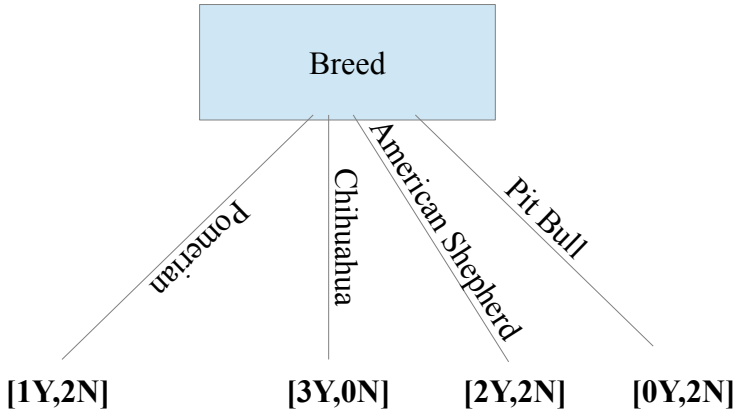
[3Y,3N] [3Y,3N]

E(Sex = Male) = -3/6log3/6 -3/6log3/6 = log2 = 1

E(Sex = Female) = -3/6log3/6 -3/6log3/6 = log2 = 1

Gain(Data,Sex) = 1 – (6/12*1 + 6/12*1) = 0

Gain From Selecting Breed as attribute



Entropy(Breed = Pmerian) = -1/3log1/3 – 2/3log2/3
= 1/3log3 +2/3log3/2
=0.9183

Entropy(Breed = Chihuahua) = -3/3log3/3 – 0log0
= 1log1 = 0

Entropy(Breed = American Shepherd = -1/2log2/4 -1/2log2/4
= -1/2log1/2 -12/log1/2
=1

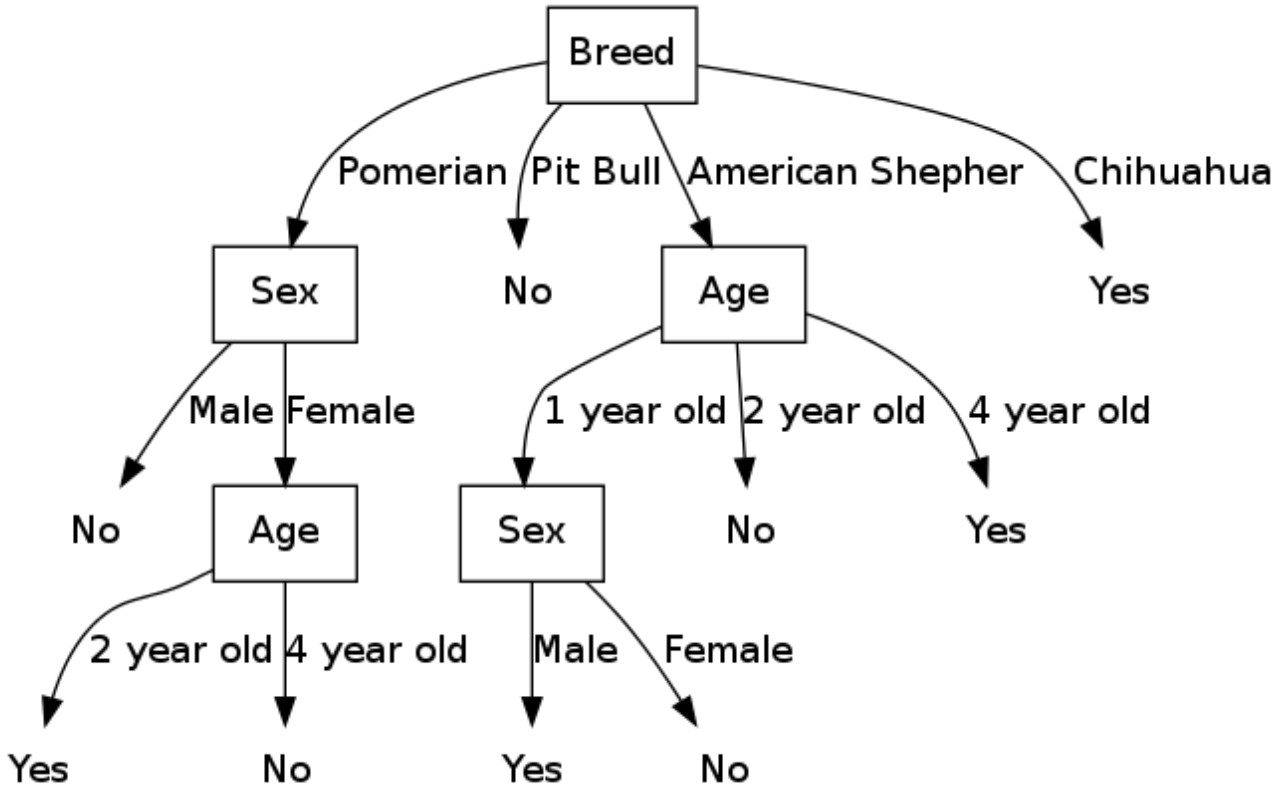
Entropy(Breed=Pitbull) = 0 – 2/2log2/2 = 0

