

ECE5984 – Applications of Machine Learning

Lecture 9 – Decision Trees

Creed Jones, PhD

Course updates

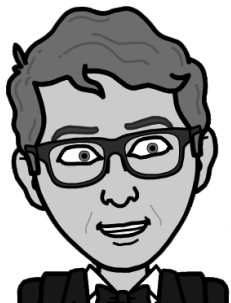
- HW2 due TODAY
 - Be sure and submit by 11:59 PM
 - Don't wait until the last minute – Canvas is often busy just before the deadline
- Quiz 3
 - next Thursday, February 24

Today's Objectives

Decision Tree models

- Concept
- Entropy
- Information Gain
- The ID3 algorithm for constructing efficient trees
- Examples
- Tree learning in Python

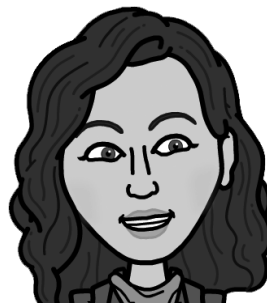
- In this chapter we are going to introduce a machine learning algorithm that tries to build predictive models **using only the most informative features**.
- In this context an informative feature is a **descriptive feature** whose values split the instances in the dataset into **homogeneous sets** with respect to the target feature value.



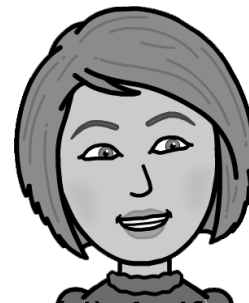
(a) Brian



(b) John



(c) Aphra



(d) Aoife

Figure: Cards showing character faces and names for the *Guess-Who* game

<u>MAN</u>	<u>LONG HAIR</u>	<u>GLASSES</u>	<u>NAME</u>
Yes	No	Yes	Brian
Yes	No	No	John
No	Yes	No	Aphra
No	No	No	Aoife

By asking questions, we want to determine a person's name

Which question would you ask first:

- 1 Is it a man?
- 2 Does the person wear glasses?

