Machine Learning and ID tree

What is machine learning (ML)?

Tom Mitchell (prof. in Carnegie Mellon University) defined

Definition:
A computer program is said to learn from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks *T*, as measured by *P*, improves with experience *E*.

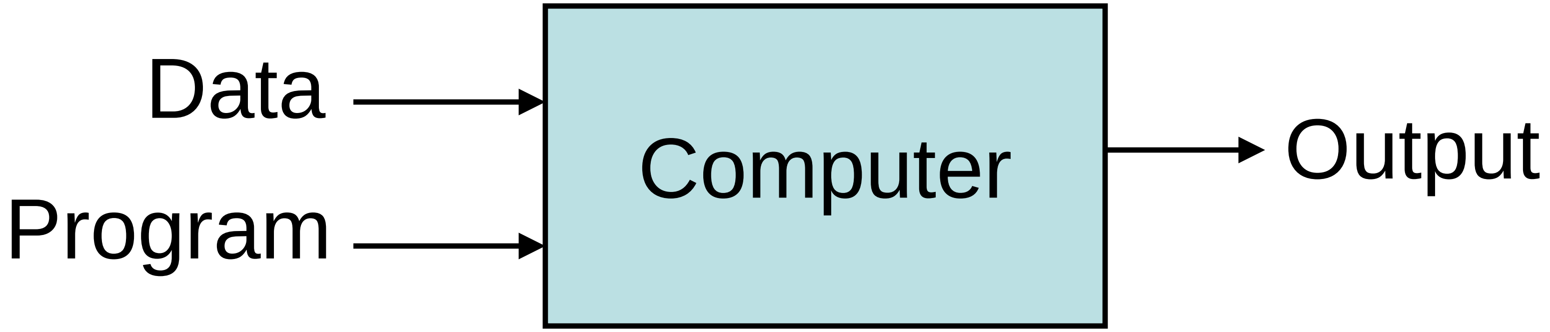
Machine Learning:

Study of algorithms that

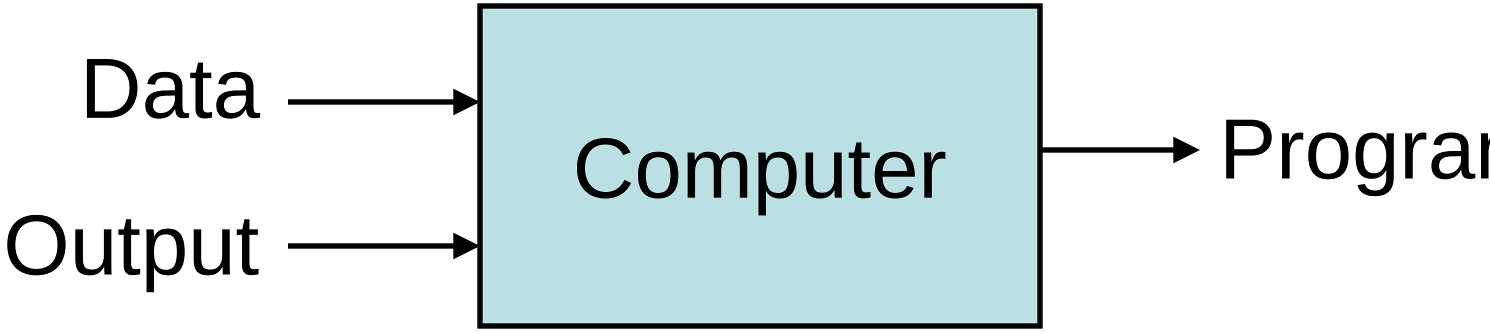
- improve their performance P
- at some task T
- with experience E

well-defined learning task: <P,T,E>

Traditional Programming



Machine Learning



Styles of machine learning

Human have many learning styles
How about machine?

Supervised Learning

- machine performs function (e.g., classification) after training on a data set where inputs and desired outputs are provided
  like decision trees

Unsupervised Learning

- Learning useful structure without labeled classes, optimization criterion, feedback signal, or any other information beyond the raw data
  like clustering

Semi-supervised Learning

??? Getting important in ML
Use unlabeled data to augment a small labeled sample to improve learning?

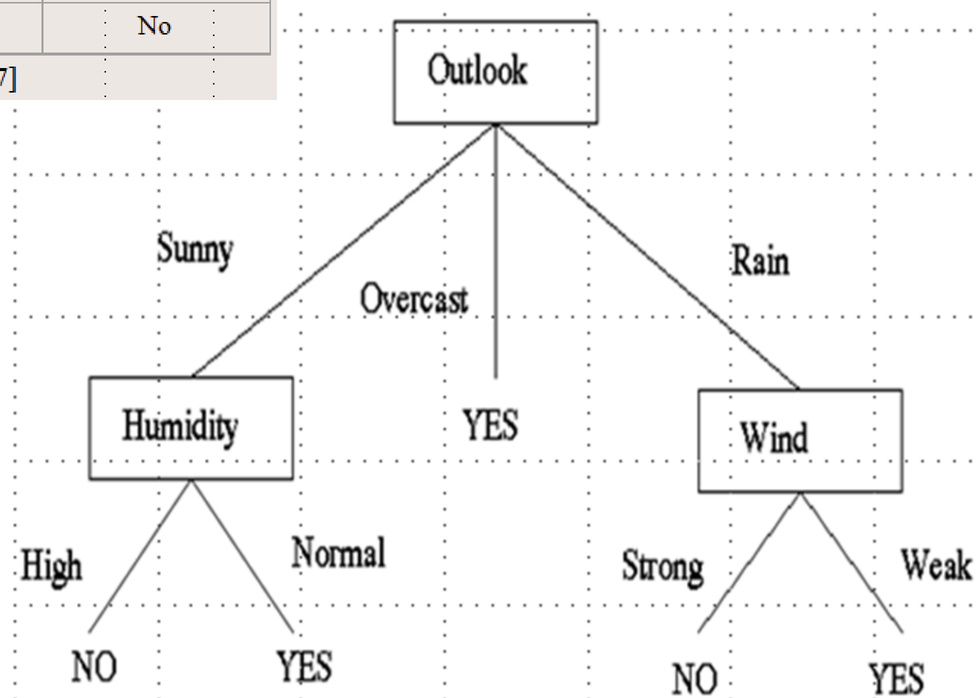
Decision Tree Learning

- **Learning Decision Trees**
  - Decision tree induction is a simple but powerful learning paradigm. In this method a set of training examples is broken down into smaller and smaller subsets while at the same time an associated decision tree get incrementally developed. At the end of the learning process, a decision tree covering the training set is returned.
  - The decision tree can be thought of as a set sentences (in Disjunctive Normal Form) written propositional logic.
  - Some characteristics of problems that are well suited to Decision Tree Learning are:
    - Attribute-value paired elements
    - Discrete target function
    - Disjunctive descriptions (of target function)
    - Works well with missing or erroneous training data

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

[See: Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997]

An example:



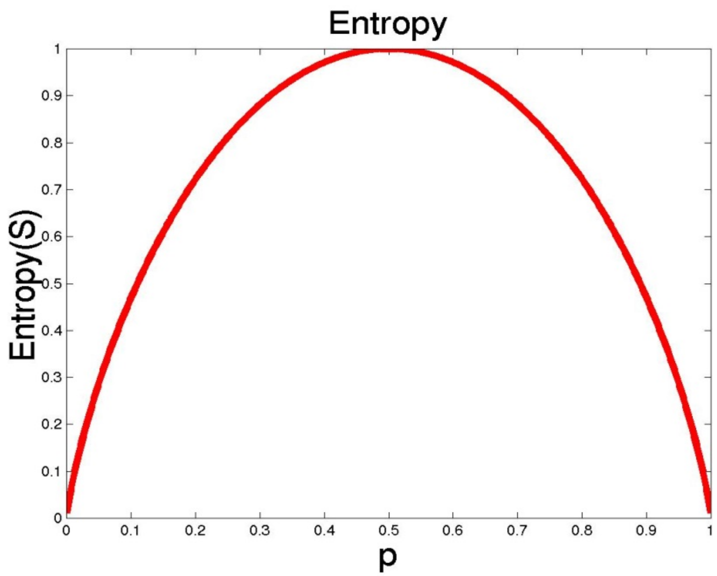
Building a Decision Tree

1. First test all attributes and select the on that would function as the best root;
2. Break-up the training set into subsets based on the branches of the root node;
3. Test the remaining attributes to see which ones fit best underneath the branches of the root node;
4. Continue this process for all other branches until
   a. all examples of a subset are of one type
   b. there are no examples left (return majority classification of the parent)
   c. there are no more attributes left (default value should be majority classification)

Determining which attribute is best (Entropy & Gain)

- Entropy (E) is the minimum number of bits needed in order to classify an arbitrary example as yes or no
- $E(S) = \Sigma^{c}_{i=1} -p_i \log_2 p_i$,
  - Where S is a set of training examples,
  - c is the number of classes, and
  - p_i is the proportion of the training set that is of class i
- For our entropy equation $0 \log_2 0 = 0$
- The information gain G(S,A) where A is an attribute
- $G(S,A) \equiv E(S) - \Sigma_{v \text{ in Values}(A)} \quad (|S_v| \ / \ |S|) * E(Sv)$

Entropy



- S is a sample of training examples
- p_+ is the proportion of positive examples
- p_- is the proportion of negative examples
- Entropy measures the impurity of S

$$\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Decision Trees

example data sets

By calculating information entropy

classifiers
and
prediction models

apply information theory

By Shanon and Weaver (1949)

The unit of information is a bit, and the amount of information in a single binary answer is $\log_2 P(v)$, where $P(v)$ is the probability of event v occurring.

Information needed for a correct answer,

$E(S) = I(p/(p+n), n/(p+n)) = -(p/(p+n) \log_2 p/(p+n)) - n/(p+n) \log_2 n/(p+n))$

Information contained in the remained sub-trees,

$\text{Remainder}(A) = \Sigma(p_i + n_i)/(p+n)\, I(p_i/(p_i + n_i), n_i/(p_i + n_i))$

$\text{Gain}(A) = I(p/(p+n), n/(p+n)) - \text{Remainder}(A)$

disorder

Example 3

Outlook	Play Tennis
Sunny	No
Sunny	No
Overcast	Yes
Rain	Yes
Rain	Yes
Rain	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rain	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rain	No

P(Outlook = Rain and Play Tennis = yes) = 3/5
P(Outlook = Rain and Play Tennis = no) = 2/5

$$Entropy(S_{rain}) = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = .971$$

$$Entropy(S_{overcast}) = -\frac{4}{4}\log_2\left(\frac{4}{4}\right) - 0\log_2(0) = 0$$

$$Entropy(S_{sunny}) = -\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) = .971$$

P(rain) = 5/14 P(overcast) = 4/14 P(sunny) = 5/14

$$\text{Entropy (Play Tennis | Outlook)} = -\frac{5}{14}(.971) - \frac{4}{14}(0) - \frac{5}{14}(.971) = .694$$

By knowing Outlook, how much information have I gained?

Entropy (Play Tennis) - Entropy (Play Tennis | Outlook) =
.940 - .694 = .246

P(Play Tennis = Yes) = 9/14
P(Play Tennis = No) = 5/14

$$\text{Entropy(Play Tennis)} = -\,9/14 \log_2 (9/14) - 5/14 \log_2 (5/14) = .940$$

$$E(S) = \Sigma^{c}_{i=1} -p_i \log_2 p_i ,$$

Information Gain

- The information gain of a feature F is the expected reduction in entropy resulting from splitting on this feature.

$$Gain(S,F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where S_v is the subset of S having value v for feature F.

- Entropy of each resulting subset weighted by its relative size.
- Example:

Ball	Size	Color	Weight	Rubber?	Result (Bounces?)
1	Small	green	Light	yes	yes
2	Small	blue	Medium	no	no
3	Medium	red	Medium	no	no
4	Small	red	Medium	yes	yes
5	Large	green	Heavy	yes	yes
6	Medium	blue	Heavy	yes	no
7	Medium	green	Heavy	yes	no
8	Small	red	Light	no	no

Figure C1: Identification Tree Training Data

S= Result (bounces?)
F = Size
|S|=8
V=1: Small
  2: Large
  3: Medium
$|S_1| = 4$
$|S_2| = 1$
$|S_3| = 3$

$E(S)= I(p/(p+n), n/(p+n)) = - (p/(p+n) \log_2 p/(p+n)) - n/(p+n)\log_2 n/(p+n))$

$|S|=8$

$E(S) = - 3/8*\log_2(3/8) – 5/8*\log_2(5/8) = 0.954434$

$$Gain(S,F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

Gain(S, Size) =?
Gain(S, Color) =?
Gain(S, Weight) =?
Gain(S, Rubber) =?

Ball	Size	Color	Weight	Rubber?	Result (Bounces?)
1	Small	green	Light	yes	yes
2	Small	blue	Medium	no	no
3	Medium	red	Medium	no	no
4	Small	red	Medium	yes	yes
5	Large	green	Heavy	yes	yes
6	Medium	blue	Heavy	yes	no
7	Medium	green	Heavy	yes	no
8	Small	red	Light	no	no

Figure C1: Identification Tree Training Data

Four possible splitting:

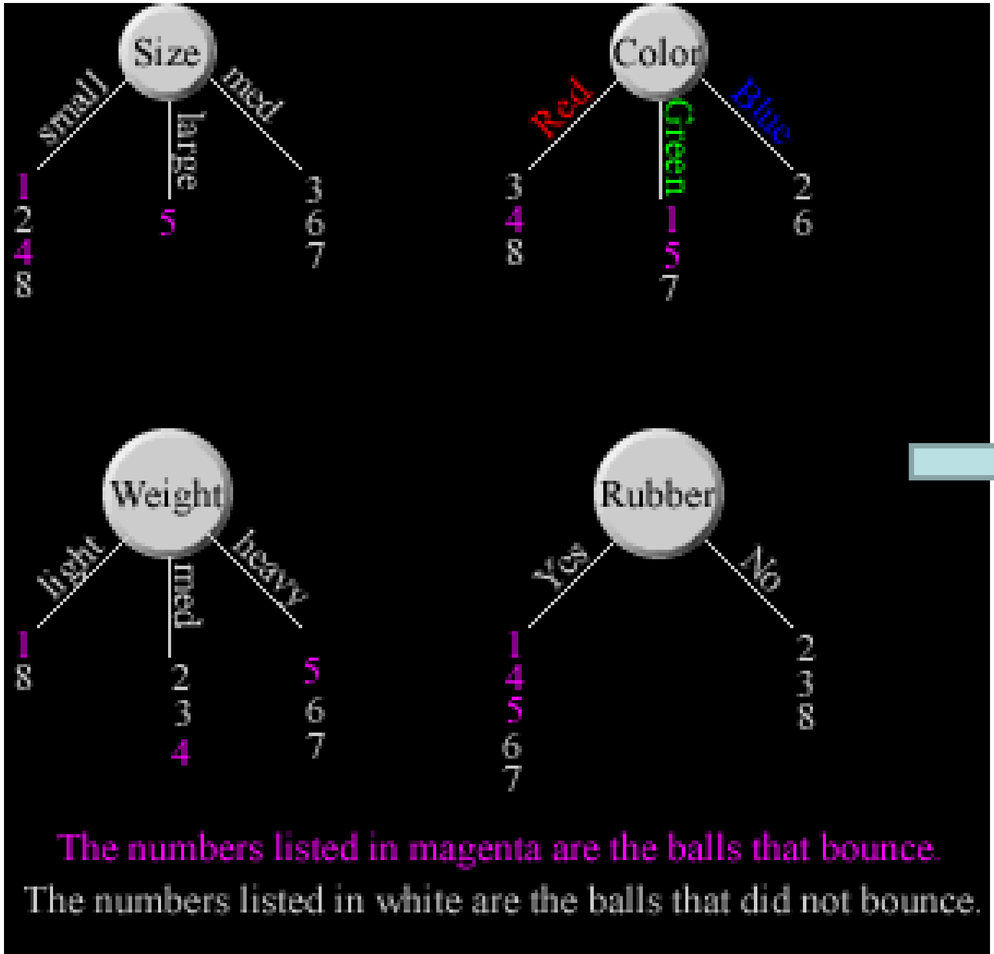
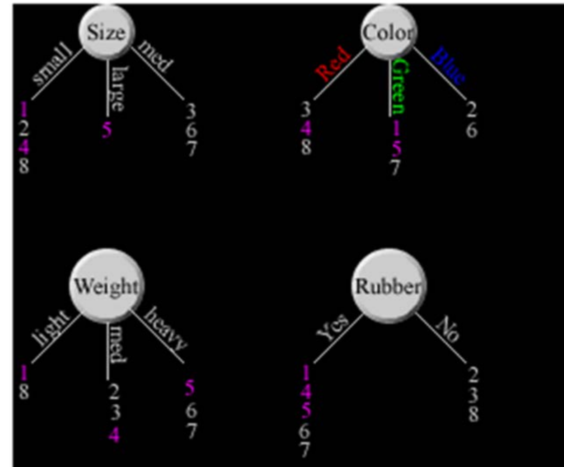


Figure C2: First step trees

Qs:
Which is better?
Which is the best?

Ball	Size	Color	Weight	Rubber?	Result (Bounces?)
1	Small	green	Light	yes	yes
2	Small	blue	Medium	no	no
3	Medium	red	Medium	no	no
4	Small	red	Medium	yes	yes
5	Large	green	Heavy	yes	yes
6	Medium	blue	Heavy	yes	no
7	Medium	green	Heavy	yes	no
8	Small	red	Light	no	no

Figure C1: Identification Tree Training Data



The numbers listed in magenta are the balls that bounce.
The numbers listed in white are the balls that did not bounce.

Figure C2: First step trees

(0.954434)

$$Gain(S,F) = Entropy(S) - \sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)$$

```
Size_Disorder = Σb (nb/nt) * (Σc -(nbc/nb)log2(nbc/nb))
= (4/8) * ((-(2/4) log2 (2/4))
  + (-(2/4) log2 (2/4))) + ((1/8) * 0) + ((3/8) * 0)
= 0.5
```

How about color_Disorder?
weight_Disorder?
rubber_Disorder?

Color: 0.69
Weight: 0.94
Rubber: 0.61

S:Size $|S|=8$
V=1: Small
2: Large
3: Medium
$|S_1| = 4$
$|S_2| = 1$
$|S_3| = 3$

Ball	Size	Color	Weight	Rubber?	Result (Bounces?)
1	Small	green	Light	yes	yes
2	Small	blue	Medium	no	no
3	Medium	red	Medium	no	no
4	Small	red	Medium	yes	yes
5	Large	green	Heavy	yes	yes
6	Medium	blue	Heavy	yes	no
7	Medium	green	Heavy	yes	no
8	Small	red	Light	no	no

Figure C1: Identification Tree Training Data

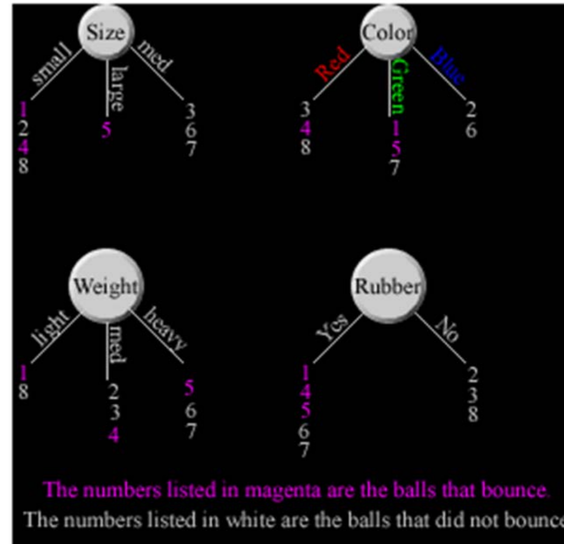


Figure C2: First step trees

$$Gain(S,F) = Entropy(S) - \boxed{\sum_{v \in Values(F)} \frac{|S_v|}{|S|} Entropy(S_v)}$$

Disorder

$$Size\_Disorder = \Sigma_b \ (nb/nt) * (\Sigma_c \ -(nbc/nb) \log_2 (nbc/nb))$$

$$= (4/8) * ((-(2/4) \log_2 (2/4))$$

$$+ (-(2/4) \log_2 (2/4))) + ((1/8) * 0) + ((3/8) * 0)$$

$$= 0.5$$

Color_Disorder = 0.69
Weight_Disorder = 0.94
Rubber_Disorder = 0.61

(1) Work in Class: Please write down their formulae.