Gain(Data,Breed) = 1 – (3/12 *0.9183 + 3/12* 0 + 4/12 * 1 + 2/12 * 0)
= 1 – (3/12 * 0.9183 + 1/3) = 0.4371

So we select the attribute Breed as the overall gain from using this highest among all the attributes.

Solution 5

The ID3 tree derived from the data set is as below.



ID3.H

```
#ifndef __ID3_LEARNING_H__
#define __ID3_LEARNING_H__


#include<iostream>
#include<stdio.h>
#include<math.h>
#include<string>
#include<list>
#include<vector>
#include<stdlib.h>
#include<fstream>
#include<sstream>
using namespace std;
enum Sex
{
 M,
 F
};

enum Breed
{
```

```
 P,
 PB ,
 AS ,
 C
};
enum Decision
{
 N,
 Y
};

enum Attribute
{
 Age,
 Sex,
 Breed
};

enum NodeType
{
 NotLeaf,
 Leaf
};

class ID3
{
 public:
 int at_;
 int numOfChildNodes_;
 NodeType nt_;
 vector<int> attrList_;
 string nodeLabel_;
 ID3 * parent_;
 ID3 *  childNodes_[4];
 string childLabels_[4];
 ID3()
 {
  numOfChildNodes_=0;
  for(int i=0;i<4;i++)
  {
    childNodes_[i]=NULL;
    childLabels_[i]="";
  }
 }
};

double calculateOverAllEntropy(int data[][4],int size);
double calculateEntropy(int numY,int numN);
double calculateGain(Attribute at,double ent,int data[][4],int size,int &leafType);
void findDistribution(Attribute at,int val,int & numY,int &numN,int data[][4],int size);

void calculatePerAttributeGain(Attribute at,int Values[],double ent,int data[][4],int datasize,int atSize,double & gain,int & leafType);

ID3 * createID3TreeNode(int at,NodeType ndt,string nodelabel,string parentLabel,ID3 * parent);
void createID3R(ID3 * parent,int at, int data[][4],int size,vector<Attribute> attributeList,string parentLabel);

void printNode(ID3 * node,bool nodeCreated,string nodeStr);
void attrToName(int at,string & atName);
void calculateLeafDecision(int data[][4],int datasize,int &leafType);
void mapParentAttrChildAttr(int atV, int childAtv,string & name);
void flushToFile(const string fileName);

#endif
```