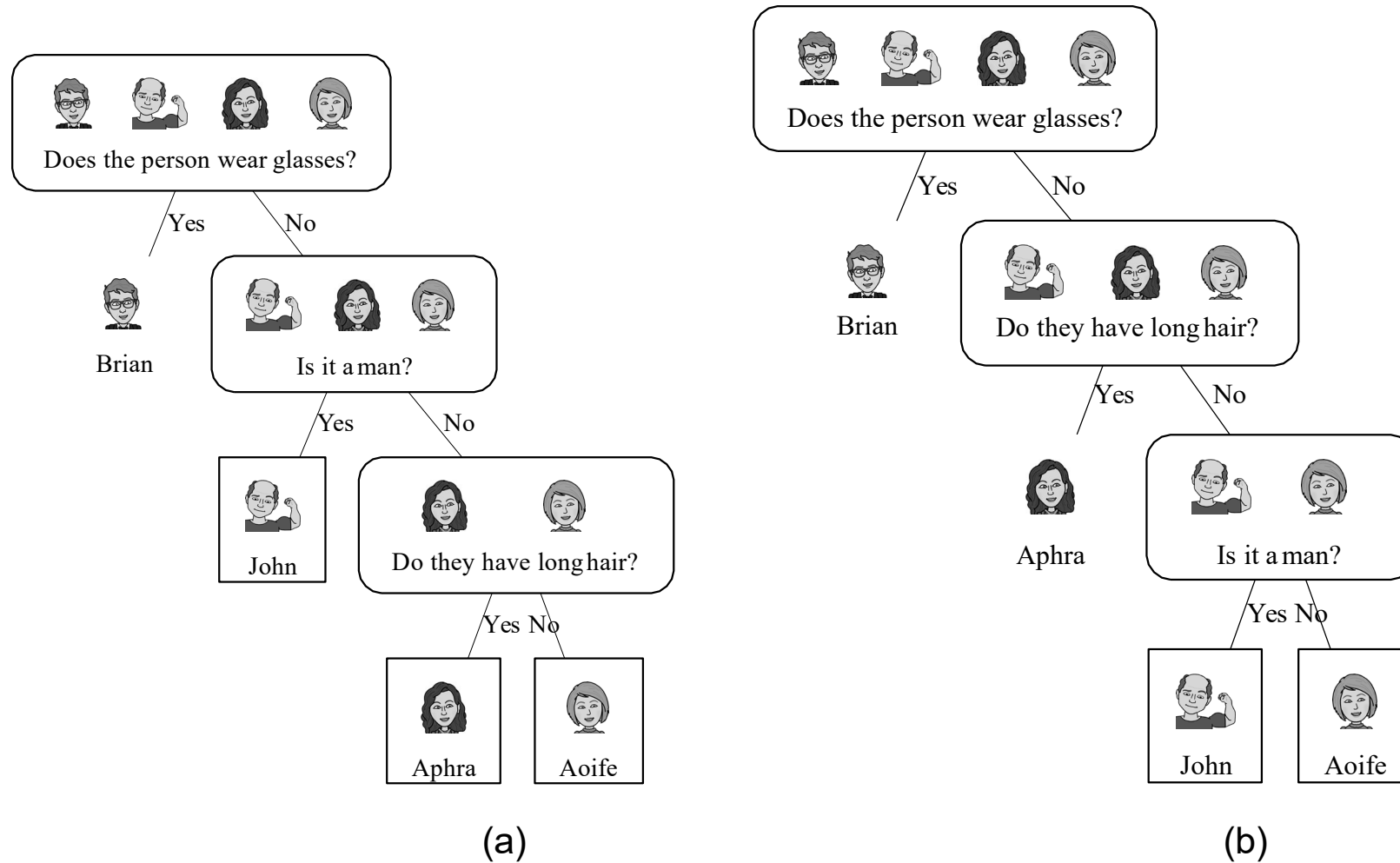
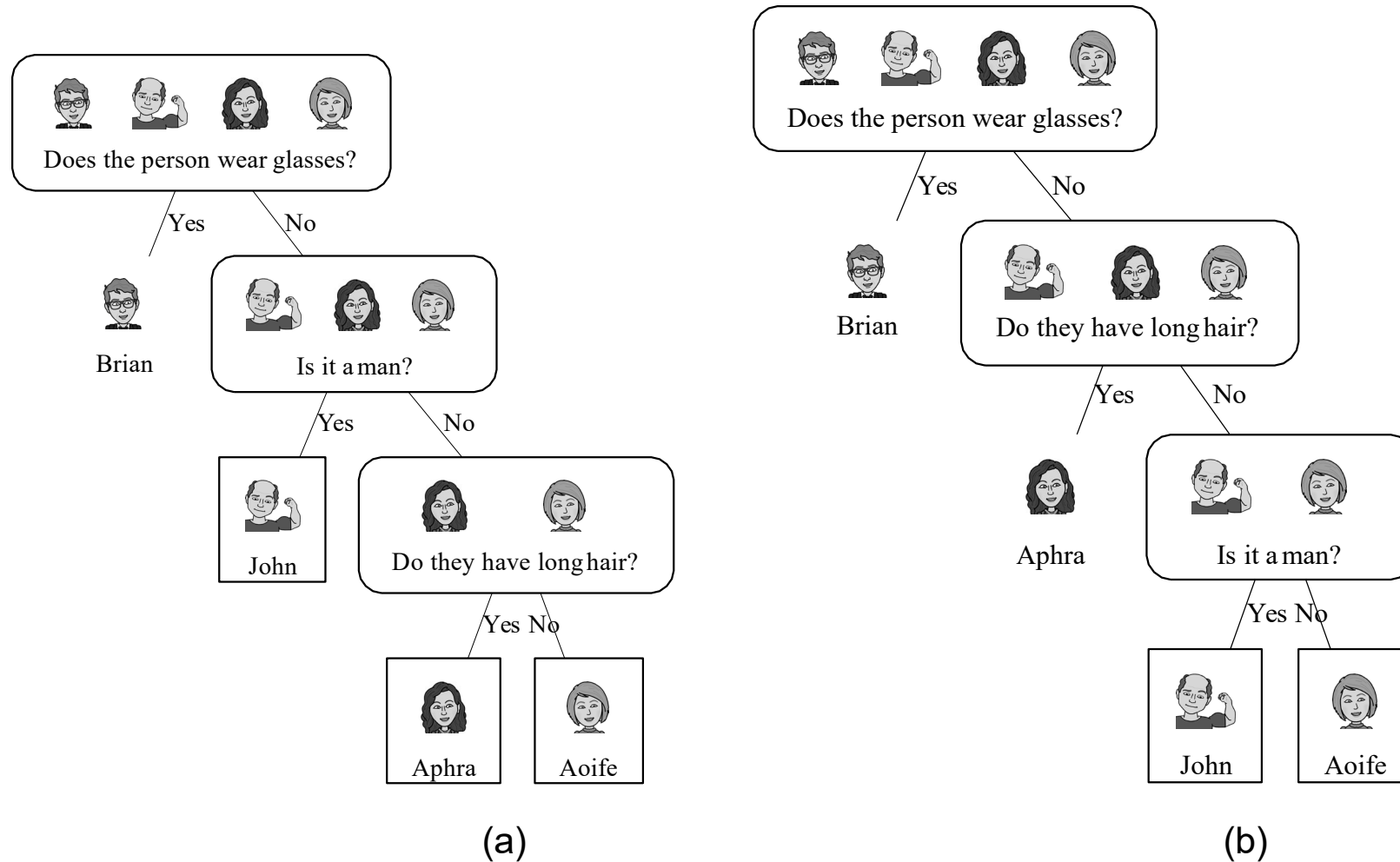


