Decision Trees

R&N CH18: Inductive Learning

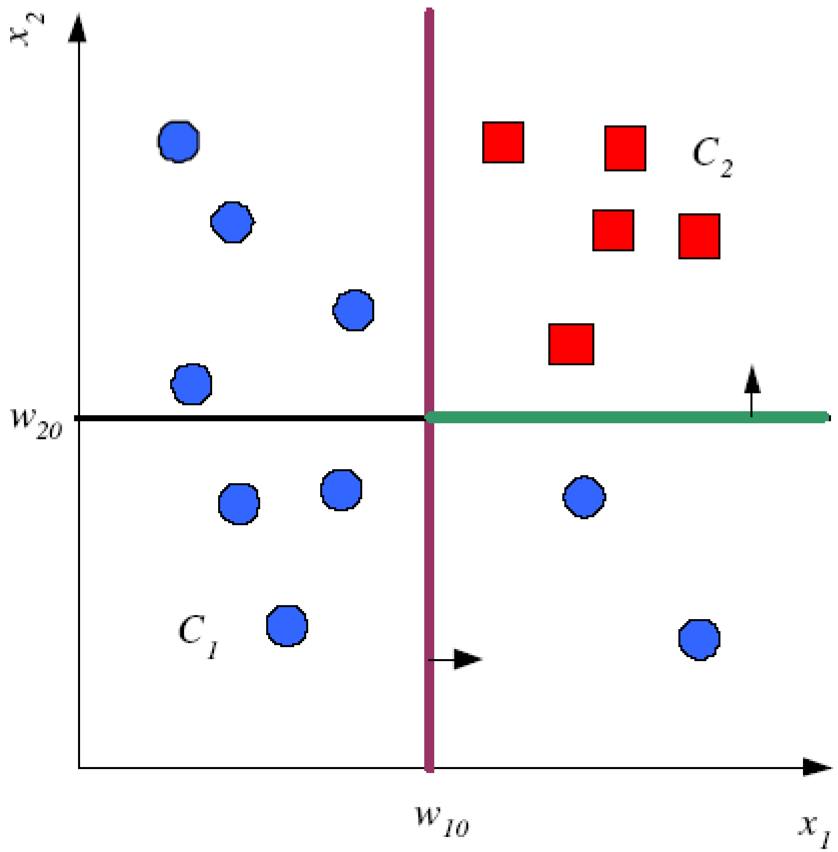
- Learn a function from examples
- f is the target function
- An example is an input-output pair: (x, f(x))

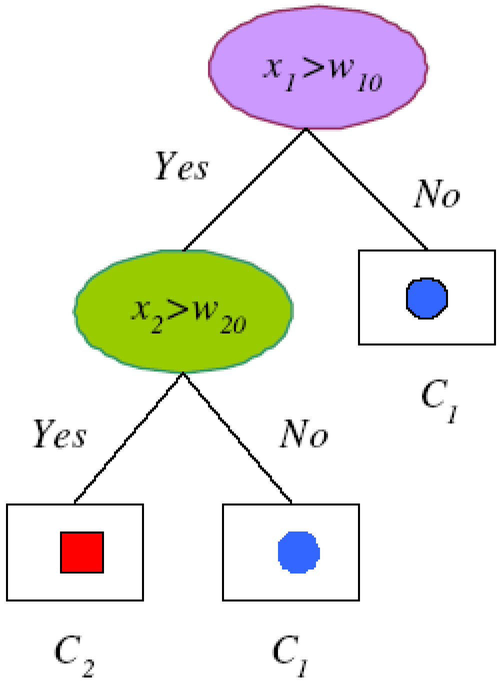
- Problem:
  - Given a hypothesis space *H*
  - Given a training set of examples: $(x_1, f(x_1)), \ldots, (x_n, f(x_n))$
  - Find a hypothesis *h*(*x*) such that *h* ~ f
    - f(x) can be discrete (classification)
    - f(x) can be continuous (regression)

Decision Trees

- Decision tree induction is one of the simplest successful forms of machine learning.
- A decision tree represents a function that:
  - takes as input a vector of attribute values
  - returns a “decision”
    - a single output value
- The input and output values can be discrete or continuous.
- Initially Concentrating on problems where:
  - the inputs have discrete values
  - the output has exactly two possible values
- Boolean Classification w/ Each example input classified
  - Either:
    - true (a positive example)
    - false (a negative example).

Tree Uses Nodes and Leaves





Restaurant Example

- Should SR (Stuart Russell) wait for a table????
- Goal Predicate: WillWait
- Attributes:
  - **Alternate:** whether there is a suitable alternative restaurant nearby.
  - **Bar:** whether the restaurant has a comfortable bar area to wait in.
  - **Fri/Sat:** true on Fridays and Saturdays.
  - **Hungry:** whether we are hungry.
  - **Patrons:** how many people are in the restaurant (values are None, Some, and Full).
  - **Price:** the restaurant's price range ($, $$, $$$).
  - **Raining:** whether it is raining outside.
  - **Reservation:** whether we made a reservation.
  - **Type:** the kind of restaurant (French, Italian, Thai, or burger).
  - **WaitEstimate:** the wait estimated by the host (0–10 minutes, 10–30, 30–60, or >60).

Decision Tree (SR's) w/ Restaurant Example

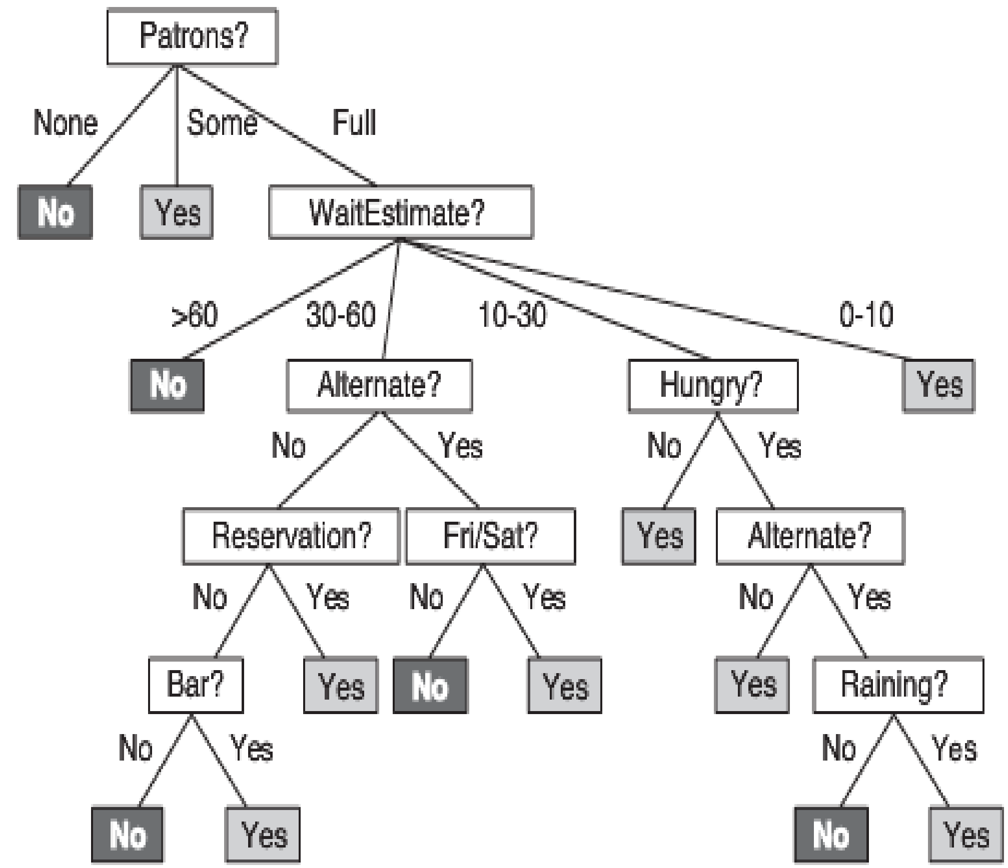
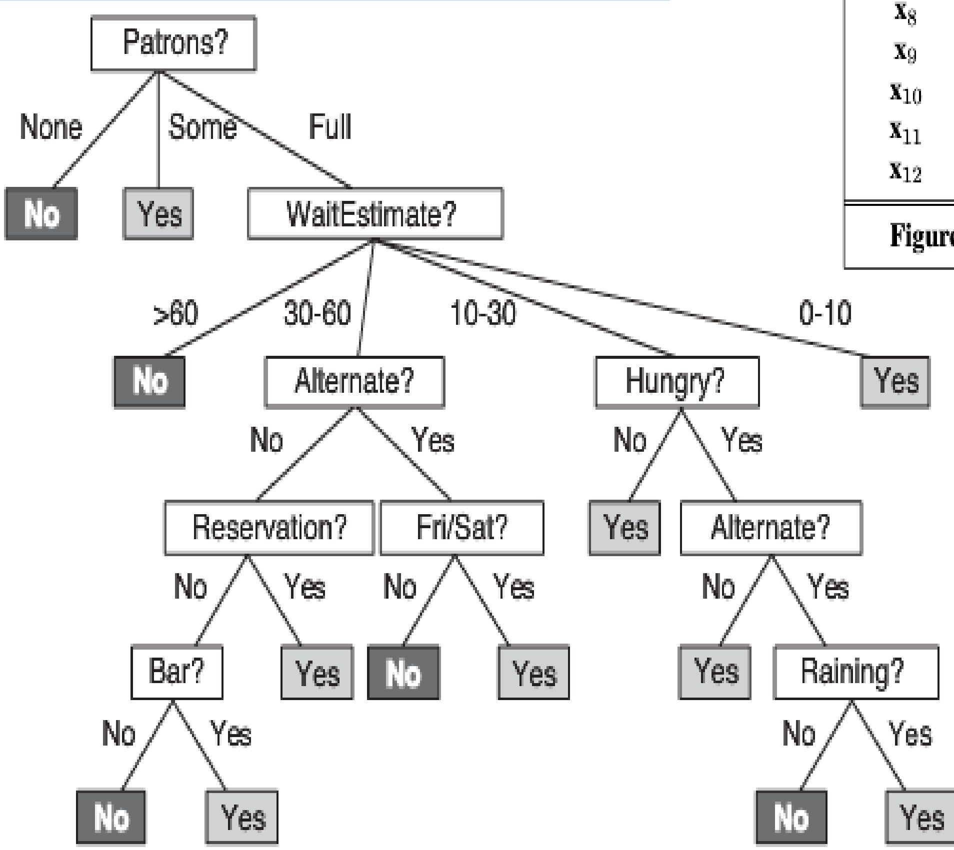


Figure 18.2 A decision tree for deciding whether to wait for a table.

Decision Tree (SR's) w/ Restaurant Example

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*\$\$\$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*\$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*\$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*\$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*\$\$\$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*\$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*\$\$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*\$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$\$\$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*\$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*\$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.