ID3.cpp

```
#include "ID3.H"
int datag[][4]= {
{1, M, C , Y },
{1 ,F, C, Y } ,
```

```
{2 ,M, C, Y } ,
{4, F, AS , Y },
{1, M, AS , Y },
{1, F, AS, N },
{2 ,F, AS, N},
{2, M, PB, N },
{1, M, PB , N },
{4 ,F, P, N } ,
{2, M, P, N },
{2 ,F, P, Y }
};

static int attributeCount1 = 0, leafCount=1;
static string globalGraphVizString = "digraph G { \n";
ID3 * root=NULL;
int main(int argc,char * argv[])
{
 vector<Attribute> attributeList;

 attributeList.push_back(Age);
 attributeList.push_back(Sex);
 attributeList.push_back(Breed);
 double overallEnt=calculateOverAllEntropy(datag,12);
 vector<Attribute>::iterator it=attributeList.begin();
 Attribute attributeWithMaxGain;
 double maxGain=0;
 int leafType;
 while(it!=attributeList.end())
 {
  double cGain = calculateGain(*(it),overallEnt,datag,12,leafType);
  if(cGain >= maxGain)
  {
    attributeWithMaxGain = *(it);
    maxGain=cGain;
  }
  it++;
 }
 //cout<<" XXXX Attribute Gain " << maxGain<<"Attribute " <<attributeWithMaxGain<<endl;
 it=attributeList.begin();
 while(it!=attributeList.end())
 {
  if(*(it) == attributeWithMaxGain)
  {
    it=attributeList.erase(it);
  }
  else
    it++;
 }
 int atV=attributeWithMaxGain;
 string atName;
 attrToName(atV,atName);
 createID3TreeNode(atV,NotLeaf,atName,"",NULL);
 //The internal labels like P,PB,AS,C
 vector<int>::iterator itAttr = root->attrList_.begin();
 while(itAttr!=root->attrList_.end())
 {
  int dataTmp[12][4]={0};
  int size=0;
  for (int i =0;i<12;i++)
  {
     if(datag[i][atV]==*(itAttr))
     {
      dataTmp[size][0]=datag[i][0];
      dataTmp[size][1]=datag[i][1];
      dataTmp[size][2]=datag[i][2];
      dataTmp[size][3]=datag[i][3];
      cout<<"Match at posn "<<i+1<<"for attr "<<*(itAttr)<<endl;
      size++;
     }
  }
```

```
    //string atName;
    attrToName(atV,atName);
    cout<<"Creating tree for sub-attribute of type" << *(itAttr) <<" for attribute "<<atName<<endl ;

    mapParentAttrChildAttr(atV,*(itAttr),atName);
    createID3R(root,*(itAttr),dataTmp,size,attributeList,atName);
    itAttr++;

  }

  printNode(root,false,"");
  globalGraphVizString+="\n}\n";
  string fileName="dt.dot";
  flushToFile(fileName);
  string cmdstr="dot -Tpng -o dt.png "+ fileName;
  system(cmdstr.c_str());
  cout<<endl<<globalGraphVizString<<endl;


  return 0;
}

double calculateOverAllEntropy(int data[][4],int size)
{
  int yVal =0;
  int nVal = 0;
  for(int i =0;i<size;i++)
  {
     if(data[i][3] == Y)
        yVal++;
     else if (data[i][3] == N)
        nVal++;
  }
  double x = size;
  double entropy = (-(yVal/x) *log2 (yVal/x) )
                  + (-(nVal/x) *log2 (nVal/x));
  //cout<<"Num of Y " << yVal <<" number of N "<< nVal<< " Entropy "<<entropy<<endl;

  return entropy;

}

double calculateGain(Attribute at,double ent,int data[][4],int size,int &leafType)
{

   double gain=ent;
   bool allVals=Y;
   switch(at)
   {
     case Age:
          {
            int ageValues[3]={1,2,4};
            calculatePerAttributeGain(at,ageValues,ent,data,size,3,gain,leafType);
          }

          break;
     case Sex:
          {
            int sexValues[2]={M,F};
            calculatePerAttributeGain(at,sexValues,ent,data,size,2,gain,leafType);
          }
          break;
     case Breed:
          {
            int breedValues[4]={P,PB,AS,C};
            calculatePerAttributeGain(at,breedValues,ent,data,size,4,gain,leafType);
          }
          break;
     default:
          break;
```

```
  }
  //cout<<"Attribute Gain " << gain<<endl;
  return gain;
}
void calculatePerAttributeGain(Attribute at,int Values[],double ent,int data[][4],int datasize,int atSize,double & gain,int & leafType)
{
  int YNDistribution[4][2]={{0,0},
                            {0,0},
                            {0,0},
                            {0,0}};
  double entropies [4]={0,0,0,0};
  int sumY=0,sumN=0;
  for(int i=0;i<atSize;i++)
  {
    int numY=0;
    int numN=0;
    findDistribution(at,Values[i],numY,numN,data,datasize);
    sumY+=numY;
    sumN+=numN;
    YNDistribution[i][0]=numY;
    YNDistribution[i][1]=numN;
  }