For the case of Size = small, continue to split this note

Ball	Size	Color	Weight	Rubber?	Result (Bounces?)
1	Small	green	Light	yes	yes
2	Small	blue	Medium	no	no
~~3~~	~~Medium~~	~~red~~	~~Medium~~	~~no~~	~~no~~
4	Small	red	Medium	yes	yes
~~5~~	~~Large~~	~~green~~	~~Heavy~~	~~yes~~	~~yes~~
~~6~~	~~Medium~~	~~blue~~	~~Heavy~~	~~yes~~	~~no~~
~~7~~	~~Medium~~	~~green~~	~~Heavy~~	~~yes~~	~~no~~
8	Small	red	Light	no	no

Figure C1: Identification Tree Training Data

(2) Work in Class: Please write down their formulae.

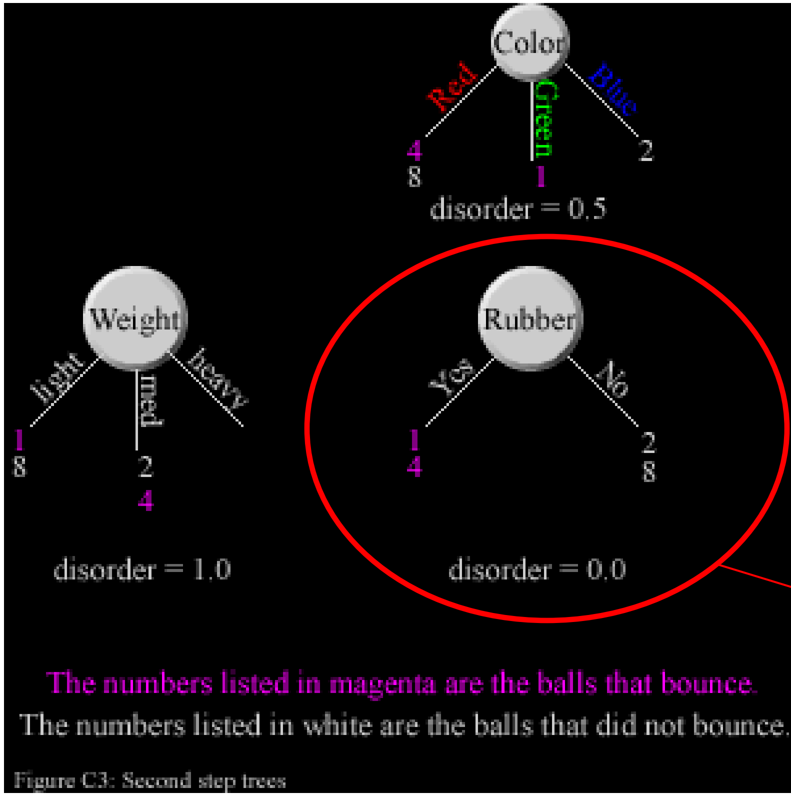


Figure C3: Second step trees

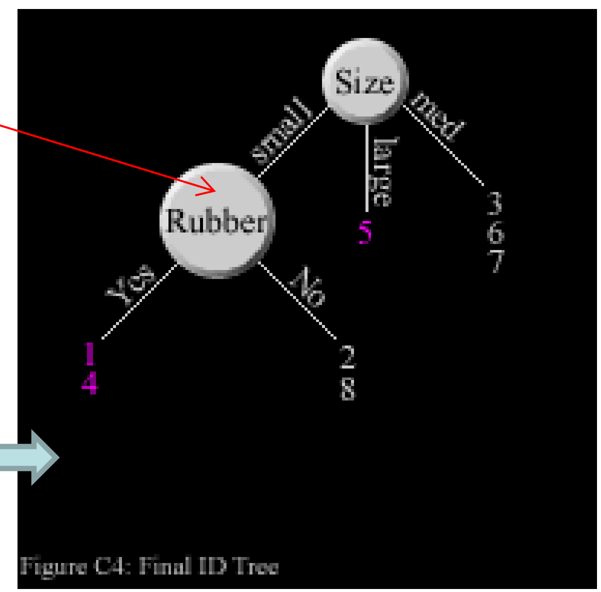


Figure C4: Final ID Tree

How about other two cases?
Split or not? Why?
- medium?
- large?

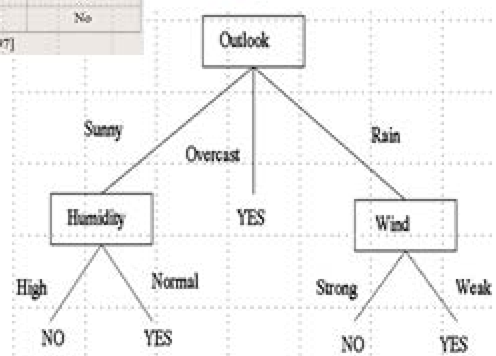
Finish splitting?
Why?

Home work

Day	Outlook	Temperature	Humidity	Wind	PlayTennis
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

[See: Tom M. Mitchell, *Machine Learning*, McGraw-Hill, 1997]

An example:



Write down all formulae of creating decision tree (why selecting Outlook as root node, and Humidity and Wind as the children nodes in) based on information gain (or remaining disorder) http://www.cs.csi.cuny.edu/~imberman/ai/Entropy%20and%20Information%20Gain.htm

conditional entropy for rain

Outlook	Play Tennis
Sunny	No
Sunny	No
Overcast	Yes
Rain	Yes
Rain	Yes
Rain	No
Overcast	Yes
Sunny	No
Sunny	Yes
Rain	Yes
Sunny	Yes
Overcast	Yes
Overcast	Yes
Rain	No

P(Outlook = Rain and Play Tennis = yes) = 3/5
P(Outlook = Rain and Play Tennis = no) = 2/5

$$Entropy(S_{rain}) = -\frac{3}{5}\log_2\left(\frac{3}{5}\right) - \frac{2}{5}\log_2\left(\frac{2}{5}\right) = .971$$

$$Entropy(S_{overcast}) = -\frac{4}{4}\log_2\left(\frac{4}{4}\right) - 0\log_2(0) = 0$$

$$Entropy(S_{sunny}) = -\frac{2}{5}\log_2\left(\frac{2}{5}\right) - \frac{3}{5}\log_2\left(\frac{3}{5}\right) = .971$$

P(rain) = 5/14   P(overcast) = 4/14   P(sunny) = 5/14

$$\text{Entropy (Play Tennis | Outlook)} = -\frac{5}{14}(.971) - \frac{4}{14}(0) - \frac{5}{14}(.971) = .694$$

By knowing Outlook, how much information have I gained?