Figure: The different question sequences that can follow in a game of *Guess-Who* beginning with the question **Does the person wear glasses?**

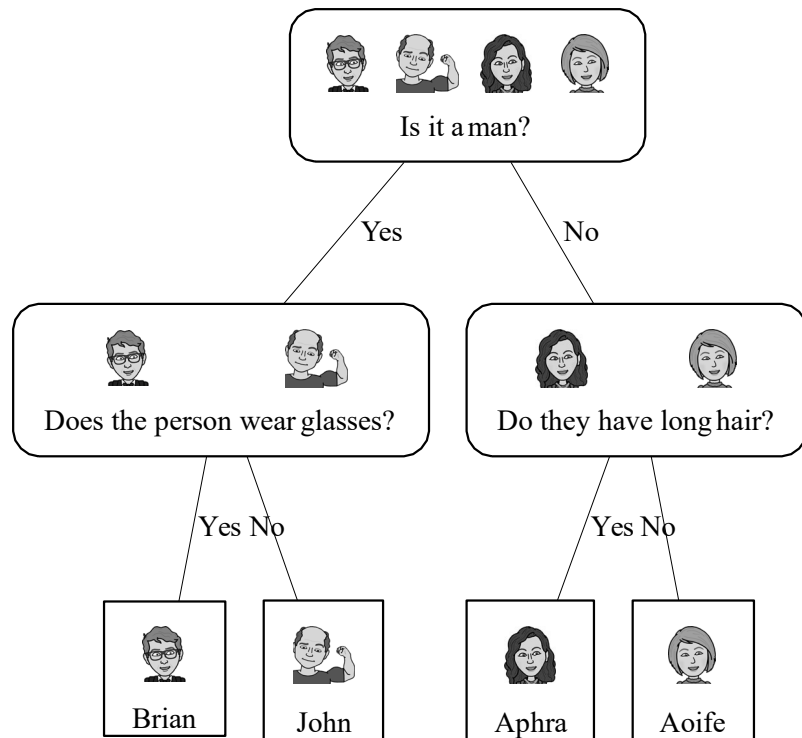


In both of the diagrams:

- One path is 1 question long, one is 2 questions long and two paths are 3 long.

Consequently, if you ask Question (2) first the average number of questions you have to ask per game is $\frac{1+2+3+3}{4} = 2.25$

Figure: The different question sequences that can follow in a game of *Guess-Who* beginning with the question **Does the person wear glasses?**



All the paths in this diagram are two questions long.

So, on average if you ask Question (1) first the average number of questions you have to ask per game is $\frac{2+2+2+2}{4} = 2$

Figure: The different question sequences that can follow in a game of *Guess-Who* beginning with the question **Is it a man?**

On average getting an answer to Question (1) seems to give you more information than an answer to Question (2): less follow up questions.

This is not because of the literal message of the answers: YES or NO.

It has to do with how the answer to each questions splits the domain into different sized sets based on the value of the descriptive feature the question is asked about and the likelihood of each possible answer to the question.

Big Idea

So the big idea here is to figure out which features are the most informative ones to ask questions about by considering the effects of the different answers to the questions, in terms of:

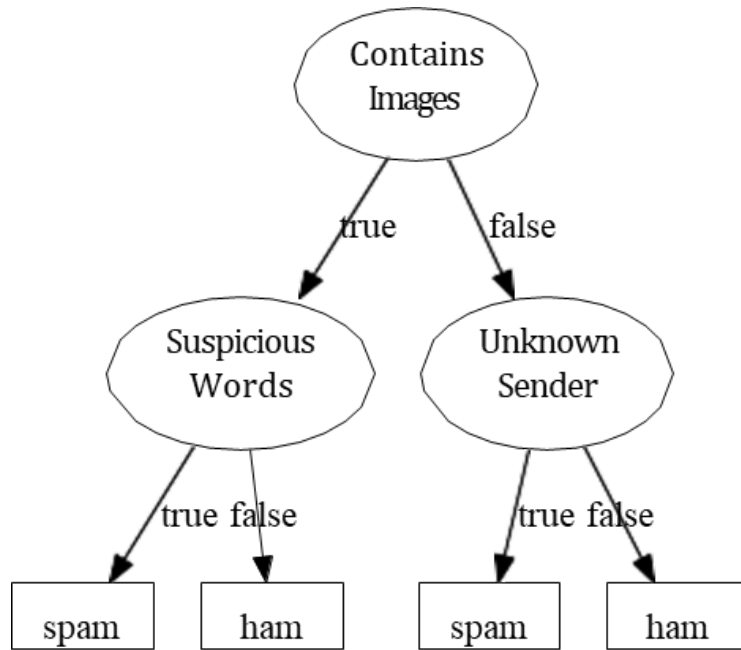
- 1 how the domain is split up after the answer is received,
- 2 and the likelihood of each of the answers.