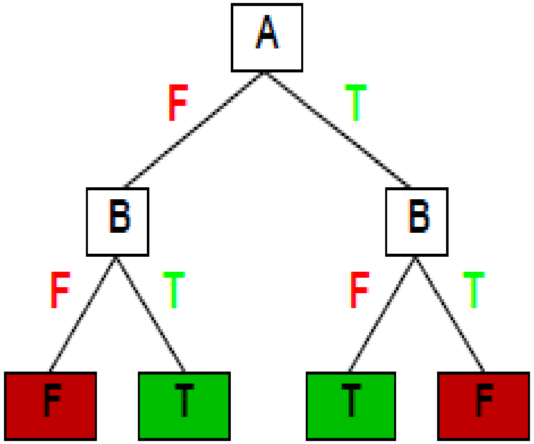
Expressiveness w/ Decision Trees

- A Boolean decision tree is logically equivalent to the assertion:
  - goal attribute is true if and only if the input attributes satisfy one of the paths leading to a leaf with value true
  - Goal $\Leftrightarrow$ ($Path_1 \vee Path_2 \vee \cdots$)
- Equivalent to Disjunctive Normal Form
- Can Represent any Propositional Logic Expression

Expressiveness

Decision trees can express any function of the input attributes.
E.g., for Boolean functions, truth table row → path to leaf:

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Trivially, there is a consistent decision tree for any training set w/ one path to leaf for each example (unless f nondeterministic in x) but it probably won't generalize to new examples

Prefer to find more compact decision trees

Consistency vs. simplicity

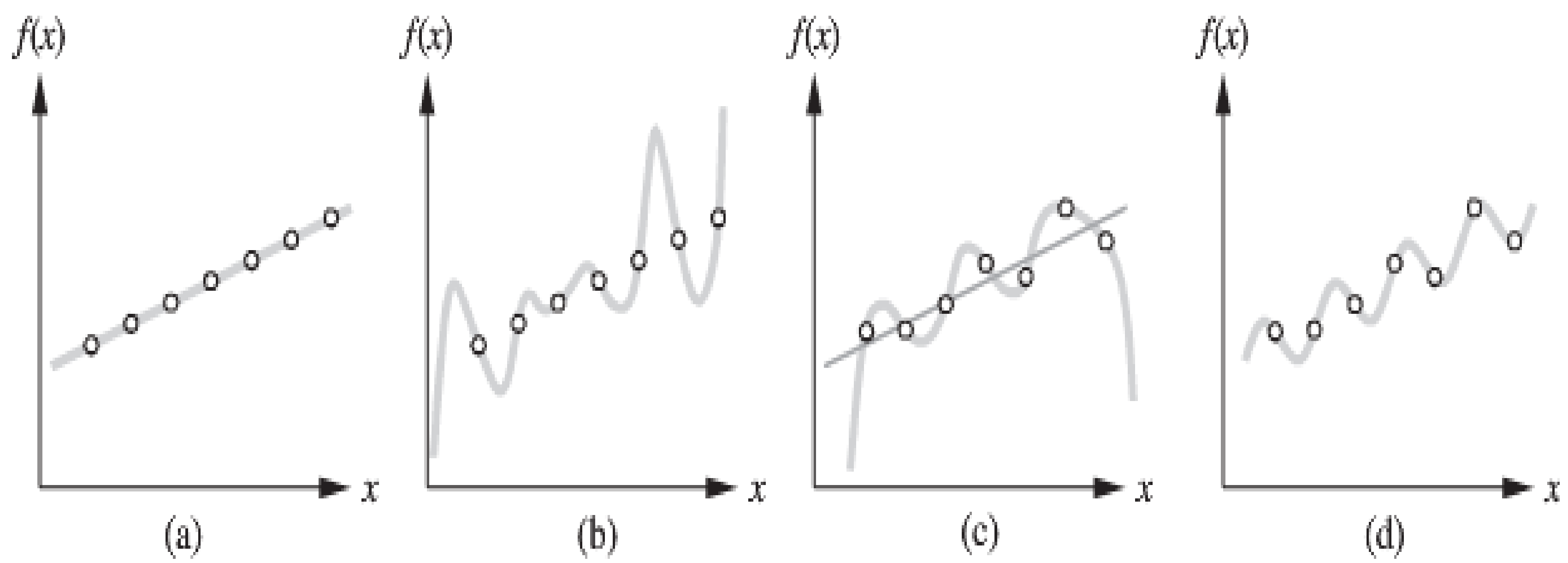
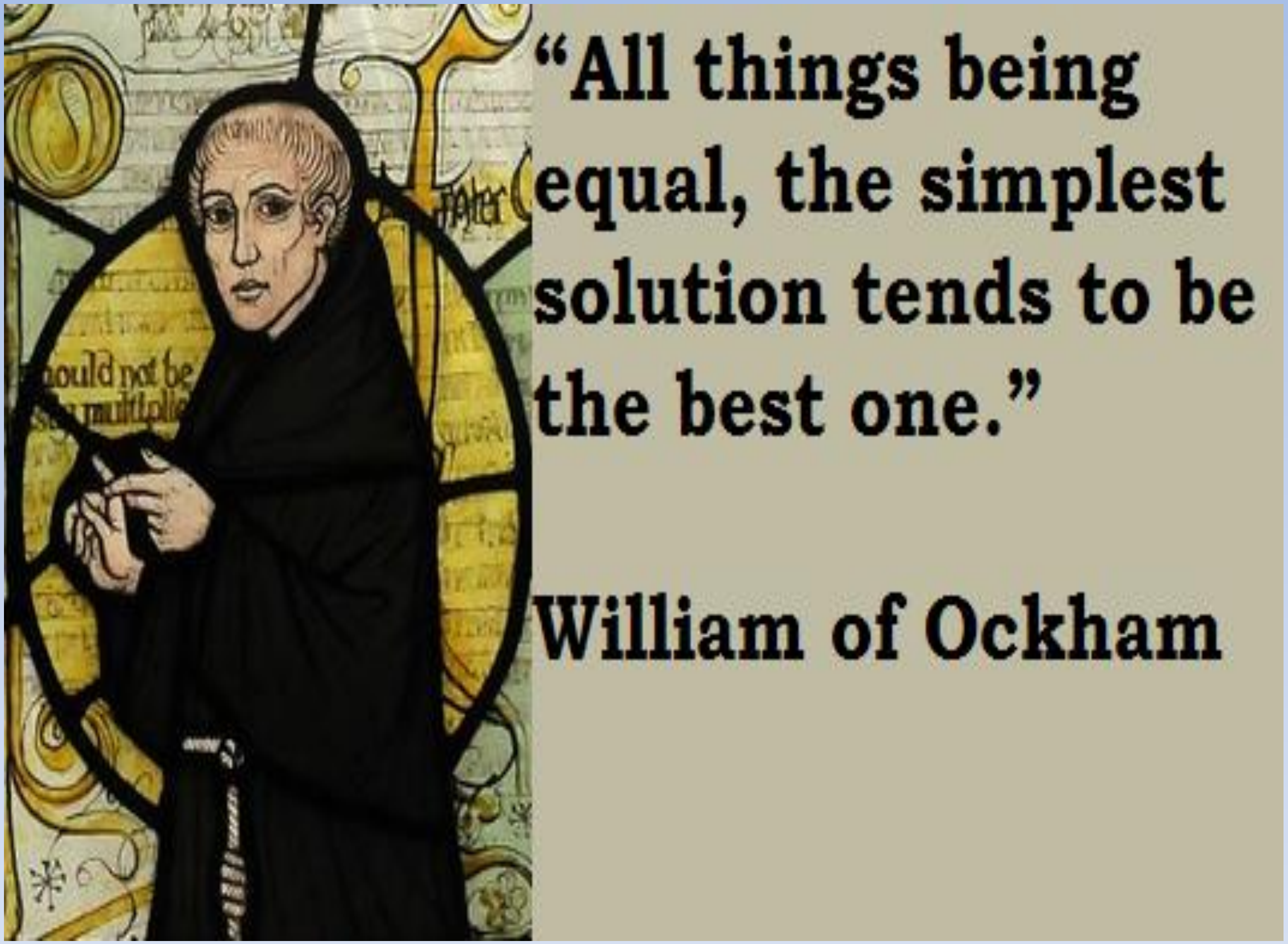


Figure 18.1 (a) Example $(x, f(x))$ pairs and a consistent, linear hypothesis. (b) A consistent, degree-7 polynomial hypothesis for the same data set. (c) A different data set, which admits an exact degree-6 polynomial fit or an approximate linear fit. (d) A simple, exact sinusoidal fit to the same data set.

Ockham’s razor



- 14th-century English philosopher William of Ockham

Consistency vs. simplicity

- Usually algorithms prefer consistency by default
- Several ways to operationalize simplicity:
  - Reduce the Hypothesis Space
  - Regularization:
    - i.e., Cautious use of small counts

Expressiveness of Decision Trees

- How many distinct decision trees with n Boolean attributes??
  - number of Boolean functions
  - number of distinct truth tables with 2^n rows = 2^{2^n}
  - E.g., with 6 Boolean attributes, there are 18,446,744,073,709,551,616 trees
- How many purely conjunctive hypotheses (e.g., Hungry ∧ ¬Rain)??
  - Each attribute can be in (positive), in (negative), or out)
  - 3^n distinct conjunctive hypotheses
- More expressive hypothesis space
  - increases chance that target function can be expressed
  - increases number of hypotheses consistent w/ training set
    - may get worse predictions

Decision Tree

NICOLAS SPARKS	Hidden Treasure (h)	Candle Light Meal (clm)	Single Father (sf)	Explosions (e)	Crying (c)	Snakes (s)
Dear John	Yes	Yes	Yes	Yes	Yes	Yes
The Notebook	Yes	Yes	No	No	Yes	Yes
Best of Me	No	No	Yes	No	Yes	No
Safe Haven	No	Yes	Yes	Yes	Yes	Yes
Message in a Bottle	No	Yes	No	Yes	Yes	Yes

INDIANA JONES	Hidden Treasure (h)	Candle Light Meal (clm)	Single Father (sf)	Explosions (e)	Crying (c)	Snakes (s)
Indiana Jones and the Temple of Doom	Yes	Yes	Yes	Yes	No	Yes
Raiders of the Lost Ark	Yes	No	Yes	Yes	Yes	Yes
Kingdom of the Crystal Skull	Yes	No	Yes	Yes	Yes	No
Indiana Jones and the Last Crusade	Yes	No	Yes	No	No	Yes

Divide and Conquer

- Internal decision nodes
  - Univariate: Uses a single attribute, x_i
    - Numeric x_i : Binary split : $x_i > w_m$
    - Discrete x_i : n-way split for n possible values
  - Multivariate: Uses all attributes, $\boldsymbol{x}$
- Leaves
  - Classification: Class labels, or proportions
  - Regression: Numeric; r average, or local fit
- Learning is greedy; find the best split recursively (Breiman et al, 1984; Quinlan, 1986, 1993)

Decision Tree Learning

- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose “most significant" attribute as root of (sub)tree

function DECISION-TREE-LEARNING(*examples*, *attributes*, *parent_examples*) **returns** a tree

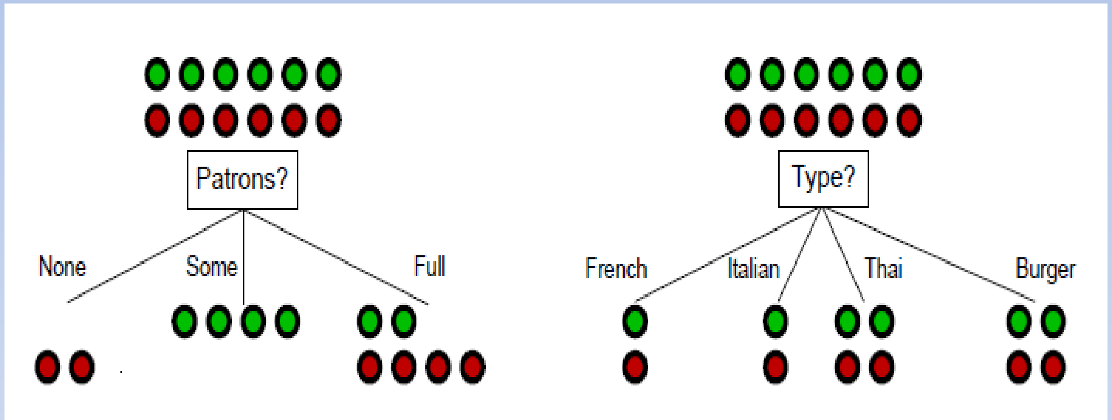
  if *examples* is empty **then return** PLURALITY-VALUE(*parent_examples*)
  else if all *examples* have the same classification **then return** the classification
  else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
  else
    $A \leftarrow \mathrm{argmax}_{a \in attributes}$ IMPORTANCE(a, *examples*)
    tree ← a new decision tree with root test A
    for each value v_k of A **do**
      $exs \leftarrow \{e : e \in examples$ **and** $e.A = v_k\}$
      subtree ← DECISION-TREE-LEARNING(*exs*, *attributes* − A, *examples*)
      add a branch to *tree* with label ($A = v_k$) and subtree *subtree*
    return *tree*

Figure 18.5 The decision-tree learning algorithm. The function IMPORTANCE is described in Section 18.3.4. The function PLURALITY-VALUE selects the most common output value among a set of examples, breaking ties randomly.

Importance(a, examples)

Choosing an Attribute w/ Importance(a, Examples)

- Idea: a good attribute splits the examples into subsets that are (ideally) “all positive" or “all negative“



- Patrons? is a better choice -- gives information about the classification
- So: we need a measure of how “good” a split is, even if the results aren’t perfectly separated out.