Entropy (Play Tennis) - Entropy (Play Tennis | Outlook) =
.940 - .694 = .246

P(Play Tennis = Yes) = 9/14
P(Play Tennis = No) = 5/14

Entropy(Play Tennis) = - 9/14 $\log_2$ (9/14) - 5/14 $\log_2$ (5/14) = .940

- $E(S) = \Sigma^{c}_{i=1} -p_i \log_2 p_i$,

Implementation of a Decision Tree

L8-src¥DecisionTree.txt

```
// compute information content,
// given # of pos and neg examples
 double computeInfo(int p, int n) {
    double total = p + n ;
    double pos = p / total ;
    double neg = n / total;
```

$$\text{Remainder}(A) = \Sigma(p_i + n_i) /(p+n)\ I(p_i/(p_i + n_i), n_i/(p_i + n_i))$$

```
    } else {
     temp = (-1.0 * (pos * Math.log(pos)/Math.log(2)))
                 - (neg * Math.log(neg)/Math.log(2)) ;
   }
    return temp ;
 }
```

```
double computeRemainder(Variable variable,
                                  Vector examples)
 {
   int positive[] = new int[variable.labels.size()];
   int negative[] = new int[variable.labels.size()];
   int index = variable.column;
```

```
   double numRecs = examples.size() ;
   for( int i=0 ; i < numValues ; i++) {
     String value = variable.getLabel(i);
     Enumeration enum = examples.elements();
     while (enum.hasMoreElements()) {
       String record[] = (String[])enum.nextElement();
       // get next record
       if (record[index].equals(value)) {
        if (record[classIndex].equals("yes")) {
            positive[i]++;
        } else  {
            negative[i]++;
        }
       }
     } /* endwhile */
```

```
  double weight = (positive[i]+negative[i]) / numRecs;
  double myrem = weight * computeInfo(positive[i],
                                      negative[i]);
   sum = sum + myrem ;
  } /* endfor */
  return sum ;
}
```

Implementation of a Decision Tree

L8-src¥DecisionTree.txt

```
// compute information content,
// given # of pos and neg examples
 double computeInfo(int p, int n) {
    double total = p + n ;
    double pos = p / total ;
    double neg = n / total;
    double temp;
    if ((p ==0) || (n == 0)) {
      temp = 0.0 ;
    } else {
     temp = (-1.0 * (pos * Math.log(pos)/Math.log(2)))
              - (neg * Math.log(neg)/Math.log(2)) ;
   }
    return temp ;
 }
```

$$E(S)= I(p/(p+n),\ n/(p+n)) = -\ (p/(p+n)\ \mathit{log}_2\, p/(p+n)\ )\ -\ n/(p+n)\mathit{log}_2\, n/(p+n)\ )$$

```
double computeRemainder(Variable variable,
                                              Vector examples)
 {
   int positive[] = new int[variable.labels.size()];
   int negative[] = new int[variable.labels.size()];
   int index = variable.column;
   int classIndex = classVar.column;
   double sum = 0 ;
   double numValues = variable.labels.size();
   double numRecs = examples.size() ;
   for( int i=0 ; i < numValues ; i++) {
     String value = variable.getLabel(i);
     Enumeration enum = examples.elements();
     while (enum.hasMoreElements()) {
       String record[] = (String[])enum.nextElement();
       if (record[index].equals(value)) {
        if (record[classIndex].equals("yes")) {
            positive[i]++;
        } else  {
            negative[i]++;
        }
       }
     } /* endwhile */
```

```
  double weight = (positive[i]+negative[i]) / numRecs;
  double myrem = weight * computeInfo(positive[i],
                                      negative[i]);
   sum = sum + myrem ;
  } /* endfor */
  return sum ;
}
```

Implementation of a Decision Tree

L8-src¥DecisionTree.txt

```
// compute information content,
// given # of pos and neg examples
 double computeInfo(int p, int n) {
    double total = p + n ;
    double pos = p / total ;
    double neg = n / total;
    double temp;
    if ((p ==0) || (n == 0)) {
      temp = 0.0 ;
    } else {
    }
    return temp ;
 }
```

$$Remainder(A) = \Sigma(p_i + n_i)/(p+n)\, I(p_i/(p_i + n_i), n_i/(p_i + n_i))$$

```
double computeRemainder(Variable variable,
                                      Vector examples)
 {
   int positive[] = new int[variable.labels.size()];
   int negative[] = new int[variable.labels.size()];
   int index = variable.column;
   int classIndex = classVar.column;
   double sum = 0 ;
   double numValues = variable.labels.size();
   double numRecs = examples.size() ;
     Enumeration enum = examples.elements();
     while (enum.hasMoreElements()) {
       String record[] = (String[])enum.nextElement();
       // get next record
       if (record[index].equals(value)) {
        if (record[classIndex].equals("yes")) {
           positive[i]++;
        } else  {
           negative[i]++;
        }
       }
     } /* endwhile */
```

```
  double weight = (positive[i]+negative[i]) / numRecs;
  double myrem = weight * computeInfo(positive[i],
                                      negative[i]);

   sum = sum + myrem ;
  } /* endfor */
  return sum ;
}
```

Implementation of a Decision Tree

L8-src¥DecisionTree.txt

```
// compute information content,
// given # of pos and neg examples
 double computeInfo(int p, int n) {
    double total = p + n ;
    double pos = p / total ;
    double neg = n / total;
    double temp;
    if ((p ==0) || (n == 0)) {
      temp = 0.0 ;
```

$$Remainder(A) = \Sigma(p_i+ n_i) /(p+n)\ I(p_i/(p_i+ n_i), n_i/(p_i+ n_i))$$

```
    return temp ;
 }
```

```
double computeRemainder(Variable variable,
                                    Vector examples)
 {
   int positive[] = new int[variable.labels.size()];
   int negative[] = new int[variable.labels.size()];
   int index = variable.column;
   int classIndex = classVar.column;
   double sum = 0 ;
   double numValues = variable.labels.size();
```

```
     while (enum.hasMoreElements()) {
       String record[] = (String[])enum.nextElement();
       // get next record
       if (record[index].equals(value)) {
        if (record[classIndex].equals("yes")) {
           positive[i]++;
        } else  {
           negative[i]++;
        }
       }
     } /* endwhile */
```

```
  double weight = (positive[i]+negative[i]) / numRecs;
  double myrem = weight * computeInfo(positive[i],
                                      negative[i]);
   sum = sum + myrem ;
  } /* endfor */
  return sum ;
}
```

Implementation of a Decision Tree

```
// return the variable with most gain
 Variable chooseVariable(Hashtable variables, Vector examples)
 {
   Enumeration enum = variables.elements() ;
   double gain = 0.0, bestGain = 0.0 ;
   Variable best = null ;
   int counts[] ;
   counts = getCounts(examples) ;
   int pos = counts[0] ;
   int neg = counts[1] ;
   double info = computeInfo(pos, neg);

   while(enum.hasMoreElements()) {
      Variable tempVar = (Variable)enum.nextElement() ;
      gain = info - computeRemainder(tempVar, examples);

      if (gain > bestGain) {
        bestGain = gain ;
        best = tempVar;
      }

   }
   return best;  //
 }
```

Implementation of a Decision Tree

```
// return the variable with most gain
 Variable chooseVariable(Hashtable variables, Vector examples)
 {
   Enumeration enum = variables.elements() ;
   double gain = 0.0, bestGain = 0.0 ;
   Variable best = null ;
   int counts[] ;
   counts = getCounts(examples) ;
   int pos = counts[0] ;
   int neg = counts[1] ;
   double info = computeInfo(pos, neg);
```

$$Gain(A) = I(p/(p+n), n/(p+n)) - Remainder(A)$$

```
      gain = info - computeRemainder(tempVar, examples);

      if (gain > bestGain) {
        bestGain = gain ;
        best = tempVar;
      }

   }
   return best;  //
 }
```

Which has the best gain?