Fundamentals

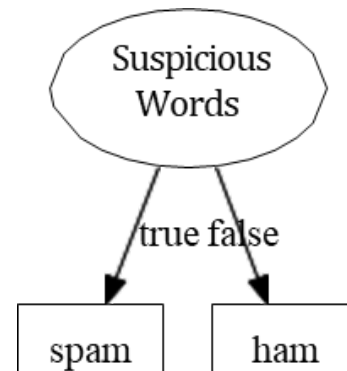
- A decision tree consists of:
 - 1 a **root node** (or starting node),
 - 2 **interior nodes**
 - 3 and **leaf nodes** (or terminating nodes).
- Each of the non-leaf nodes (root and interior) in the tree specifies a test to be carried out on one of the query's descriptive features.
- Each of the leaf nodes specifies a predicted classification for the query.

Table: An email spam prediction dataset.

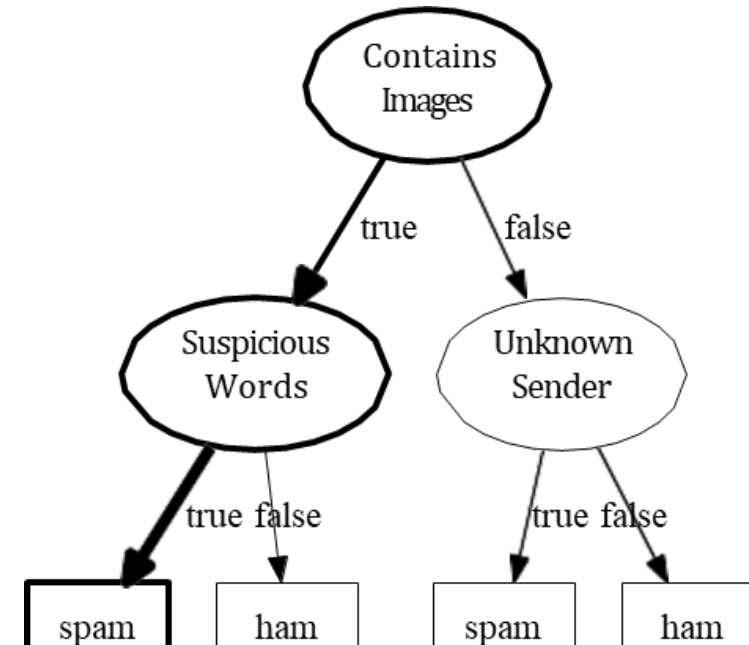
ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham



(a)



(b)



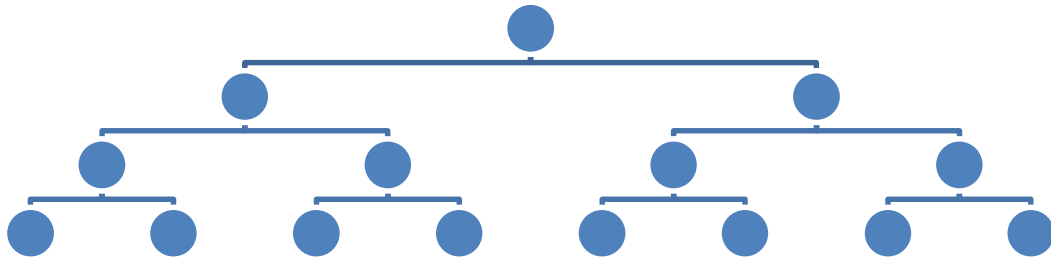
(c)

Figure: (a) and (b) show two decision trees that are consistent with the instances in the spam dataset. (c) shows the path taken through the tree shown in (a) to make a prediction for the query instance: SUSPICIOUS WORDS = *'true'*, UNKNOWN SENDER = *'true'*, CONTAINS IMAGES = *'true'*.