Information

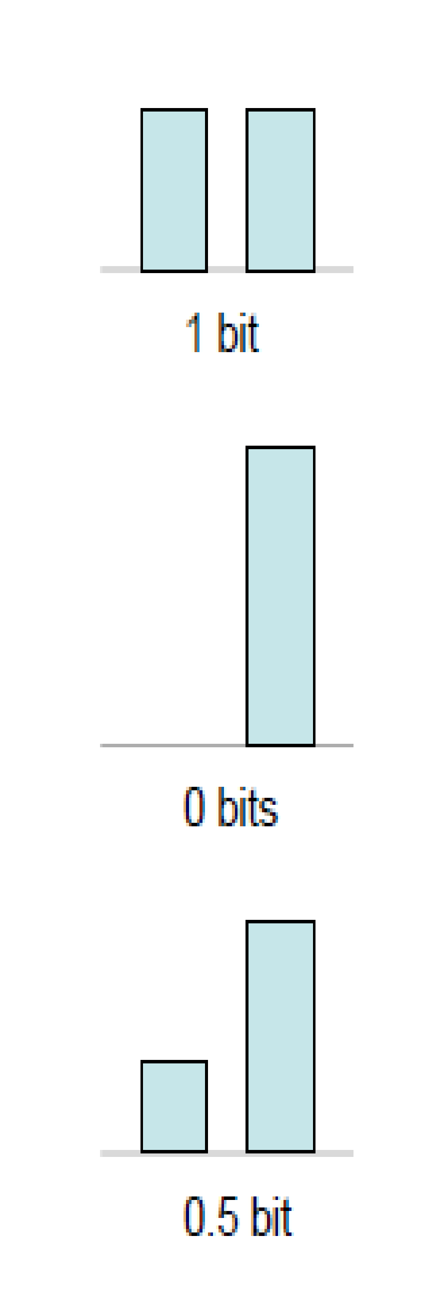
- Information answers questions
  - The more uncertain about the answer initially…
  - …the more information in the answer!
- Scale: Bits

  1 Bit = answer to Boolean question with prior <0.5, 0.5>
- Information (aka Entropy) in an answer when prior is $<P_1, \ldots, P_n>$ is

$$= \sum_{i=1}^{n} -P_i * Log_2(P_i)$$

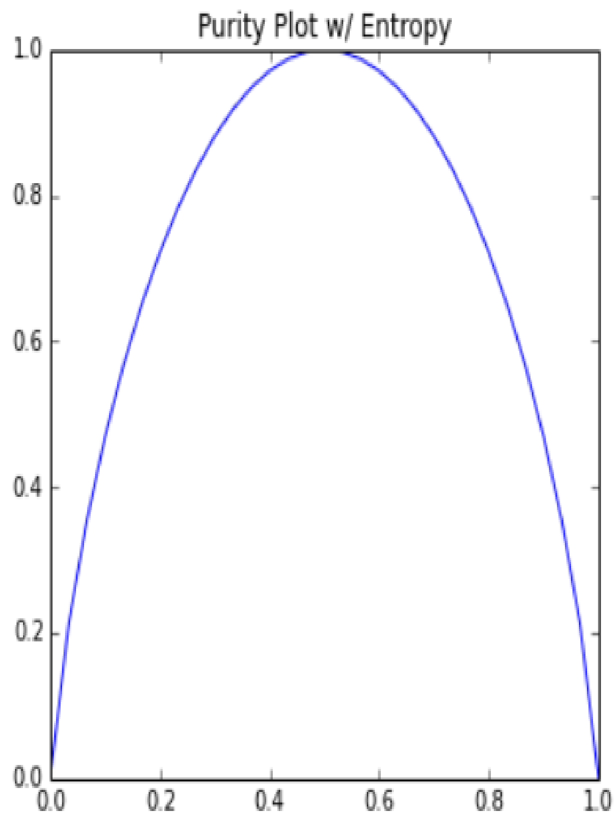
Entropy

- More Uniform = Higher Entropy
- More Values = Higher Entropy
- More Peaked = Lower Entropy
- Rare Values almost "don't count"



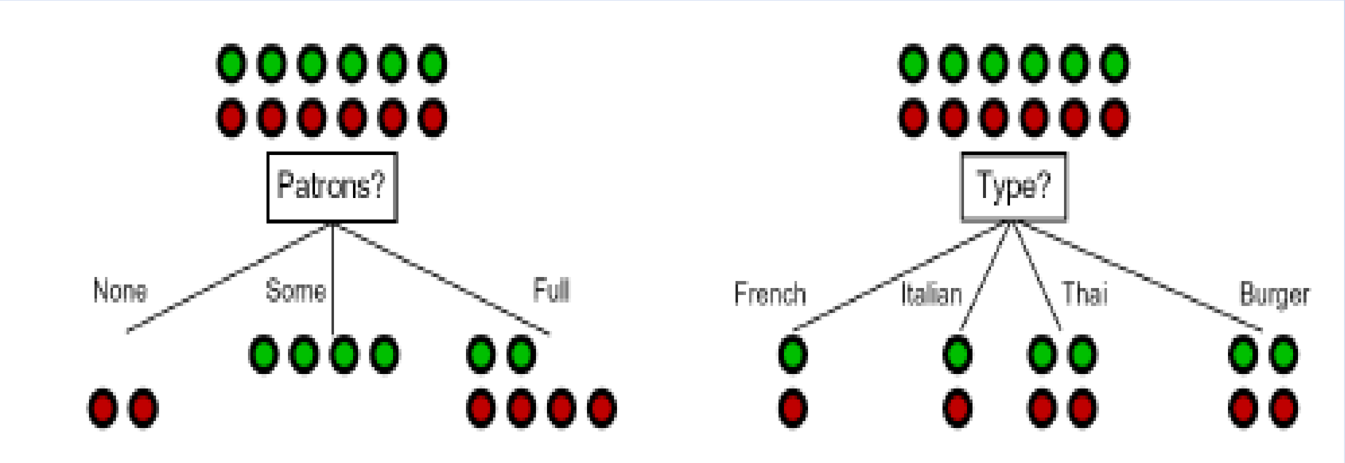
```
n = 30
samples = [(i, n-i) for i in range(n+1)]
x = [i/float(n) for (i,_) in samples]
```

```
yEntropy = [ExpectedPurity([l], Entropy) for l in samples]

plt.figure(figsize=(5, 5))
plt.plot(x, yEntropy)
plt.title("Purity Plot w/ Entropy")
plt.show()
```



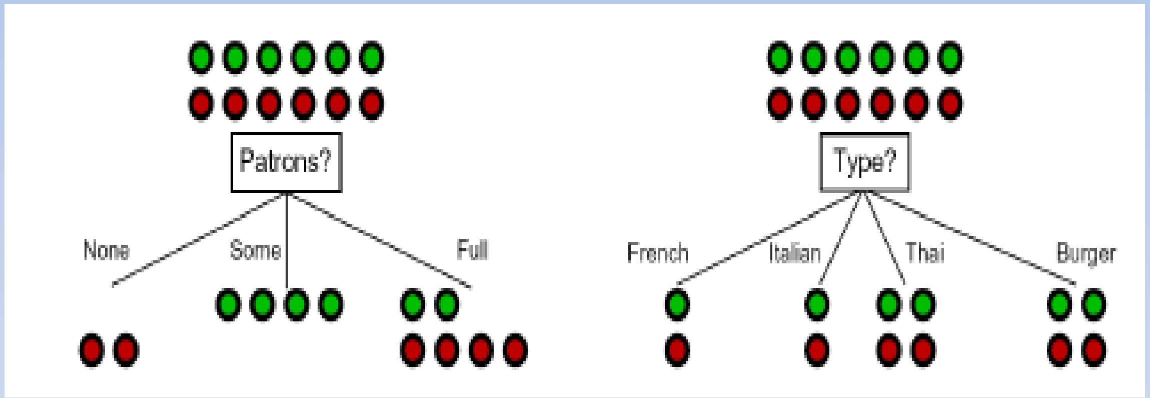
Entropy and Decision Trees

- How should we use Entropy to help choose Attributes.
- IDEA:
  - Check entropy before splitting on attribute.
  - Check entropy after splitting on attribute.
  - Information Gain is difference in Entropy before and after split.
- Problem: After Split, there are more than one distribution



Entropy and Decision Trees Patrons?

- Problem: After Split, there are more than one distribution
- Solution: Use Expected Entropy weighted by the number of examples:

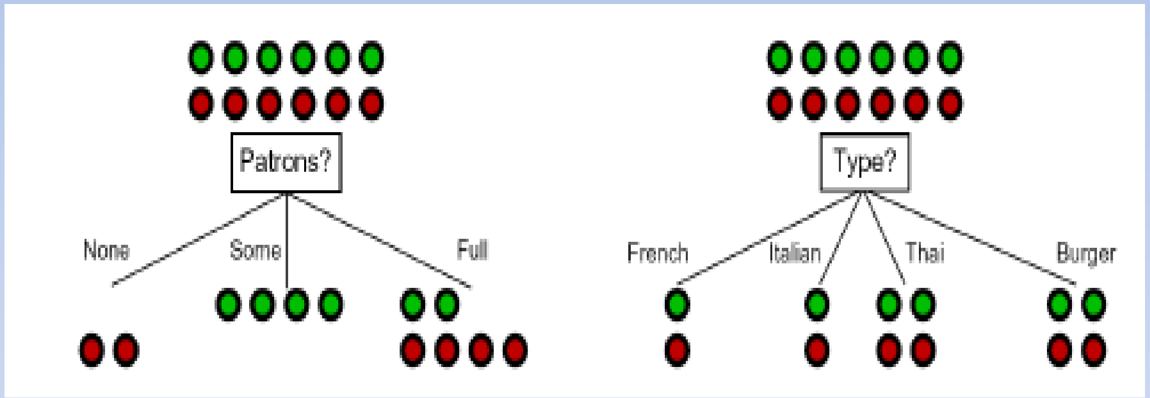


- Patrons?: Entropy=

  (2/12)*Entropy(<0, 1>) +
  4/12*Entropy(<1,0>)+(6/12)*Entropy(2/6,4/6)]
  ≈ 0.459

Entropy and Decision Trees Type?

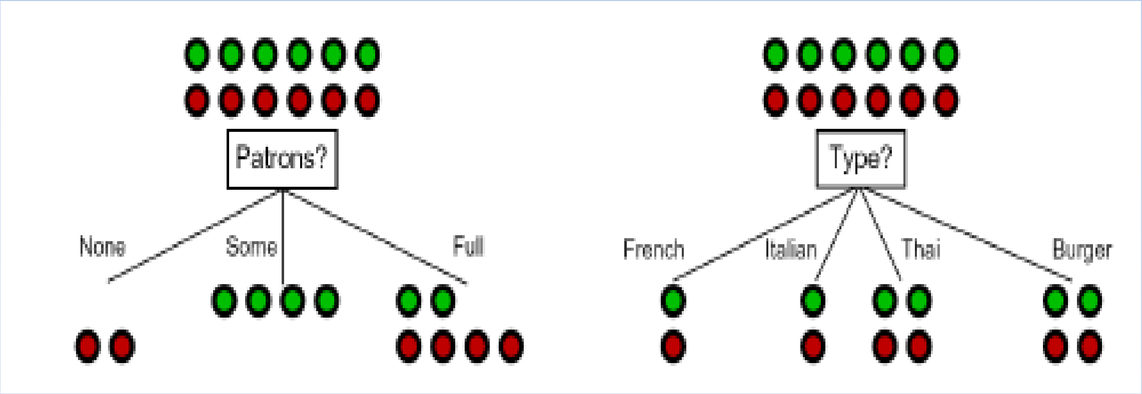
- Problem: After Split, there are more than one distribution
- Solution: Use Expected Entropy weighted by the number of examples:



- Type?: Entropy=
  (2/12)*Entropy(<0.5,0.5>) + 2/12*Entropy(<0.5,0.5>)+(4/12)*Entropy(0.5,0.5),
  (4/12)*Entropy(0.5,0.5)]
  = 1.0

Entropy and Decision Trees

- Problem: After Split, there are more than one distribution
- Solution: Use Expected Entropy weighted by the number of examples:



- Information Gain from Patron? Split ≈ 0.541
- Information Gain from Type? Split ≈ 0.0
- Patron? Is the Better Attribute to Choose!