Gain(S, Size) =?
Gain(S, Color) =?
Gain(S, Weight) =?
Gain(S, Rubber) =?

Demo

- A decision tree. (Run LearnApplet.java in Eclipse)

  C:Huang/Java2012/AI-2/(bin,src)/decisionTree/……

  L8-src¥LearnApplet1.zip

- Example data

  L8-src¥LearnApplet1¥resttree.dat.txt

  resttree.dat

  resttree.dfn

Results:

```
Starting DecisionTree
Info = 1.0
reservation gain = 0.020720839623907805
alternate gain = 0.0
FriSat gain = 0.020720839623907805
hungry gain = 0.19570962879973086
price gain = 0.19570962879973075
patrons gain = 0.5408520829727552
waitEstimate gain = 0.20751874963942196
bar gain = 0.0
rtype gain = 1.1102230246251565E-16
raining gain = 0.0
Choosing best variable: patrons
 Subset - there are 4 records with patrons = some
 Subset - there are 6 records with patrons = full
Info = 0.9182958340544896
reservation gain = 0.2516291673878229
alternate gain = 0.10917033867559889
FriSat gain = 0.10917033867559889
hungry gain = 0.2516291673878229
price gain = 0.2516291673878229
```

```
patrons gain = 0.0
waitEstimate gain = 0.2516291673878229
bar gain = 0.0
rtype gain = 0.2516291673878229
raining gain = 0.10917033867559889
Choosing best variable: reservation
 Subset - there are 2 records with reservation = yes
 Subset - there are 4 records with reservation = no
Info = 1.0
reservation gain = 0.0
alternate gain = 0.31127812445913283
FriSat gain = 0.31127812445913283
hungry gain = 0.31127812445913283
price gain = 0.0
patrons gain = 0.0
waitEstimate gain = 0.5
bar gain = 0.0
rtype gain = 0.0
raining gain = 0.31127812445913283
Choosing best variable: waitEstimate
 Subset - there are 0 records with waitEstimate = 0-10
 Subset - there are 2 records with waitEstimate = 30-60
```

Output:

```
Info = 1.0
reservation gain = 0.0
alternate gain = 0.0
FriSat gain = 1.0
hungry gain = 0.0
price gain = 0.0
patrons gain = 0.0
waitEstimate gain = 0.0
bar gain = 1.0
rtype gain = 1.0
raining gain = 0.0
Choosing best variable: FriSat
 Subset - there are 1 records with FriSat = no
 Subset - there are 1 records with FriSat = yes
 Subset - there are 1 records with waitEstimate = 10-30
 Subset - there are 1 records with waitEstimate = >60
 Subset - there are 2 records with patrons = none
DecisionTree -- classVar = ClassField
Interior node - patrons
Link - patrons=some
Leaf node - yes
Link - patrons=full
```

```
Interior node - reservation
Link - reservation=yes
Leaf node - no
Link - reservation=no
Interior node - waitEstimate
Link - waitEstimate=0-10
Leaf node - yes
Link - waitEstimate=30-60
Interior node - FriSat
Link - FriSat=no
Leaf node - no
Link - FriSat=yes
Leaf node - yes
Link - waitEstimate=10-30
Leaf node - yes
Link - waitEstimate=>60
Leaf node - no
Link - patrons=none
Leaf node - no
Stopping DecisionTree - success!
```

Info = 1.0

waitEstimate gain = 0.0

raining gain = 0.0

hungry gain = 0.0

price gain = 1.0

FriSat gain = 0.0

bar gain = 1.0

patrons gain = 0.0

alternate gain = 0.0

rtype gain = 1.0

reservation gain = 1.0

Choosing best variable: price

Subset - there are 1 records with price = $$$

Subset - there are 1 records with price = $

Subset - there are 0 records with price = $$

Subset - there are 2 records with waitEstimate = >60

Subset - there are 2 records with patrons = none

DecisionTree -- classVar = ClassField

Interior node - patrons

Link - patrons=some

Leaf node - yes

Link - patrons=full

Interior node - waitEstimate

Link - waitEstimate=0-10

Leaf node - yes

Link - waitEstimate=30-60

Interior node - FriSat

Link - FriSat=no

Leaf node - no

Link - FriSat=yes

Leaf node - yes

Link - waitEstimate=10-30

Interior node - price

Link - price=$$$

Leaf node - no

Link - price=$

Leaf node - yes

Link - price=$$

Leaf node - yes

Link - waitEstimate=>60

Leaf node - no

Link - patrons=none

Leaf node - no

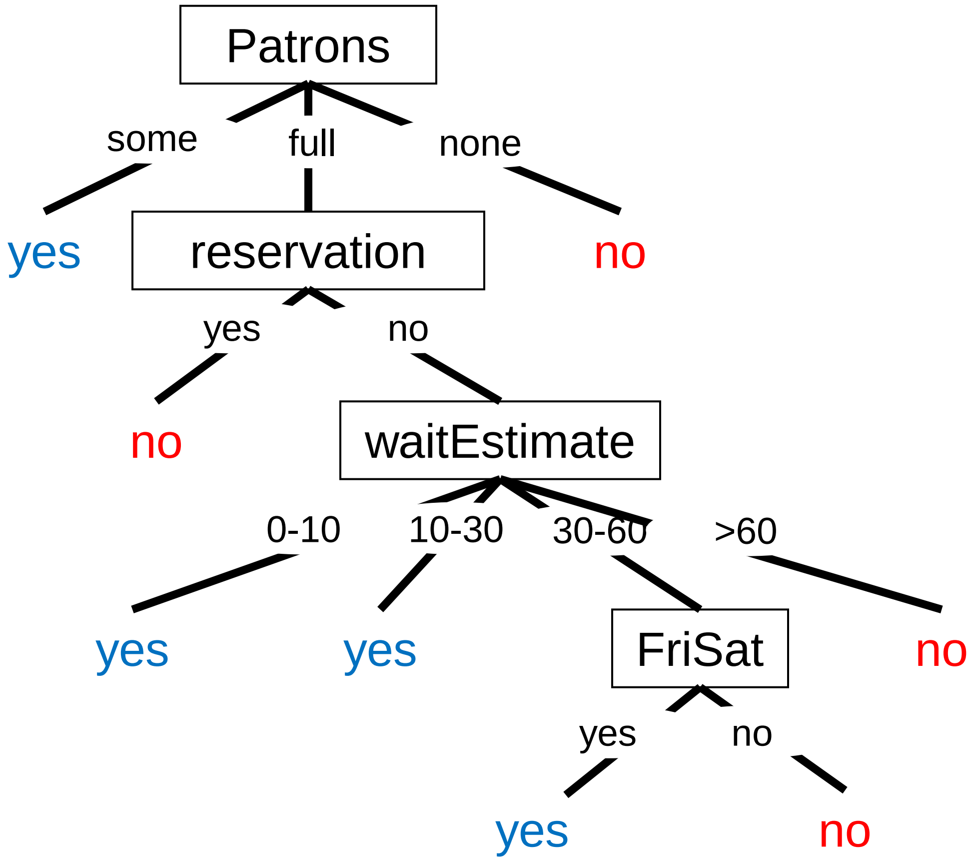
Stopping DecisionTree - success!