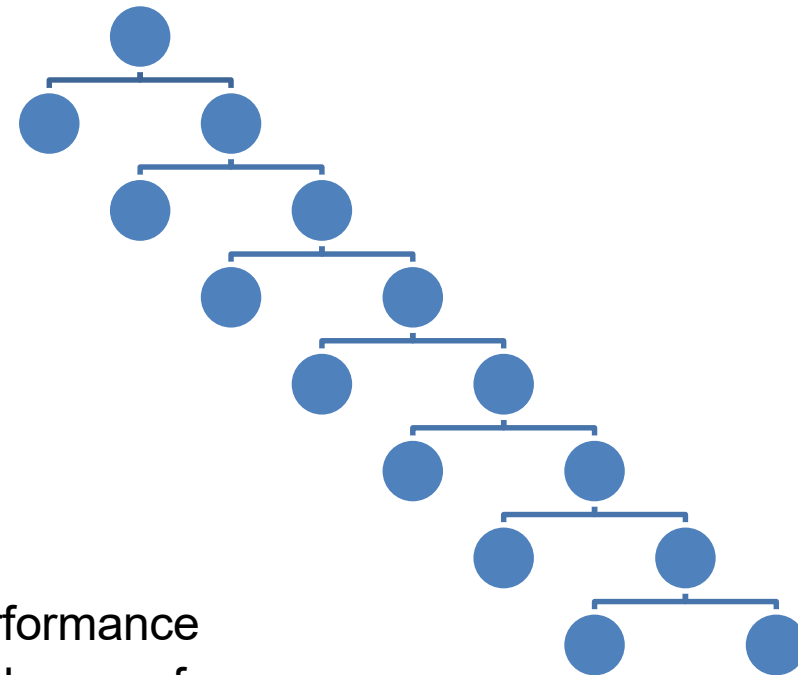
- Both of these trees will return identical predictions for all the examples in the dataset.
- So, which tree should we use?
- Apply the same approach as we used in the
- *Guess-Who* game: prefer decision trees that use less tests (shallower trees).
- This is (sort of) an example of Occam's Razor.

Which tree is better?

If equiprobable, the average number of steps is $\propto \log_2 N$



If equiprobable, the average number of steps is $\propto N - 1$



- Tree depth impacts accuracy as well as performance
 - More decisions to make means more chance of error, and less robustness in the presence of noise

How do we create shallow trees?

- The tree that tests SUSPICIOUS WORDS at the root is very shallow because the SUSPICIOUS WORDS feature perfectly splits the data into pure groups of '*spam*' and '*ham*'.
- Descriptive features that split the dataset into pure sets with respect to the target feature provide information about the target feature.
- So we can make shallow trees by testing the informative features early on in the tree.
- All we need to do that is a computational metric of the purity of a set: **entropy**

Claude Shannon's entropy model defines a computational measure of the impurity of the elements of a set.

An easy way to understand the entropy of a set is to think in terms of the uncertainty associated with guessing the result if you were to make a random selection from the set.

Entropy is related to the probability of a outcome.

High probability → Low entropy

Low probability → High entropy

If we take the **log** of a probability and multiply it by -1 we get this mapping!

Shannon's model of entropy is a weighted sum of the logs of the probabilities of each of the possible outcomes when we make a random selection from a set.

$$H(t) = - \sum_{i=1}^S \left(p(t = i) \log(p(t = i)) \right)$$

What is the entropy of a set of 52 different playing cards?

$$\begin{aligned}
 H(card) &= - \sum_{i=1}^{52} p(card = i) \log_2 p(card = i) \\
 &= - \sum_{i=1}^{52} \frac{1}{52} \log_2 \left(\frac{1}{52} \right) = - \sum_{i=1}^{52} 0.019 \log_2 (0.019) \\
 &= - \sum_{i=1}^{52} -0.1096 = 5.700 \text{ bits}
 \end{aligned}$$

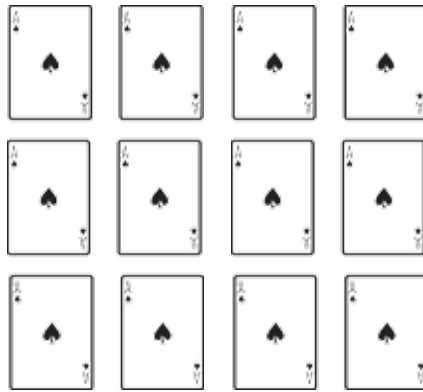
What does this mean – what is the “entropy of a deck of cards”?

What is the entropy of a set of 52 different playing cards?

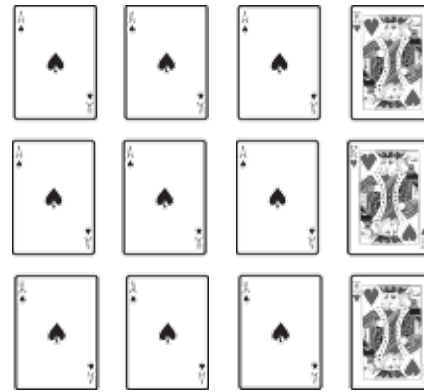
$$H(card) = 5.700 \text{ bits}$$

What does this mean – what is the “entropy of a deck of cards”?