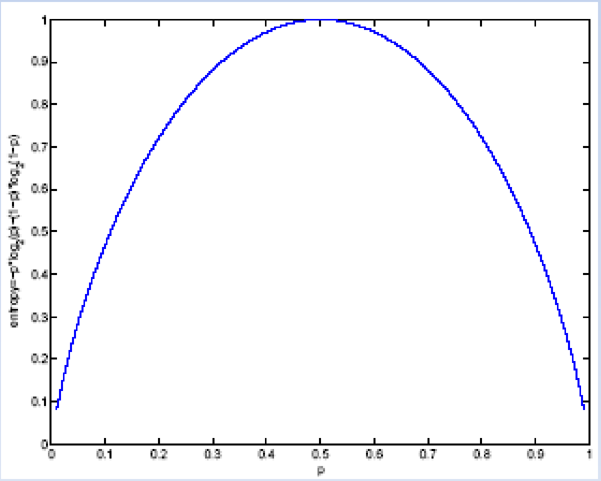
Classification Trees (ID3,CART,C4.5)

- For node *m*, N_m instances reach *m*, N^i_m belong to C_i

$$\hat{P}(C_i \mid \mathbf{x}, m) \equiv p^i_m = \frac{N^i_m}{N_m}$$

- Node *m* is pure if p^i_m is 0 or 1
- Measure of impurity is entropy

$$\mathcal{I}_m = -\sum_{i=1}^{K} p^i_m \log_2 p^i_m$$



Best Split

- If node m is pure, generate a leaf and stop, otherwise split and continue recursively
- Impurity after split: N_{mj} of N_m take branch j. N^i_{mj} belong to C_i

$$\hat{P}(C_i \mid \mathbf{x}, m, j) \equiv p^i_{mj} = \frac{N^i_{mj}}{N_{mj}} \qquad \mathcal{I}'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p^i_{mj} \log_2 p^i_{mj}$$

- Find the variable and split that min impurity (among all variables -- and split positions for numeric variables)

Purity Metrics w/ Python

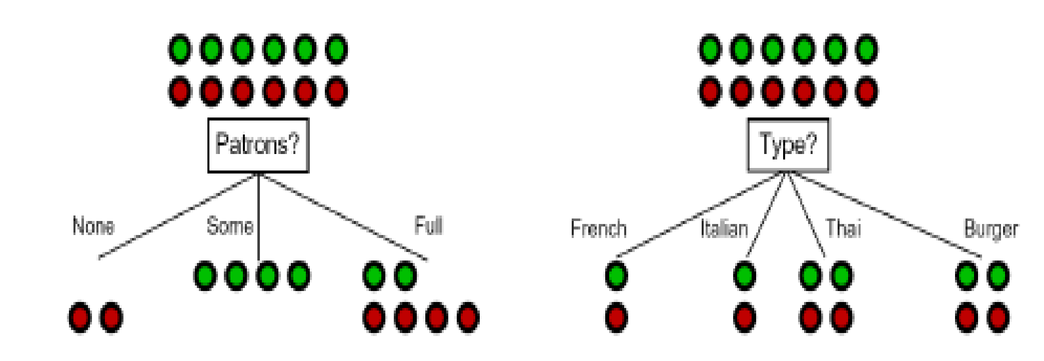
Purity Metrics

```
%matplotlib inline
```

```
import matplotlib.pyplot as plt
import math
```

```
def Entropy(L):
    outEntropy = 0.0
    for prob in L:
        if not prob:   # Normally undefined
            continue
        outEntropy += prob*math.log(prob,2)
    return -1*outEntropy

def Gini(L):
    outGini = 0.0
    for prob in L:
        outGini += prob*(1-prob)
    return outGini

def ExpectedPurity(L, PurityFN):
    outPurity = popSize = 0.0
    for l in L:
        samSize = sum(l)
        popSize += samSize
        outPurity += samSize * PurityFN([sample/float(samSize) for sample in l])
    return outPurity/popSize
```

Restaurant Example



Now Patrons Example

```
InitialSample = [(6, 6)]
PatronSplit = [(0,2),(4,0), (2,4)]
TypeSplit = [(1,1), (1,1), (2,2), (2,2)]
print "Entropy of Initial Sample: ", ExpectedPurity(InitialSample, Entropy)
print "Gini of Initial Sample", ExpectedPurity(InitialSample, Gini)
print "Purity w/ Patron (Entropy/Gini): ",
print ExpectedPurity(PatronSplit, Entropy),"/", ExpectedPurity(PatronSplit, Gini)
print "Information Gain w/ Patron Split (Entropy/Gini): ",
print ExpectedPurity(InitialSample, Entropy)-ExpectedPurity(PatronSplit, Entropy),"/",
print ExpectedPurity(InitialSample, Entropy)-ExpectedPurity(PatronSplit, Gini)
print
print "Purity w/ Type (Entropy/Gini): ",
print ExpectedPurity(TypeSplit, Entropy),"/", ExpectedPurity(TypeSplit, Gini)
print "Information Gain w/ Type Split (Entropy/Gini): ",
print ExpectedPurity(InitialSample, Entropy)-ExpectedPurity(TypeSplit, Entropy),"/",
print ExpectedPurity(InitialSample, Gini)-ExpectedPurity(TypeSplit, Gini)
```

```
Entropy of Initial Sample:  1.0
Gini of Initial Sample 0.5
Purity w/ Patron (Entropy/Gini):  0.459147917027 / 0.222222222222
Information Gain w/ Patron Split (Entropy/Gini):  0.540852082973 / 0.777777777778

Purity w/ Type (Entropy/Gini):  1.0 / 0.5
Information Gain w/ Type Split (Entropy/Gini):  0.0 / 0.0
```

Wishlist for a purity measure

- Properties we require from a purity measure:
  - When node is pure, measure should be zero
  - When impurity is maximal (i.e. all classes equally likely), measure should be maximal
  - Measure should obey *multistage property* (i.e. decisions can be made in several stages):

$$\text{measure}([2,3,4]) = \text{measure}([2,7]) + (7/9) \times \textit{measure}([3,4])$$

- Entropy is the only function that satisfies all three properties!

Alpaydin, CH 9

But entropy is not the only possible measure. For a two-class problem where $p^1 \equiv p$ and $p^2 = 1 - p$, $\phi(p, 1-p)$ is a nonnegative function measuring the impurity of a split if it satisfies the following properties (Devroye, Györfi, and Lugosi 1996):

- $\phi(1/2, 1/2) \geq \phi(p, 1-p)$, for any $p \in [0, 1]$.
- $\phi(0, 1) = \phi(1, 0) = 0$.
- $\phi(p, 1-p)$ is increasing in p on $[0, 1/2]$ and decreasing in p on $[1/2, 1]$.

Examples are

1. Entropy

$$\phi(p, 1-p) = -p \log_2 p - (1-p) \log_2 (1-p) \tag{9.4}$$

Equation 9.3 is the generalization to $K > 2$ classes.

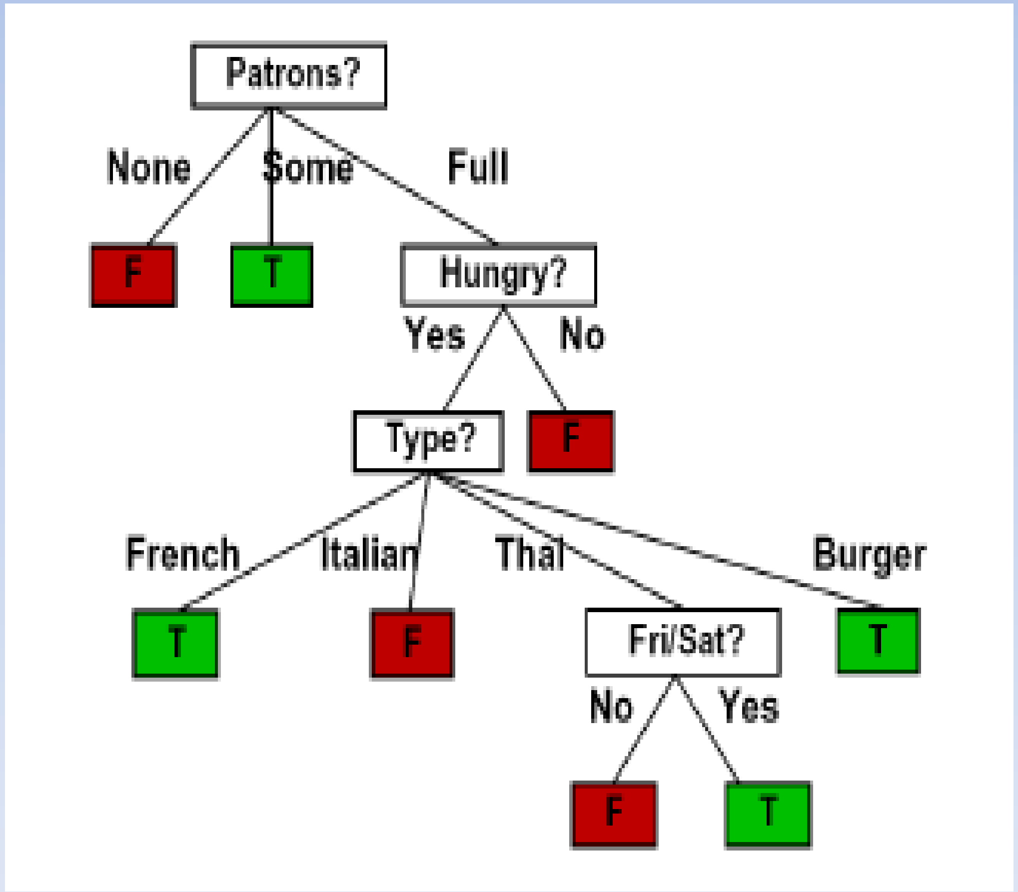
GINI INDEX

2. *Gini index* (Breiman et al. 1984)

$$\phi(p, 1-p) = 2p(1-p) \tag{9.5}$$

Decision Trees

- Decision Tree Learned



Gini versus Information Gain

Gini impurity [edit]

Not to be confused with Gini coefficient.

Used by the CART (classification and regression tree) algorithm, Gini impurity is a measure of how often a randomly chosen element from the set would be incorrectly labeled if it was randomly labeled according to the distribution of labels in the subset. Gini impurity can be computed by summing the probability f_i of an item with label i being chosen times the probability $1 - f_i$ of a mistake in categorizing that item. It reaches its minimum (zero) when all cases in the node fall into a single target category.

To compute Gini impurity for a set of items with J classes, suppose $i \in \{1, 2, \ldots, J\}$, and let f_i be the fraction of items labeled with class i in the set.

$$I_G(f) = \sum_{i=1}^{J} f_i(1 - f_i) = \sum_{i=1}^{J}(f_i - f_i{}^2) = \sum_{i=1}^{J} f_i - \sum_{i=1}^{J} f_i{}^2 = 1 - \sum_{i=1}^{J} f_i{}^2 = \sum_{i \neq k} f_i f_k$$

Information gain [edit]

Main article: Information gain in decision trees

Used by the ID3, C4.5 and C5.0 tree-generation algorithms. Information gain is based on the concept of entropy from information theory.

Entropy is defined as below

$$I_E(f) = -\sum_{i=1}^{J} f_i \log_2 f_i$$

Information Gain = Entropy(parent) - Weighted Sum of Entropy(Children)