  int x = datasize;
  double sumEnt=0;
  for (int i=0;i<atSize;i++)
  {
     entropies[i]=calculateEntropy(YNDistribution[i][0],YNDistribution[i][1]);
     sumEnt +=((YNDistribution[i][0]+YNDistribution[i][1])/(double)x) * entropies[i];
     if(entropies[i] ==-0) entropies[i]=0;
     cout<<"ent" << entropies[i] << "sumEnt"<<sumEnt<<endl;
  }
  gain = gain-sumEnt;
  cout<<"Overall entropy "<<sumEnt<<"Gain is "<<gain<<endl;
  if(sumY==datasize)
  {
    cout<<"Leaf type is YES"<<endl;
    leafType=Y;
  }
  else if(sumN==datasize)
  {
    cout<<"Leaf type is NO"<<endl;
    leafType=N;
  }
  else
  {
    cout<<" NOT Leaf type"<<endl;
    leafType=-1;
  }

}

double calculateEntropy(int yVal,int nVal)
{
 double x = yVal+nVal;
 double entropy =0;
 if((nVal==0) && (yVal==0))
    entropy = 0;
 else if(yVal==0)
    entropy = (-(nVal/x) *log2 (nVal/x));
 else if (nVal ==0)
    entropy = (-(yVal/x) *log2 (yVal/x));
 else
    entropy = (-(yVal/x) *log2 (yVal/x) )
              + (-(nVal/x) *log2 (nVal/x));
 return entropy;
}
void findDistribution(Attribute at,int val,int & numY,int &numN,int data[][4],int size)
{
 int attributeToCheck=at;
 for(int i =0;i<size;i++)
```

```
 {
    if((data[i][attributeToCheck] == val ) &&
      (data[i][3] == Y ) )
      numY++;
    else if((data[i][attributeToCheck] == val ) &&
          (data[i][3] == N ) )
      numN++;
 }

}
ID3 * createID3TreeNode(int at,NodeType ndt,string label,string parentLabel,ID3 * parent)
{

 if(root==NULL)
 {
  root= new ID3[1];
  root->at_ =at;
  root->nt_ =ndt;
  root->nodeLabel_=label;
  root->parent_=parent;
  switch(at)
  {
   case Age:
       break;
   case Sex:
       break;
   case Breed:
       root->attrList_.push_back(P);
       root->attrList_.push_back(PB);
       root->attrList_.push_back(AS);
       root->attrList_.push_back(C);
       break;
   default:
       break;

  }
  return root;
 }
 else
 {
  ID3 * node = new ID3[1];
  node->at_ =at;
  node->nt_ =ndt;
  node->nodeLabel_=label;
  node->parent_=parent;
  node->parent_->childNodes_[node->parent_->numOfChildNodes_]=node;
  node->parent_->childLabels_[node->parent_->numOfChildNodes_]=parentLabel;
  node->parent_->numOfChildNodes_++;
  switch(at)
  {
   case Age:
       node->attrList_.push_back(1);
       node->attrList_.push_back(2);
       node->attrList_.push_back(4);
       break;
   case Sex:
       node->attrList_.push_back(M);
       node->attrList_.push_back(F);
       break;
   case Breed:
       node->attrList_.push_back(P);
       node->attrList_.push_back(PB);
       node->attrList_.push_back(AS);
       node->attrList_.push_back(C);

       break;
   default:
       break;
  }
  return node;
```

```
  }
}
void createID3R(ID3 * parent,int at, int data[][4],int datasize,vector<Attribute> attributeList,string parentLabel)
{
 cout<<"Data size is "<<datasize<<endl;
 if(datasize ==0)
 {
  cout<<"0 size "<<endl;
  return;
 }
 double overallEnt=calculateOverAllEntropy(data,datasize);
 Attribute attributeWithMaxGain;
 double maxGain=0;
 int leafDecisionType=0;
 ID3 * node=NULL;

 vector<Attribute>::iterator it=attributeList.begin();
 if(!attributeList.empty())
 {
 while(it!=attributeList.end())
 {
  double cGain = calculateGain(*(it),overallEnt,data,datasize,leafDecisionType);
  if((leafDecisionType==Y) || (leafDecisionType==N)) break;
  if(cGain >= maxGain)
  {
    attributeWithMaxGain = *(it);
    maxGain=cGain;
  }
  it++;
 }
 }
 else
 {
  calculateLeafDecision(data,datasize,leafDecisionType);
 }
 string atName;
 attrToName(attributeWithMaxGain,atName);
 cout<<"Max gain attribute is "<<atName<<endl;
 //No reduction in entropy so this has to be the leaf node.
 if(((overallEnt-maxGain) ==1) || (leafDecisionType!=-1))
 {
  //Create the leaf node if any elements
  string leafLabel="";
  if(leafDecisionType==Y) leafLabel ="Yes";
  if(leafDecisionType==N) leafLabel ="No";
  cout<<"Create a leaf node of type "<<leafLabel<<endl;
  node=createID3TreeNode(attributeWithMaxGain,Leaf,leafLabel,parentLabel,parent);
  return;
 }
 else
 {
  cout<<"Creating a node for type"<<atName<<endl;