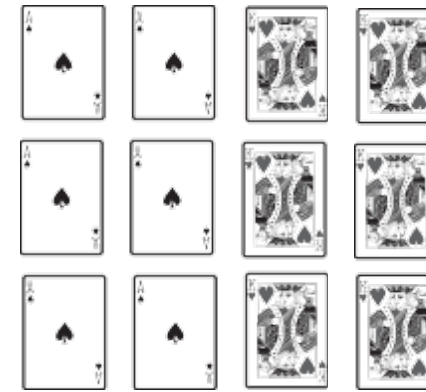
- It's an indication that drawing one card has many possible outcomes (52) and that the probability of any one of them is low
- It's the uncertainty of guessing a result chosen at random
- It's a measure of the diversity (heterogeneity) of a set
- A good decision tree will contain queries that reduce the total entropy in the sample



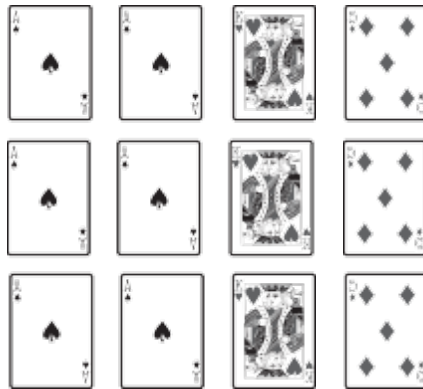
(a) $H(card) = 0.00$



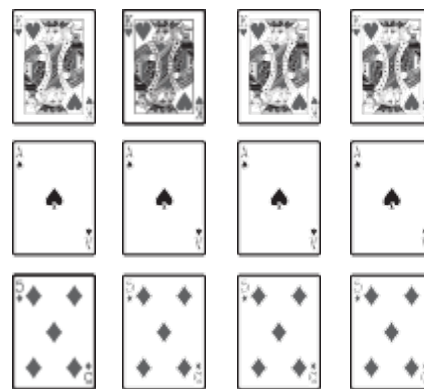
(b) $H(card) = 0.81$



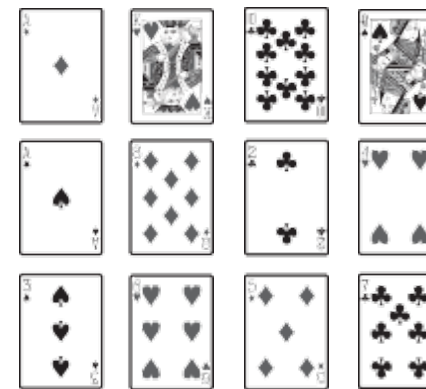
(c) $H(card) = 1.00$



(d) $H(card) = 1.50$



(e) $H(card) = 1.58$



(f) $H(card) = 3.58$

The entropy of different sets of playing cards measured in bits.