Draw a decision tree!

(3) Work in class

Please draw a decision tree for p28 ad p29 the running results of the decision tree!

decision tree from the running results



Whole dataset

alternate	bar	FriSat	hungry	patrons	price	raining	reservation	rtype	waitEstimate	ClassField
yes	no	no	yes	some	$$$	no	yes	French	0-10	yes
yes	no	no	yes	full	$	no	no	Thai	30-60	no
no	yes	no	no	some	$	no	no	Burger	0-10	yes
yes	no	yes	yes	full	$	no	no	Thai	10-30	yes
yes	no	yes	no	full	$$$	no	yes	French	>60	no
no	yes	no	yes	some	$$	yes	yes	Italian	0-10	yes
no	yes	no	no	none	$	yes	no	Burger	0-10	no
no	no	no	yes	some	$$	yes	yes	Thai	0-10	yes
no	yes	yes	no	full	$	yes	no	Burger	>60	no
yes	yes	yes	yes	full	$$$	no	yes	Italian	10-30	no
no	no	no	no	none	$	no	no	Thai	0-10	no
yes	yes	yes	yes	full	$	no	no	Burger	30-60	yes

Subset of dataset

Patrons	reservation	ClassField
full	no	no
	no	yes
	yes	no
	no	no
	yes	no
	no	yes

Reservation	waitEstimate	ClassField
no	30-60	no
	10-30	yes
	>60	no
	30-60	yes

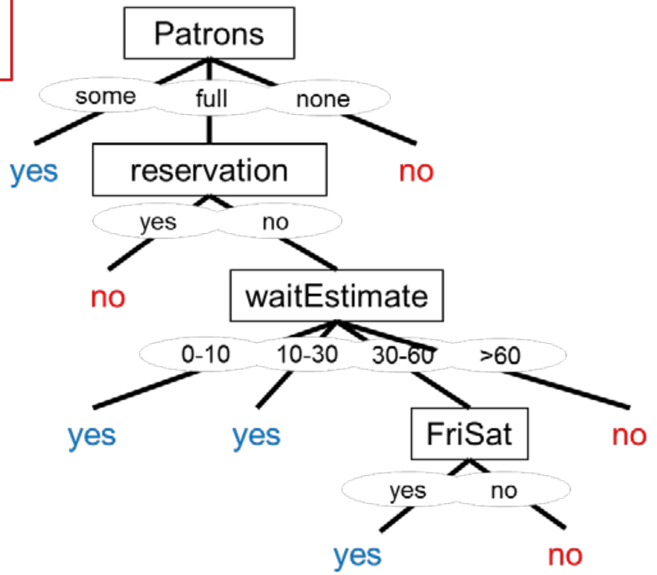
waitEstimate	FriSat	ClassField
30-60	no	no
	yes	yes

Subset of dataset

Patrons	reservation	ClassField
full	no	no
	no	yes
	yes	no
	no	no
	yes	no
	no	yes

Reservation	waitEstimate	ClassField
no	30-60	no
	10-30	yes
	>60	no
	30-60	yes

waitEstimate	FriSat	ClassField
30-60	no	no
	yes	yes



Calculate the following conditional entropy:

Remainder(reservation/patron) =?

Remainder(waitEstimate/reservation) = ?

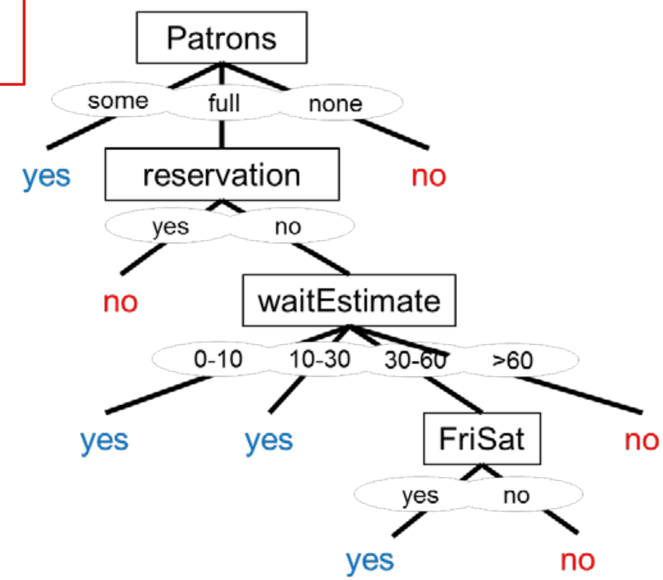
Remainder(FriSat/waitEstimate)= ?