$$IG(T, a) = H(T) - H(T|a)$$

- Alpaydin, PseudoCode

```
GenerateTree(𝒳)
    If NodeEntropy(𝒳)< θ_I /* eq. 9.3
        Create leaf labelled by majority class in 𝒳
        Return
    i ← SplitAttribute(𝒳)
    For each branch of x_i
        Find 𝒳_i falling in branch
        GenerateTree(𝒳_i)
SplitAttribute(𝒳)
    MinEnt← MAX
    For all attributes i = 1, . . . , d
        If x_i is discrete with n values
            Split 𝒳 into 𝒳_1, . . . , 𝒳_n by x_i
            e ← SplitEntropy(𝒳_1, . . . , 𝒳_n) /* eq. 9.8 */
            If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
            For all possible splits
                Split 𝒳 into 𝒳_1, 𝒳_2 on x_i
                e←SplitEntropy(𝒳_1, 𝒳_2)
                If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

Overfitting & Regularization

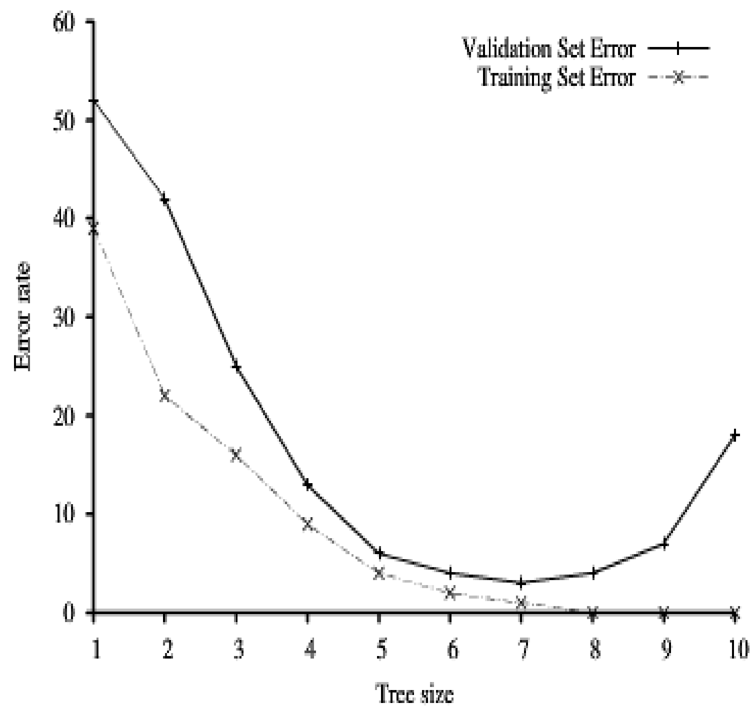
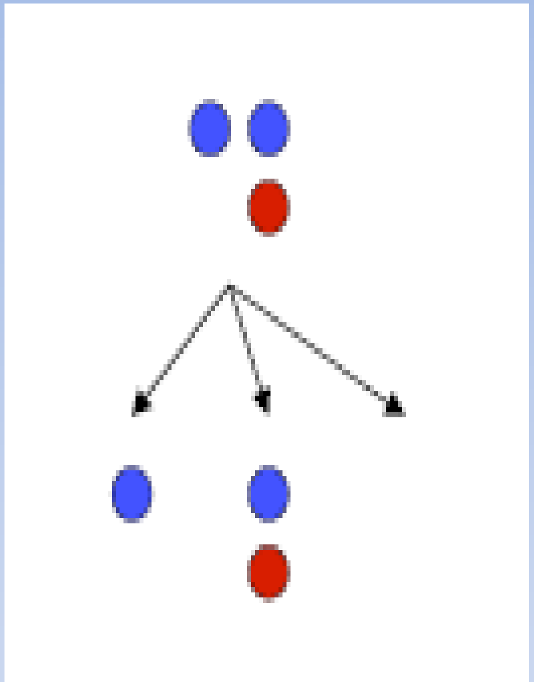


Figure 18.9 Error rates on training data (lower, dashed line) and validation data (upper, solid line) for different size decision trees. We stop when the training set error rate asymptotes, and then choose the tree with minimal error on the validation set; in this case the tree of size 7 nodes.

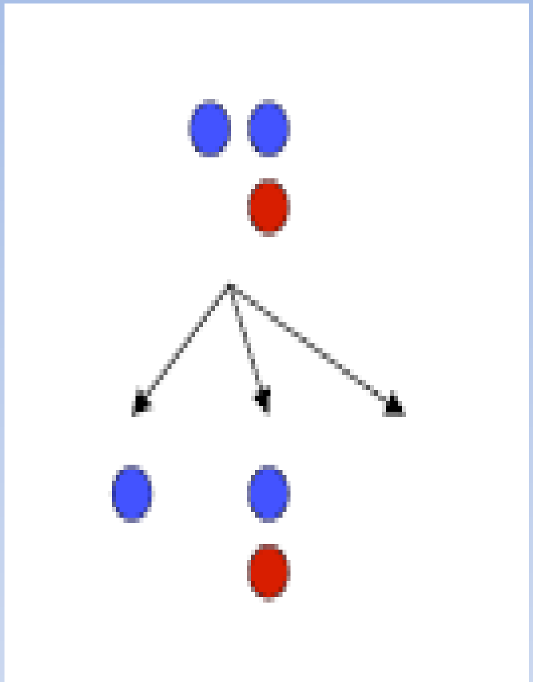
Chi-Squared Pruning

- What do we expect from a 3-way split?
- Probably shouldn't split if counts are so small they could be due to chance.
- A chi-squared test can tell us how likely it is that deviations from a perfect split are due to chance.



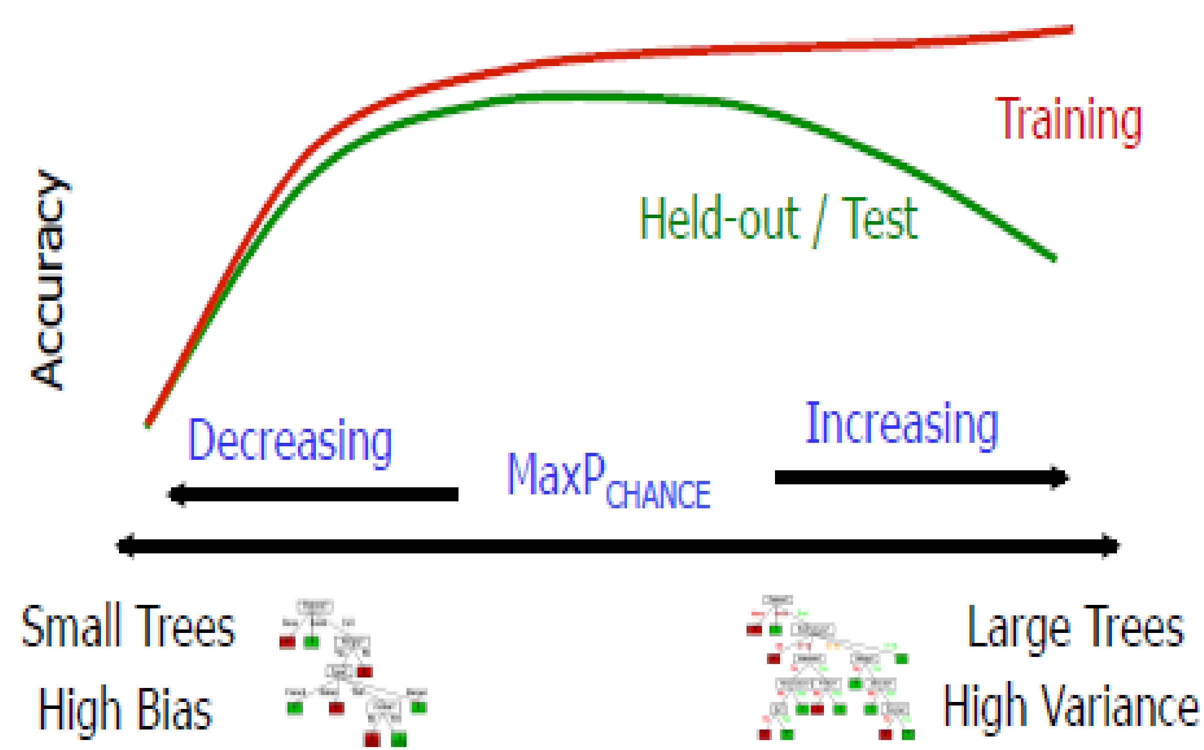
Chi-Squared Pruning

- Build full decision tree
- Begin at bottom of tree
- Delete splits in where the probability that the information gain is due to chance is larger than some cuttoff.
- Continue upward till no more prunable nodes.



Regularization w/ Chi-Squared

- $\text{MaxP}_{\text{CHANCE}}$ is a regularization parameter
- Generally, set it using held-out data (as usual)



Pruning Trees

- Remove subtrees for better generalization (decrease variance)
  - Prepruning: Early stopping
  - Postpruning: Grow the whole tree then prune subtrees that overfit on the pruning set
- Prepruning is faster, postpruning is more accurate (requires a separate pruning set)

jupyter DecisionTree Last Checkpoint: 5 minutes ago (autosaved)

File Edit View Insert Cell Kernel Help

Code CellToolbar

In [1]:
```
# To support both python 2 and python 3
from __future__ import division, print_function, unicode_literals

# Common imports
import numpy as np

# to make this notebook's output stable across runs
np.random.seed(42)

# To plot pretty figures
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt

import pandas as pd

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

In [2]:
```
# Load data
iris = load_iris()
print( iris.data.shape )

df = pd.DataFrame(iris['data'], columns=iris['feature_names'])
df['target'] = iris['target']

print(df.describe())
print(df.head(4))
```

```
(150, 4)
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.054000           3.758667
std             0.828066          0.433594           1.764420
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.198667    1.000000
std            0.763161    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

In [3]:
```
from sklearn.model_selection import train_test_split

train_set, test_set = train_test_split(df, test_size=0.2, random_state=42)

print(train_set.describe())
print( "\nTraining Percentages:\n" )
print(train_set.count()/df.count())
```

```
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         120.000000        120.000000         120.000000
mean            5.809167          3.057500           3.727500
std             0.823805          0.446398           1.751252
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.500000
50%             5.750000          3.000000           4.250000
75%             6.400000          3.325000           5.100000
max             7.700000          4.400000           6.700000

       petal width (cm)      target
count        120.000000  120.000000
mean           1.182500    0.991667
std            0.753442    0.814736
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000

Training Percentages:

sepal length (cm)    0.8
sepal width (cm)     0.8
petal length (cm)    0.8
petal width (cm)     0.8
target               0.8
dtype: float64
```

In [4]:
```
X = train_set.drop("target", axis=1) # drop labels for training set
Y = train_set['target'].copy()

classifier = DecisionTreeClassifier(max_depth=3)
classifier.fit(X,Y)
y_pred = classifier.predict(X)
print ("Accuracy on Training: ",sum(y_pred==Y)/len(Y))

y_pred = classifier.predict(X)
print ("Accuracy on Training: ",sum(y_pred==Y)/len(Y))
```

```
Accuracy on Training:  0.958333333333
Accuracy on Training:  0.958333333333
```

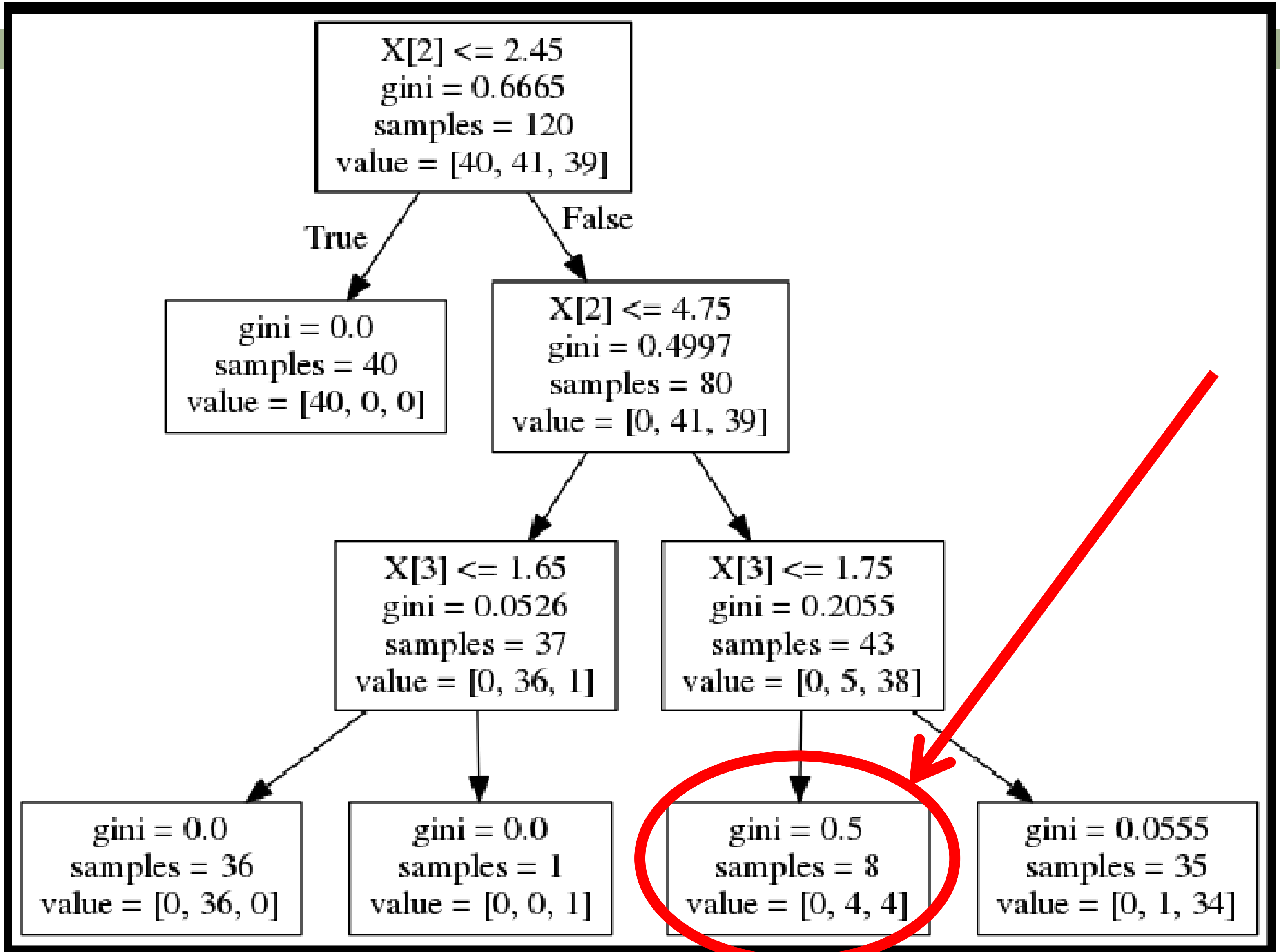
In [5]:
```
X = test_set.drop("target", axis=1) # drop labels for training set
Y = test_set['target'].copy()

y_pred = classifier.predict(X)
print ("Accuracy on Test Set: ",sum(y_pred==Y)/len(Y))
```