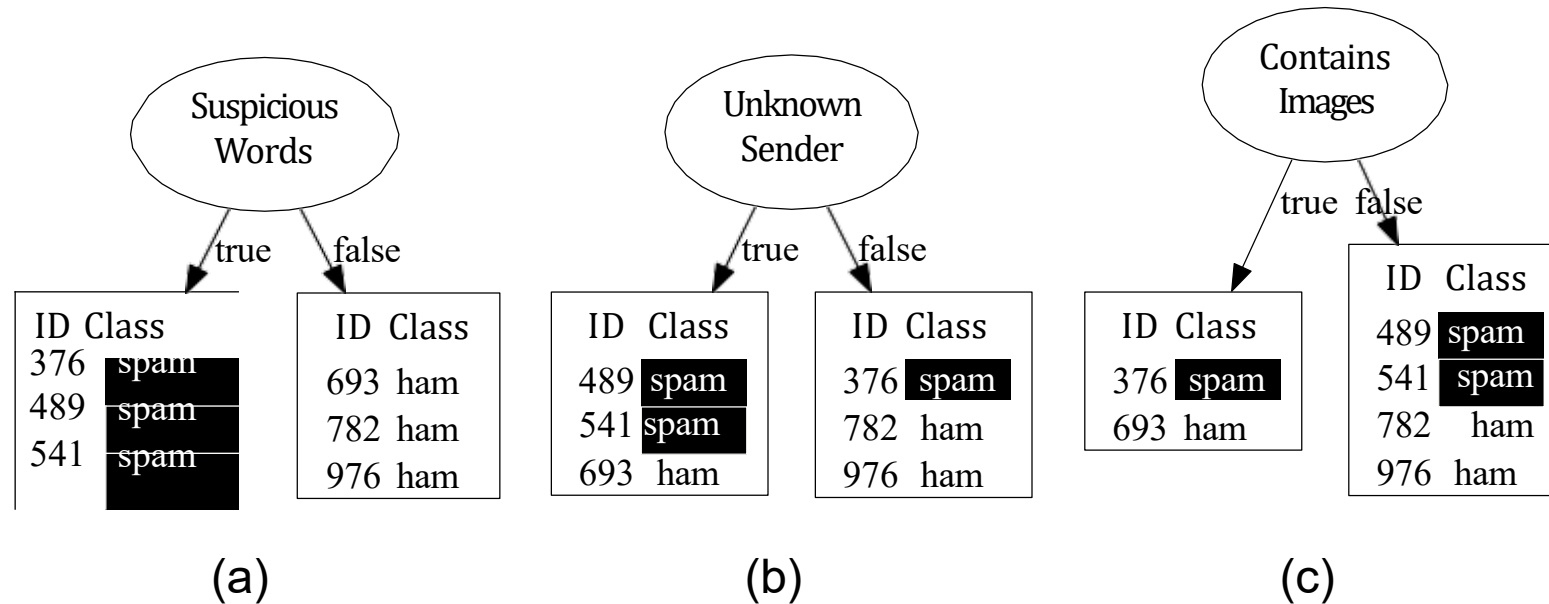


Figure: How the instances in the spam dataset split when we partition using each of the different descriptive features from the spam dataset in Table [1](#) ^[15]

Our intuition is that the ideal discriminatory feature will partition the data into **pure** subsets where all the instances in each subset have the same classification.


SUSPICIOUS WORDS perfect split.

UNKNOWN SENDER mixture but some information (when *'true'* most instances are *'spam'*).

CONTAINS IMAGES no information.

One way to implement this idea is to use a metric called **information gain**.

Information Gain

-  The information gain of a descriptive feature can be understood as a measure of the reduction in the overall entropy of a prediction task by testing on that feature.

Computing information gain involves the following 3 equations:

$$H(t, \mathcal{D}) = - \sum_{l \in \text{levels}(t)} [p(t = l) \log_2(p(t = l))]$$

$$\text{rem}(d, \mathcal{D}) = \sum_{l \in \text{levels}(d)} \underbrace{\frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|}}_{\text{weighting}} \underbrace{H(t, \mathcal{D}_{d=l})}_{\text{entropy of partition } \mathcal{D}_{d=l}}$$

$$IG(d, \mathcal{D}) = H(t, \mathcal{D}) - \text{rem}(d, \mathcal{D})$$

Calculate the **entropy** for the target feature in the dataset:

$$H(t, \mathcal{D}) = - \sum_{l \in \text{levels}(t)} [p(t = l) \log_2(p(t = l))]$$

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

Calculate the **entropy** for the target feature in the dataset:

$$H(t, \mathcal{D}) = - \sum_{l \in \text{levels}(t)} [p(t = l) \log_2(p(t = l))]$$

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

$$\begin{aligned}
 H(t, \mathcal{D}) &= - \sum_{l \in \text{levels}(t)} [p(t = l) \log_2(p(t = l))] \\
 &= -[p(t = \text{"spam"}) \log_2(p(t = \text{"spam"})) + p(t = \text{"ham"}) \log_2(p(t = \text{"ham"}))] \\
 &= -\left[\frac{1}{2} \log_2\left(\frac{1}{2}\right) + \frac{1}{2} \log_2\left(\frac{1}{2}\right)\right] = -[-\log_2(2)] = 1 \text{ bit}
 \end{aligned}$$

Calculate the **remainder** for the UNKNOWN SENDER feature.

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} H(t, \mathcal{D}_{d=l})$$

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

$$rem(SENDER, \mathcal{D}) = \frac{|\mathcal{D}_{SENDER=T}|}{|\mathcal{D}|} H(t, \mathcal{D}_{SENDER=T}) + \frac{|\mathcal{D}_{SENDER=F}|}{|\mathcal{D}|} H(t, \mathcal{D}_{SENDER=F})$$

$$\begin{aligned}
 &= -\frac{3}{6} \sum_{l \in \{"spam", "ham"\}} [p(t=l) \log_2(p(t=l))] - \frac{3}{6} \sum_{l \in \{"spam", "ham"\}} [p(t=l) \log_2(p(t=l))] \\
 &= -\frac{3}{6} \left[\frac{2}{3} \log_2 \left(\frac{2}{3} \right) + \frac{1}{3} \log_2 \left(\frac{1}{3} \right) \right] - \frac{3}{6} \left[\frac{1}{3} \log_2 \left(\frac{1}{3} \right) + \frac{2}{3} \log_2 \left(\frac{2}{3} \right) \right] = 0.9183 \text{ bits}
 \end{aligned}$$

Calculate the **remainder** for the CONTAINS IMAGES feature.