  node=createID3TreeNode(attributeWithMaxGain,NotLeaf,atName,parentLabel,parent);
 }
 //Remove the attribute from subsequent iteration
 it=attributeList.begin();
 while(it!=attributeList.end())
 {
  if(*(it) == attributeWithMaxGain)
  {
    it=attributeList.erase(it);
  }
  else
    it++;
 }
 int atV = attributeWithMaxGain;
 vector<int>::iterator itAttr = node->attrList_.begin();
 while(itAttr!=node->attrList_.end())
 {
  int dataTmp[12][4]={0};
```

```
   int size=0;
   for (int i =0;i<datasize;i++)
   {
      if(data[i][atV]==*(itAttr))
      {
       dataTmp[size][0]=data[i][0];
       dataTmp[size][1]=data[i][1];
       dataTmp[size][2]=data[i][2];
       dataTmp[size][3]=data[i][3];
       size++;
      }
  }

  string atName;
  attrToName(atV,atName);
  cout<<"Creating tree for sub-attribute of type" << *(itAttr) <<" for attribute "<<atName<<endl ;

  mapParentAttrChildAttr(atV,*(itAttr),atName);
  createID3R(node,*(itAttr),dataTmp,size,attributeList,atName);
  itAttr++;
 }
}

void printNode(ID3 * node,bool nodeCreated,string nodeStr)
{
 cout<<"Node Details"<<endl;
 string nodeType;
 cout<<"Attribute node::"<<node->nodeLabel_<<endl;
 string parentNode=nodeStr;
 if((node->nt_==NotLeaf) && !nodeCreated)
 {
  ostringstream convert;
  convert << attributeCount1;
  globalGraphVizString += "attr"+convert.str();
  globalGraphVizString+=" [shape=\"rectangle\", label="+node->nodeLabel_+"]\n";
  parentNode +="attr"+convert.str();
  attributeCount1++;
 }
 else if((node->nt_==Leaf) && !nodeCreated)
 {
   ostringstream convert;
   convert << leafCount;
   globalGraphVizString += "leaf"+convert.str();
   globalGraphVizString+=" [shape=\"plaintext\", label="+node->nodeLabel_+"]\n";
   parentNode +="leaf"+convert.str();
   leafCount++;
 }

 for(int i=0;i<node->numOfChildNodes_;i++)
 {
   cout<<"Child label "<<node->childLabels_[i]<<endl;
   string nodeNext="";
   if(node->childNodes_[i]->nt_==NotLeaf)
   {
     ostringstream convert;
     convert << attributeCount1;
     globalGraphVizString += "attr"+convert.str();
     globalGraphVizString+=" [shape=\"rectangle\", label="+node->childNodes_[i]->nodeLabel_+"]\n";
     nodeNext +="attr"+convert.str();
     attributeCount1++;
   }
   else if (node->childNodes_[i]->nt_==Leaf)
   {
      ostringstream convert;
      convert << leafCount;
      globalGraphVizString += "leaf"+convert.str();
      globalGraphVizString +=" [shape=\"plaintext\", label="+node->childNodes_[i]->nodeLabel_+"]\n";
      nodeNext +="leaf"+convert.str();
      leafCount++;
   }
```

```
    globalGraphVizString += parentNode + " -> "+ nodeNext+" [label =\""+node->childLabels_[i]+"\"]\n";
    printNode(node->childNodes_[i],true,nodeNext);
  }


}
void attrToName(int at,string & atName)
{
  if(at==Age) atName="Age";
  if(at==Sex) atName="Sex";
  if(at==Breed) atName="Breed";

}
void calculateLeafDecision(int data[][4],int datasize,int &leafType)
{
  int numY=0,numN=0;
  for(int i =0;i<datasize;i++)
  {
        if(data[i][3] == Y)
          numY++;
        else if(data[i][3] == N )
        numN++;
  }
  if(numY==datasize)
  {
     cout<<"Leaf type is YES"<<endl;
     leafType=Y;
  }
  else if(numN==datasize)
  {
    cout<<"Leaf type is NO"<<endl;
    leafType=N;
  }
  else
  {
    cout<<" NOT Leaf type"<<endl;
    leafType=-1;
  }


}
void mapParentAttrChildAttr(int atV, int childAtv,string & name)
{
   switch(atV)
   {
     case Age:
          if(childAtv==1) name = "1 year old";
          if(childAtv==2) name = "2 year old";
          if(childAtv==4) name = "4 year old";
          break;
     case Sex:
          if(childAtv==M) name = "Male";
          if(childAtv==F) name = "Female";
          break;
     case Breed:
          if(childAtv==P) name = "Pomerian";
          if(childAtv==PB) name = "Pit Bull";
          if(childAtv==AS) name = "American Shepher";
          if(childAtv==C) name = "Chihuahua";
          break;
   }
   cout<<"Mapped name "<<name<<endl;
}
void flushToFile(const string fileName)
{
  ofstream myfile;
  myfile.open (fileName.c_str());
  myfile << globalGraphVizString;
  myfile.close();
```

```
}
```