```
Accuracy on Test Set:  1.0
```

In [10]:
```
import StringIO
from sklearn import tree
#from graphviz import *

import matplotlib.image as mpimg
import pydotplus

dotfile = open("dtree2.dot", 'w')
tree.export_graphviz(classifier, out_file = dotfile)
dotfile.close()
dot_data = StringIO.StringIO()
tree.export_graphviz(classifier, out_file=dot_data)
graph = pydotplus.graph_from_dot_file("dtree2.dot")
graph.write_png('test.png')
img=mpimg.imread('test.png')
fig = plt.figure(figsize=(10, 10))
plt.axis("off")
plt.imshow(img, cmap = plt.cm.binary,
           interpolation="nearest")
plt.show()
```

```
Fontconfig warning: ignoring C.UTF-8: not a valid language tag
```

Iris Decision Tree



45

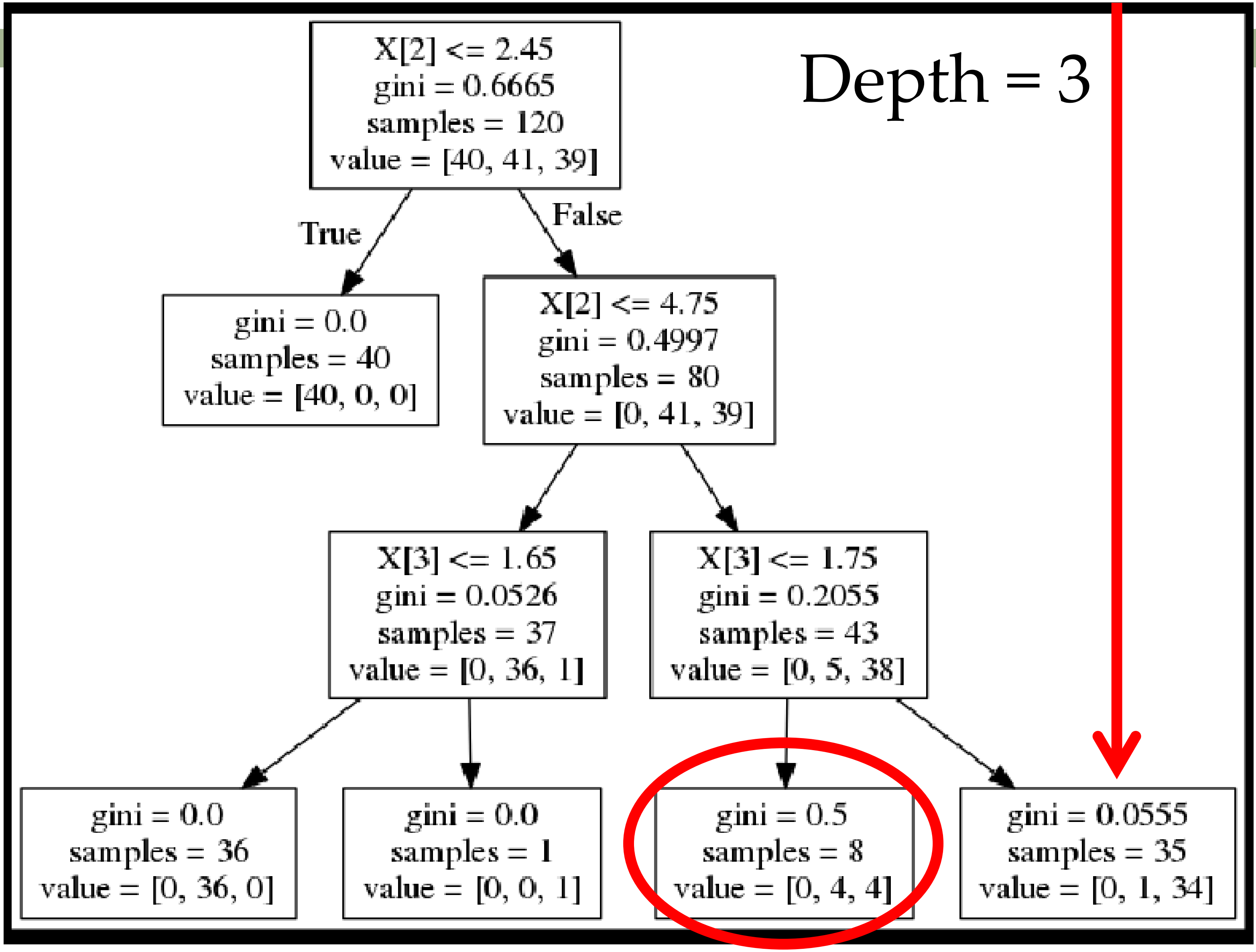
In [4]:
```
X = train_set.drop("target", axis=1) # drop labels for training set
Y = train_set['target'].copy()

classifier = DecisionTreeClassifier(max_depth=3)
classifier.fit(X,Y)
y_pred = classifier.predict(X)
print ("Accuracy on Training: ",sum(y_pred==Y)/len(Y))

y_pred = classifier.predict(X)
print ("Accuracy on Training: ",sum(y_pred==Y)/len(Y))
```

```
Accuracy on Training:  0.958333333333
Accuracy on Training:  0.958333333333
```

In [5]:
```
X = test_set.drop("target", axis=1) # drop labels for training set
Y = test_set['target'].copy()

y_pred = classifier.predict(X)
print ("Accuracy on Test Set: ",sum(y_pred==Y)/len(Y))
```

```
Accuracy on Test Set:  1.0
```

Iris Decision Tree



Question 18.6

18.6 Consider the following data set comprised of three binary input attributes (A_1, A_2, and A_3) and one binary output:

Example	A_1	A_2	A_3	Output y
$\mathbf{x}_1$	1	0	0	0
$\mathbf{x}_2$	1	0	1	0
$\mathbf{x}_3$	0	1	0	0
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	1	1	0	1

Use the algorithm in Figure 18.5 (page 702) to learn a decision tree for these data. Show the computations made to determine the attribute to split at each node.

Question 18.6

18.6 Consider the following data set comprised of three binary input attributes (A_1, A_2, and A_3) and one binary output:

Example	A_1	A_2	A_3	Output y
$\mathbf{x}_1$	1	0	0	0
$\mathbf{x}_2$	1	0	1	0
$\mathbf{x}_3$	0	1	0	0
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	1	1	0	1

Use the algorithm in Figure 18.5 (page 702) to learn a decision tree for these data. Show the computations made to determine the attribute to split at each node.

- InitialSet = [(3, 2)]
- A1 = [(1, 0), (2, 2)]
- A2 = [(2, 0), (1, 2)]
- A3 = [(1, 1), (2, 1)]

18.6 Consider the following (
A_3) and one binary output:

Example	A_1	A_2	A_3	(
$\mathbf{x}_1$	1	0	0	
$\mathbf{x}_2$	1	0	1	
$\mathbf{x}_3$	0	1	0	
$\mathbf{x}_4$	1	1	1	
$\mathbf{x}_5$	1	1	0	

Use the algorithm in Figure 18
computations made to determi

```
def test2():
    ' Run 18.6 Example'
    InitialSet = [(3.0,2.0)]
    A1 = [(1.0,0.0), (2.0, 2.0)]
    A2 = [(2.0,0.0), (1.0, 2.0)]
    A3 = [(1.0,1.0), (2.0, 1.0)]
    print 'Entropy of Initial Sample ', expectedEntropy(InitialSet)
    print 'Entropy after A1 Split ', expectedEntropy(A1)
    print 'Entropy after A2 Split ', expectedEntropy(A2)
    print 'Entropy after A3 Split ', expectedEntropy(A3)

    print ('Information Gain from A1 Split ',
        expectedEntropy(InitialSet)-expectedEntropy(A1))
    print ('Information Gain from A2 Split ',
        expectedEntropy(InitialSet)-expectedEntropy(A2))
    print ('Information Gain from A3 Split ',
        expectedEntropy(InitialSet)-expectedEntropy(A3))
```

```
>>> test2()
Entropy of Initial Sample  0.970950594455
Entropy after A1 Split  0.8
Entropy after A2 Split  0.550977500433
Entropy after A3 Split  0.950977500433
('Information Gain from A1 Split ', 0.17095059445466854)
('Information Gain from A2 Split ', 0.4199730940219748)
('Information Gain from A3 Split ', 0.01997309402197489)
```

- InitialSet = [(3,
- A1 = [(1, 0), (2,
- A2 = [(2, 0), (1,
- A3 = [(1, 1), (2, 1)]

Question 18.6 w/ A2 Split

18.6 Consider the following data set comprised of three binary input attributes (A_1, A_2, and A_3) and one binary output:

Example	A_1	A_2	A_3	Output y
$\mathbf{x}_3$	0	1	0	0
$\mathbf{x}_4$	1	1	1	1
$\mathbf{x}_5$	1	1	0	1

Use the algorithm in Figure 18.5 (page 702) to learn a decision tree for these data. Show the computations made to determine the attribute to split at each node.

- InitialSet = [(1, 2)]
- A1 = [(1, 0), (0, 2)]
- A3 = [(1, 1), (0, 1)]

Question 18.6 w/ A2 Split

18.6 Consider the following d
A_3) and one binary output:

Example	A_1	A_2	A_3	C
$\mathbf{x}_3$	0	1	0	
$\mathbf{x}_4$	1	1	1	
$\mathbf{x}_5$	1	1	0	

Use the algorithm in Figure 18.
computations made to determin

```
def test3():
    ' Run 18.6 Example (Part 2)'
    InitialSet = [(1.0,2.0)]
    A1 = [(1.0,0.0), (0.0, 2.0)]
    A3 = [(1.0,1.0), (0.0, 1.0)]
    print 'Entropy of Initial Sample ', expectedEntropy(InitialSet)
    print 'Entropy after A1 Split ', expectedEntropy(A1)
    print 'Entropy after A3 Split ', expectedEntropy(A3)

    print ('Information Gain from A1 Split ',
        expectedEntropy(InitialSet)-expectedEntropy(A1))
    print ('Information Gain from A3 Split ',
        expectedEntropy(InitialSet)-expectedEntropy(A3))
```

- InitialSet = [(1, 2)]
- A1 = [(1, 0), (0, 2)]
- A3 = [(1, 1), (0, 1)]