Subset of dataset

Patrons	reservation	ClassField
full	no	no
	no	yes
	yes	no
	no	no
	yes	no
	no	yes

Reservation	waitEstimate	ClassField
no	30-60	no
	10-30	yes
	>60	no
	30-60	yes

waitEstimate	FriSat	ClassField
30-60	no	no
	yes	yes



Calculate

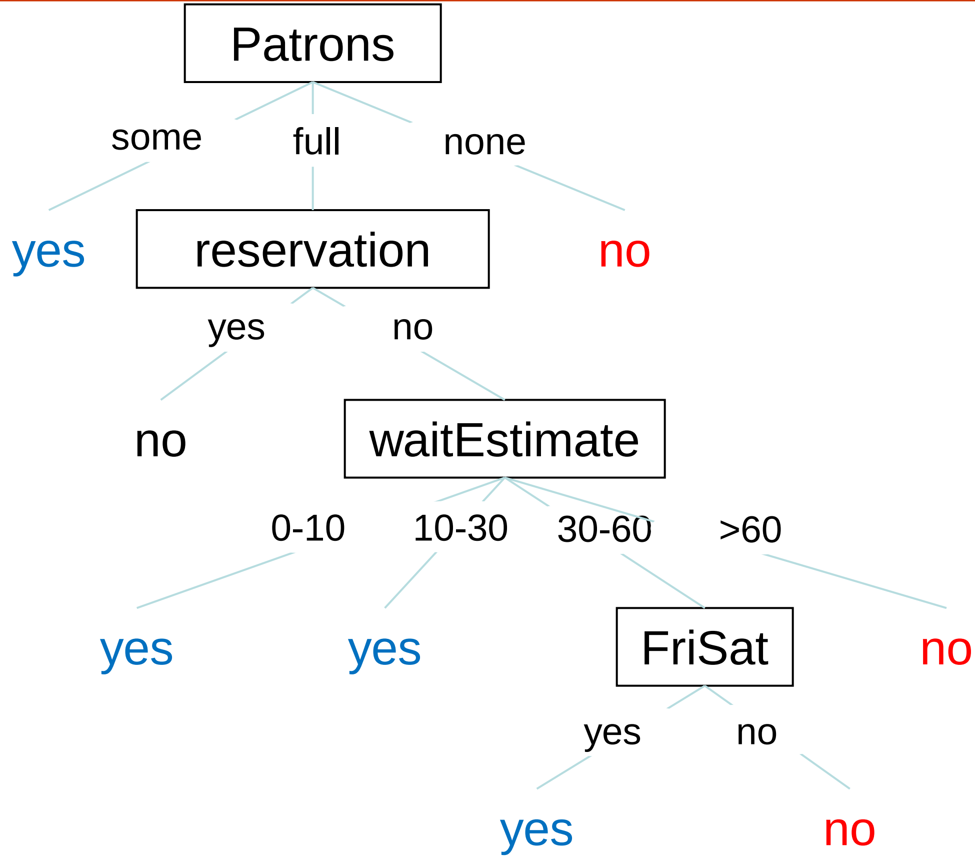
Remainder(reservation/patron) =
2/6*0 + 4/6*(-2/4*log2 (2/4) -2/4*log2 (2/4))

Remainder(waitEstimate/reservation) = ?
1/4*0 + 1/4*0 + 2/4*(-1/2*log2(1/2) -1/2*log2(1/2)) = 0.5

Remainder(FriSat/waitEstimate)= ?
1/2*0 +1/2*0 = 0

(3). Work in class

Please draw a decision tree for p12 ad p13 the running results of the decision tree!



ID Trees to Rules

Once an ID tree is constructed successfully, it can be used to generate a rule-set, which will serve to perform the necessary classifications of the ID tree. This is done by creating a single rule for each path from the root to a leaf in the ID tree.

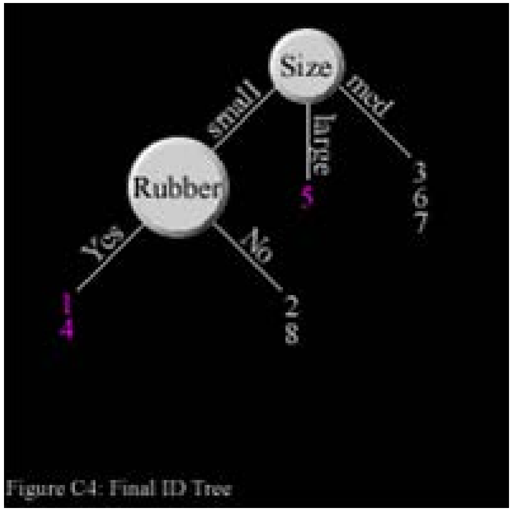


Figure C4: Final ID Tree

R1: if (size = large)
then (ball does bounce)

R2: if (size = medium)
then (ball does not bounce)

R3: if (size = small)
(rubber = no)
then (ball does not bounce)

R4: if (size = small)
(rubber = yes)
then (ball does bounce)

Refined Rules

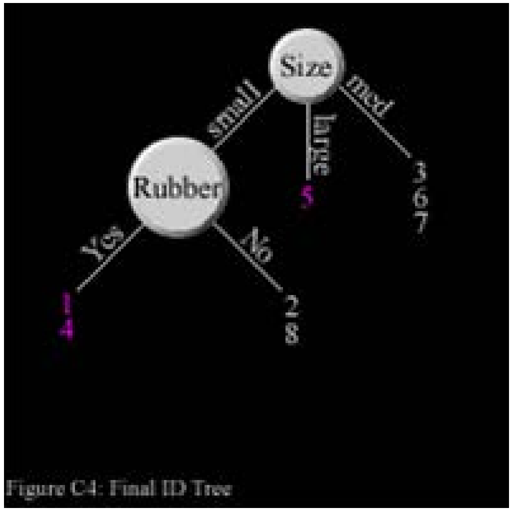


Figure C4: Final ID Tree

R1: if (size = large)
then (ball does bounce)
R2: if (size = medium)
then (ball does not bounce)
R3: if (size = small)
(rubber = no)
then (ball does not bounce)
R4: if (size = small)
(rubber = yes)
then (ball does bounce)

R1: if (size = large)
then (ball does bounce)
R2: if (size = medium)
then (ball does not bounce)
R3: if (rubber = no)
then (ball does not bounce)
R4: if (size = small)
(rubber = yes)
then (ball does bounce)

Rules are used in rule-based (forward chaining or backward chaining) systems.

Eliminating unnecessary rule conditions

R3: if (size = small)
(rubber = no)
then (ball does not bounce)

Ball	Size	Color	Weight	Rubber?	Result (Bounces?)
1	Small	green	Light	yes	yes
2	Small	blue	Medium	no	no
3	Medium	red	Medium	no	no
4	Small	red	Medium	yes	yes
5	Large	green	Heavy	yes	yes
6	Medium	blue	Heavy	yes	no
7	Medium	green	Heavy	yes	no
8	Small	red	Light	no	no

Figure C1: Identification Tree Training Data

Looking at the probability with
event A = (size=small) and event B = (ball does not bounce)

Calculate:
P(B|A) = (3 non rubber balls do not bounce / 8 total) = 0.375
P(B) = (3 non rubber balls do not bounce / 8 total) = 0.375

P(B|A) = P(B) therefore B is independent of A

What does this mean?

A and B → no relation, no dependency

R3: if ~~(size = small)~~
(rubber = no)
then (ball does not bounce)

Eliminating unnecessary rule conditions

Ball	Size	Color	Weight	Rubber?	Result (Bounces?)
1	~~Small~~	~~green~~	~~Light~~	~~yes~~	~~yes~~
2	~~Small~~	~~blue~~	~~Medium~~	~~no~~	~~no~~
3	Medium	red	Medium	no	no
4	Small	red	Medium	yes	yes
~~5~~	~~Large~~	~~green~~	~~Heavy~~	~~yes~~	~~yes~~
6	Medium	blue	Heavy	yes	no
7	Medium	green	Heavy	yes	no
8	~~Small~~	~~red~~	~~Light~~	~~no~~	~~no~~

Figure C1: Identification Tree Training Data

R3: if ~~(size = small)~~
(rubber = no)
then (ball does not bounce)

Looking at the probability with
event A = (rubber=no) and event B = (ball does not bounce)

Calculate:
P(B|A) = (3 balls do not bounce / 8 total) = 3/8
P(B) = (5 balls do not bounce / 8 total) = 5/8

$\boxed{P(B|A) \neq P(B)}$ therefore A and B are not independent

What does this mean?

No change on R3

R3: if (rubber = no)
then (ball does not bounce)

Home Work

Read the following site:

http://ai-depot.com/Tutorial/RuleBased.html