```
>>> test3()
Entropy of Initial Sample  0.918295834054
Entropy after A1 Split  0.0
Entropy after A3 Split  0.666666666667
('Information Gain from A1 Split ', 0.9182958340544896)
('Information Gain from A3 Split ', 0.2516291673878229)
```

18.6 Co

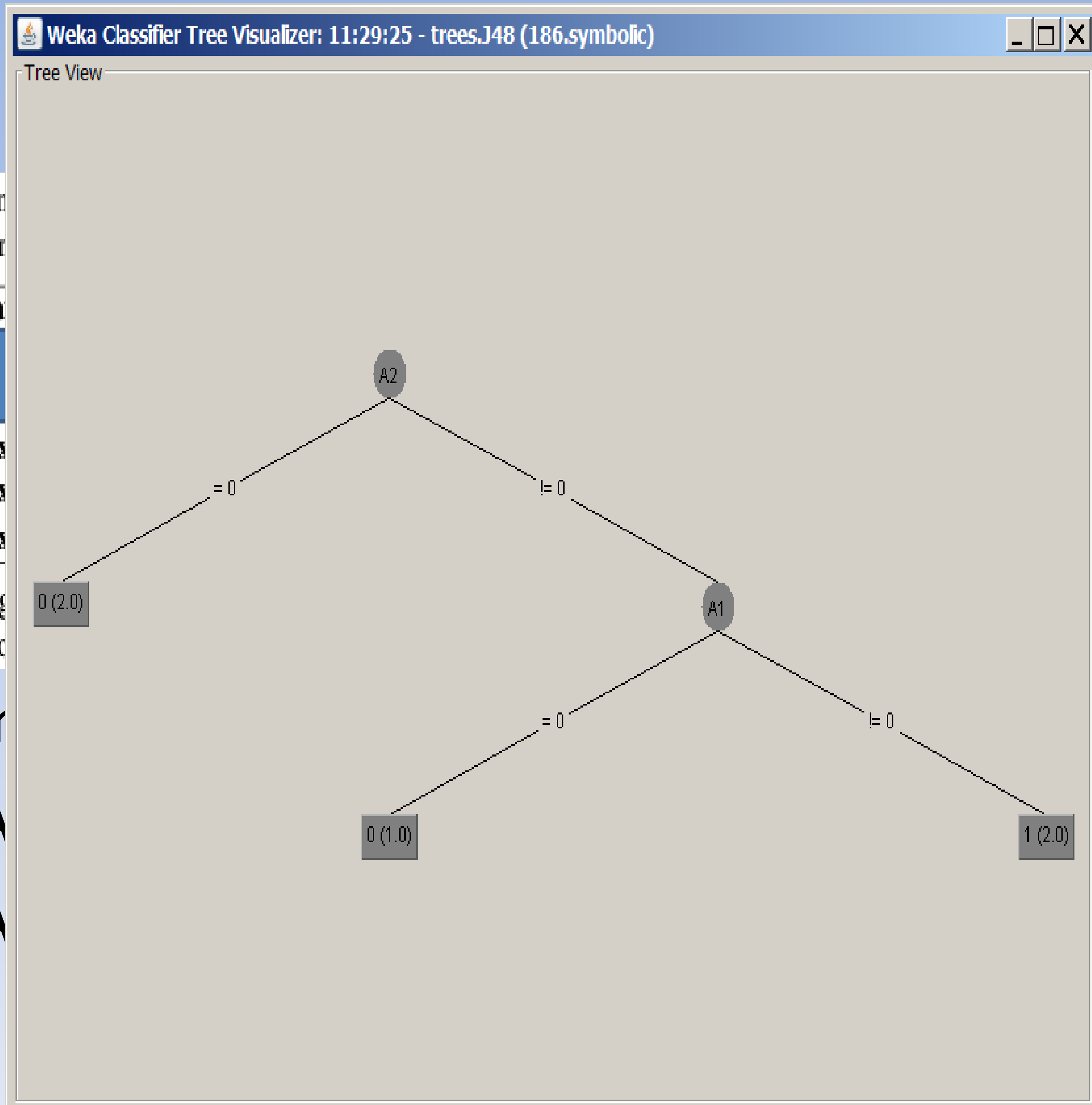
A_3) and o

Exa

Use the al

computatio

- In
- A
- A



tropy(InitialSet)
py(A1)
py(A3)

(A1))

(A3))

054

9182958340544896)
2516291673878229)

e18.6.arff - Wor...

Home View

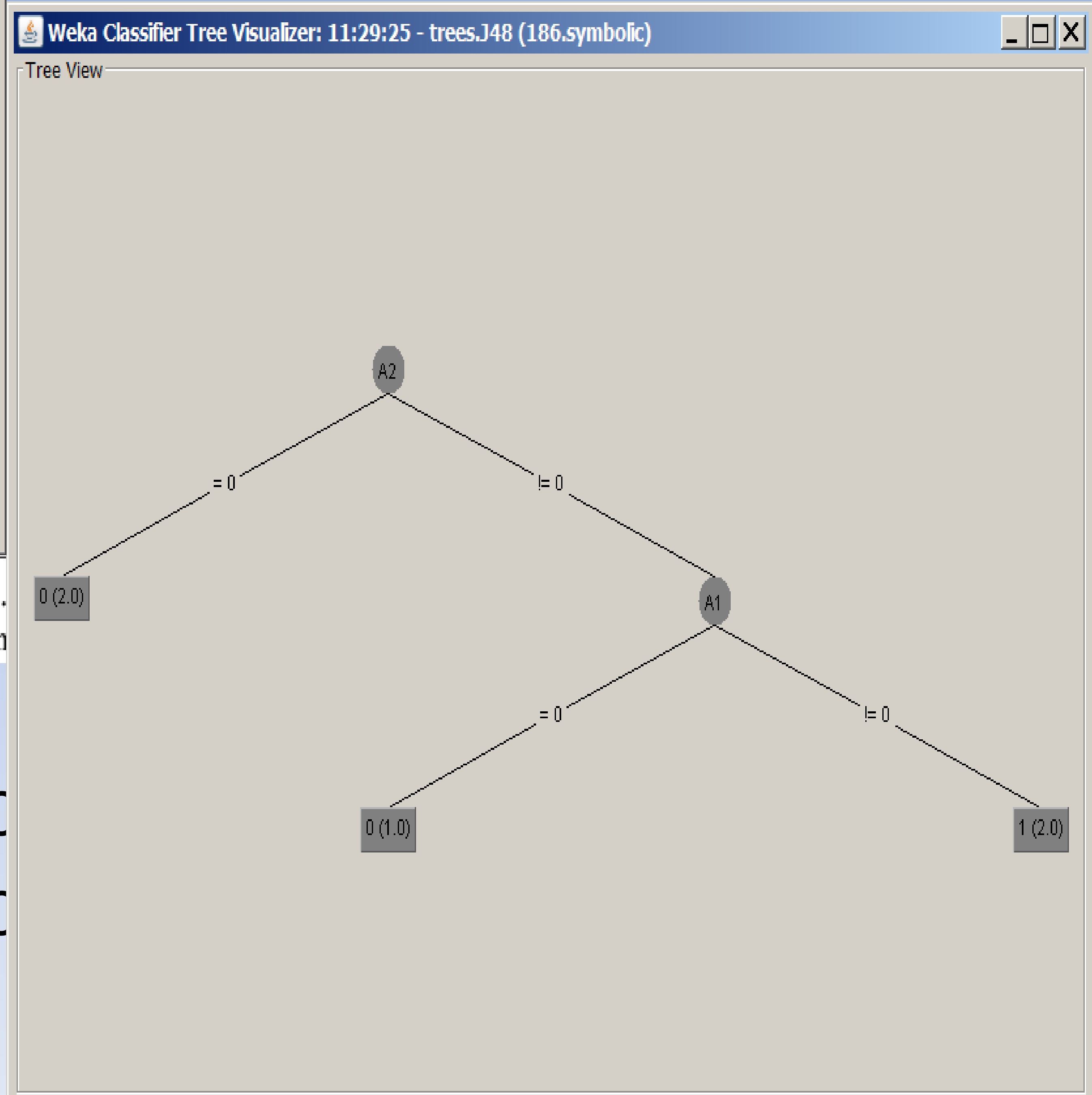
```
@relation 186.symbolic

@attribute A1 {0,1}
@attribute A2 {0,1}
@attribute A3 {0,1}
@attribute Y {0, 1}

@data
1,  0,  0,  0
1,  0,  1,  0
0,  1,  0,  0
1,  1,  1,  1
1,  1,  0,  1
```

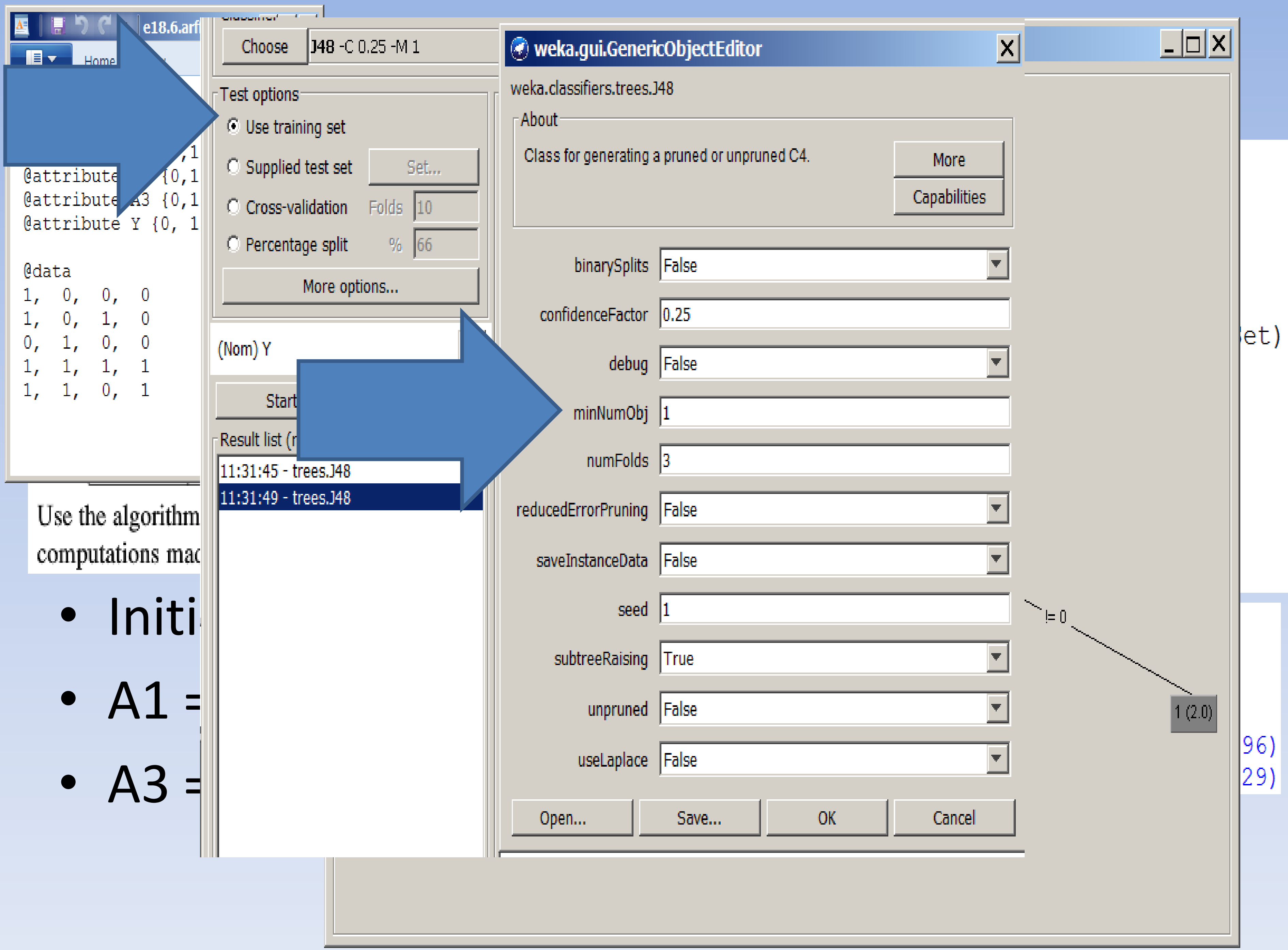
Use the algorithm in Figure 18.
computations made to determin

- InitialSet
- A1 = [(1.C
- A3 = [(1.C

et)

96)
29)





e18.6.arff
Home
@attribute
@attribute A3 {0,1
@attribute Y {0, 1
@data
1, 0, 0, 0
1, 0, 1, 0
0, 1, 0, 0
1, 1, 1, 1
1, 1, 0, 1
Use the algorithm
computations mac
• Initi
• A1 =
• A3 =
Choose
J48 -C 0.25 -M 1
Test options
Use training set
Supplied test set
Set...
Cross-validation
Folds
10
Percentage split
%
66
More options...
(Nom) Y
Start
Result list (r
11:31:45 - trees.J48
11:31:49 - trees.J48
weka.gui.GenericObjectEditor
weka.classifiers.trees.J48
About
Class for generating a pruned or unpruned C4.
More
Capabilities
binarySplits False
confidenceFactor 0.25
debug False
minNumObj 1
numFolds 3
reducedErrorPruning False
saveInstanceData False
seed 1
subtreeRaising True
unpruned False
useLaplace False
Open...
Save...
OK
Cancel
!= 0
1 (2.0)
et)
96)
29)

Regression Trees

- Error at node *m*:

$$b_m(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_m : \mathbf{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

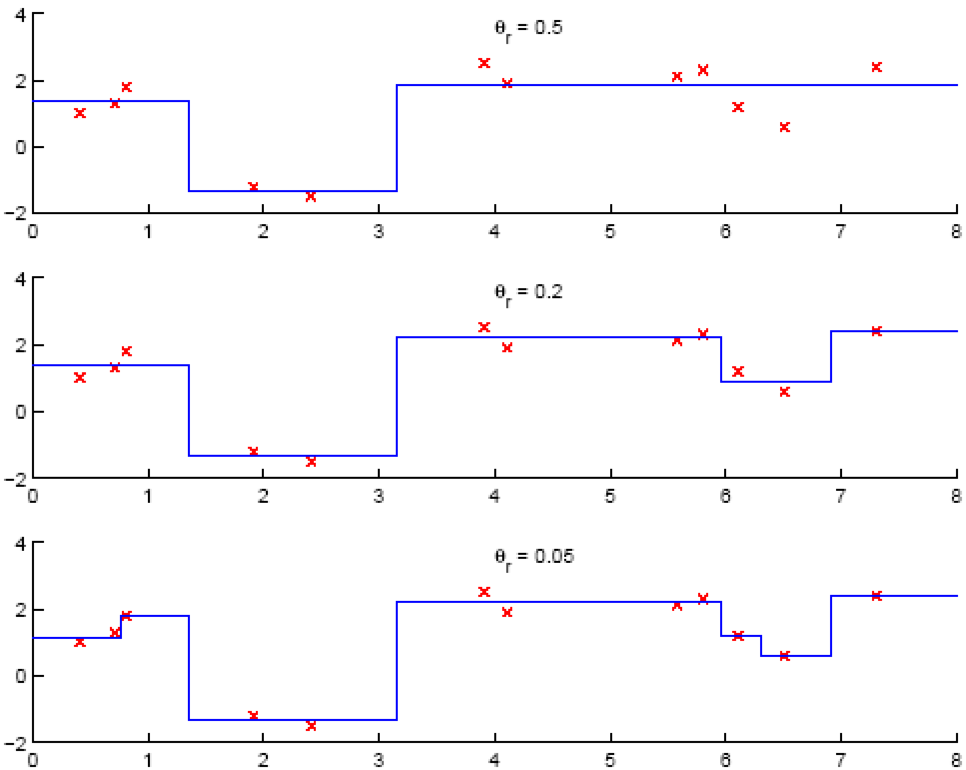
$$E_m = \frac{1}{N_m} \sum_t \left(r^t - g_m\right)^2 b_m\left(\mathbf{x}^t\right) \qquad g_m = \frac{\sum_t b_m\left(\mathbf{x}^t\right) r^t}{\sum_t b_m\left(\mathbf{x}^t\right)}$$

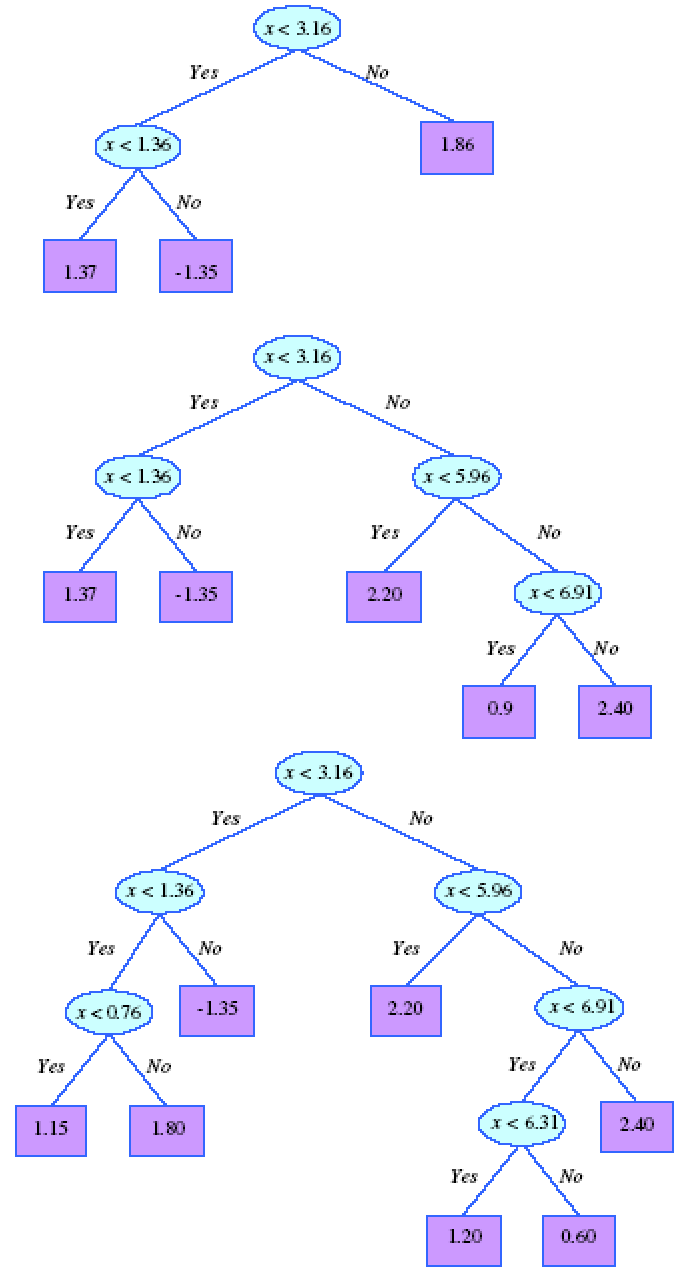
- After splitting:

$$b_{mj}(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x} \in \mathcal{X}_{mj} : \mathbf{x} \text{ reaches node } m \text{ and branch } j \\ 0 & \text{otherwise} \end{cases}$$

$$E'_m = \frac{1}{N_m} \sum_j \sum_t \left(r^t - g_{mj}\right)^2 b_{mj}\left(\mathbf{x}^t\right) \qquad g_{mj} = \frac{\sum_t b_{mj}\left(\mathbf{x}^t\right) r^t}{\sum_t b_{mj}\left(\mathbf{x}^t\right)}$$

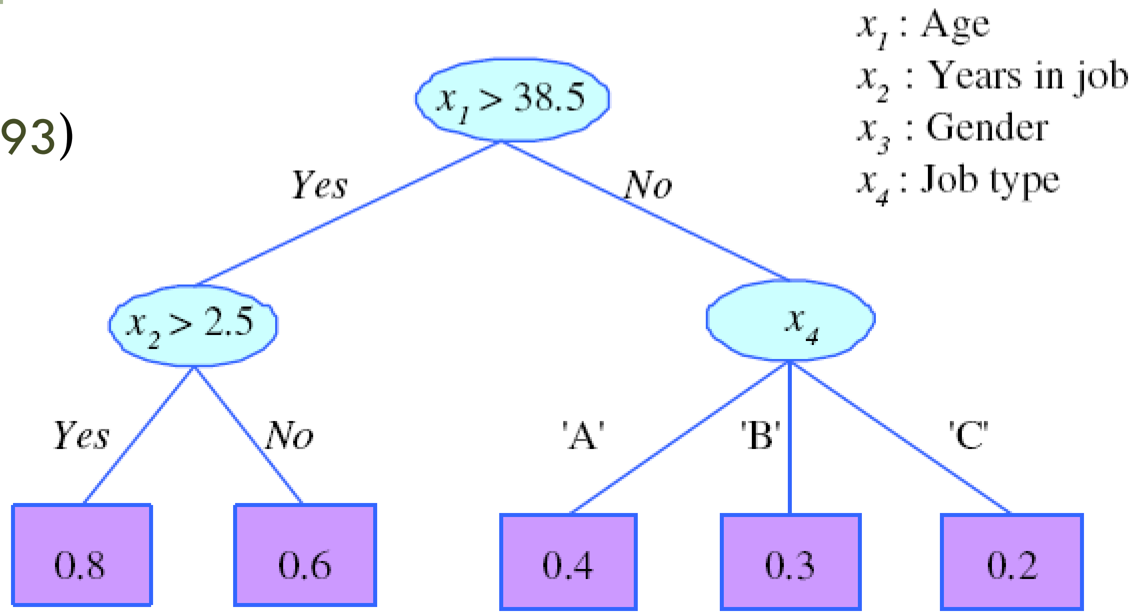
Model Selection in Trees





Rule Extraction from Trees

C4.5Rules
(Quinlan, 1993)



R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

Learning Rules

- Rule induction is similar to tree induction but
  - tree induction is breadth-first,
  - rule induction is depth-first; one rule at a time
- Rule set contains rules; rules are conjunctions of terms
- Rule covers an example if all terms of the rule evaluate to true for the example
- Sequential covering: Generate rules one at a time until all positive examples are covered
- IREP (Fürnkrantz and Widmer, 1994), Ripper (Cohen, 1995)

```
Ripper(Pos,Neg,k)
  RuleSet ← LearnRuleSet(Pos,Neg)
  For k times
    RuleSet ← OptimizeRuleSet(RuleSet,Pos,Neg)
LearnRuleSet(Pos,Neg)
  RuleSet ← ∅
  DL ← DescLen(RuleSet,Pos,Neg)
  Repeat
    Rule ← LearnRule(Pos,Neg)
    Add Rule to RuleSet
    DL' ← DescLen(RuleSet,Pos,Neg)
    If DL'>DL+64
      PruneRuleSet(RuleSet,Pos,Neg)
      Return RuleSet
    If DL'<DL DL ← DL'
      Delete instances covered from Pos and Neg
  Until Pos = ∅
  Return RuleSet
```

```
PruneRuleSet(RuleSet,Pos,Neg)
  For each Rule ∈ RuleSet in reverse order
    DL ← DescLen(RuleSet,Pos,Neg)
    DL' ← DescLen(RuleSet-Rule,Pos,Neg)
    IF DL'<DL Delete Rule from RuleSet
  Return RuleSet
OptimizeRuleSet(RuleSet,Pos,Neg)
  For each Rule ∈ RuleSet
      DL0 ← DescLen(RuleSet,Pos,Neg)
      DL1 ← DescLen(RuleSet-Rule+
        ReplaceRule(RuleSet,Pos,Neg),Pos,Neg)
      DL2 ← DescLen(RuleSet-Rule+
        ReviseRule(RuleSet,Rule,Pos,Neg),Pos,Neg)
      If DL1=min(DL0,DL1,DL2)
        Delete Rule from RuleSet and
            add ReplaceRule(RuleSet,Pos,Neg)
      Else If DL2=min(DL0,DL1,DL2)
        Delete Rule from RuleSet and
            add ReviseRule(RuleSet,Rule,Pos,Neg)
  Return RuleSet
```

William Cohen

William W. Cohen

Professor, Machine Learning Department and Language Technology Institute, Carnegie Mellon University

Director of the Undergraduate Minor in Machine Learning and Co-Director of the Master of Science in Machine Learning program

[Bio | Announcements and FAQs | Teaching | Projects | Publications (recent, all) | Software | Datasets | Talks | Students & Colleagues | Blog | Contact Info | Other Stuff]

Prospective visitors/students: see announcements

- SLIPPER is an old old rule-learning system Yoram Singer and I developed. This code is provided with absolutely no warranty, promise of support, or really, any expectation that it will keep working. You are totally on your own with this one, friend.
- WHIRL is another old system I wrote. Currently, I am not distributing it, but ask me if you're interested in reviving the source code.
- To get a copy of RIPPER, please send mail to my evil twin brother, wcohen -AT- gmail.com. As an alternative to that ancient code: I haven't used it myself, but I've heard good things about J-RIP, a Ripper clone written for WEKA.

Datasets

The following datasets are available for anyone to use for research purposes:

- Zhilin Yang is distributing the data from our ACL-2017 paper on semi-supervised QA.

Announcements and FAQs

- **I'm moving to Google.** After the spring 2018 semester ends, I will move from CMU to Google. I will be leading a new research group in AI/ML that will be located in Pittsburgh in Google's Bakery Square location. And case you're wondering - yes, we will be hiring!

- **Can I visit CMU and work with you?, or can I apply to CMU and work with you as a grad student?** I will continue to advise my current students as needed through May 2019, but I will not be taking any new students or hosting nay visitors.

- **What's the difference between 10-601 and ...?** If you're having trouble with the MLD's growing menu of intro ML courses here's a draft of a document that explains the differences. If you're not sure if you're qualified, the prereqs for the course are listed on the course home page, and we're fairly strict about enforcing them for undergrads. Grad students should have equivalent experience: good programming skills - the equivalent of a one-semester college course - and some mathematical maturity, including prior exposure to calculus, probability and linear algebra. If you're not sure about your background there is a self-assessment test you can take.

Multivariate Trees