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} H(t, \mathcal{D}_{d=l})$$

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

$$rem(\text{IMAGES}, \mathcal{D}) = \frac{|\mathcal{D}_{\text{IMAGES}=T}|}{|\mathcal{D}|} H(t, \mathcal{D}_{\text{IMAGES}=T}) + \frac{|\mathcal{D}_{\text{IMAGES}=F}|}{|\mathcal{D}|} H(t, \mathcal{D}_{\text{IMAGES}=F})$$

$$\begin{aligned}
 &= -\frac{2}{6} \sum_{l \in \{\text{"spam"}, \text{"ham"}\}} [p(t=l) \log_2(p(t=l))] - \frac{4}{6} \sum_{l \in \{\text{"spam"}, \text{"ham"}\}} [p(t=l) \log_2(p(t=l))] \\
 &= -\frac{2}{6} \left[\frac{1}{2} \log_2 \left(\frac{1}{2} \right) + \frac{1}{2} \log_2 \left(\frac{1}{2} \right) \right] - \frac{4}{6} \left[\frac{2}{4} \log_2 \left(\frac{2}{4} \right) + \frac{2}{4} \log_2 \left(\frac{2}{4} \right) \right] = 1 \text{ bit}
 \end{aligned}$$

Calculate the **information gain** for the three descriptive feature in the dataset.

$$IG(d, \mathcal{D}) = H(t, \mathcal{D}) - \text{rem}(d, \mathcal{D})$$

$$IG(\text{SUSPICIOUS WORDS}, D) = H(\text{CLASS}, D) - \text{rem}(\text{SUSPICIOUS WORDS}, D) = 1 - 0 = 1 \text{ bit}$$

$$IG(\text{UNKNOWN SENDER}, D) = H(\text{CLASS}, D) - \text{rem}(\text{UNKNOWN SENDER}, D) = 1 - 0.9183 = 0.0817 \text{ bits}$$

$$IG(\text{CONTAINS IMAGES}, D) = H(\text{CLASS}, D) - \text{rem}(\text{CONTAINS IMAGES}, D) = 1 - 1 = 0 \text{ bits}$$

The results of these calculations match our intuitions.

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

Standard Approach: The ID3 Algorithm

- ID3 Algorithm (Iterative Dichotomizer 3)
- Attempts to create the shallowest tree that is consistent with the data that it is given.
- The ID3 algorithm builds the tree in a recursive, depth-first manner, beginning at the root node and working down to the leaf nodes.

- 1 The algorithm begins by choosing the best descriptive feature to test (i.e., the best question to ask first) using **information gain**.
- 2 A root node is then added to the tree and labelled with the selected test feature.
- 3 The training dataset is then partitioned using the test.
- 4 For each partition a branch is grown from the node.
- 5 The process is then repeated for each of these branches **using the relevant partition of the training set in place of the full training set and with the selected test feature excluded from further testing.**

The algorithm defines three situations where the recursion stops and a leaf node is constructed:

- 1 All of the instances in the dataset have the same classification (target feature value) then return a leaf node tree with that classification as its label.
- 2 The set of features left to test is empty then return a leaf node tree with the majority class of the dataset as its classification (*Note: data set is not consistent*)
- 3 The dataset is empty return a leaf node tree with the majority class of the dataset at the parent node that made the recursive call.

- The ID3 algorithm works the same way for larger more complicated datasets.
- There have been many extensions and variations proposed for the ID3 algorithm:
 - using different impurity measures (Gini, Gain Ratio)
 - handling continuous descriptive features
 - to handle continuous targets
 - pruning to guard against over-fitting
 - using decision trees as part of an ensemble (Random Forests)
- We cover these extensions in Section 4.4.

Let's look at another example (from Russell and Norvig, *Artificial Intelligence*)

- Problem: decide whether to wait for a table at a restaurant, based on the following attributes:
 1. Alternate: is there an alternative restaurant nearby?
 2. Bar: is there a comfortable bar area to wait in?
 3. Fri/Sat: is today Friday or Saturday?
 4. Hungry: are we hungry?
 5. Patrons: number of people in the restaurant (None, Some, Full)
 6. Price: price range (\$, \$\$, \$\$\$)
 7. Raining: is it raining outside?
 8. Reservation: have we made a reservation?
 9. Type: kind of restaurant (French, Italian, Thai, Burger)
 10. WaitEstimate: estimated waiting time (0-10, 10-30, 30-60, >60)

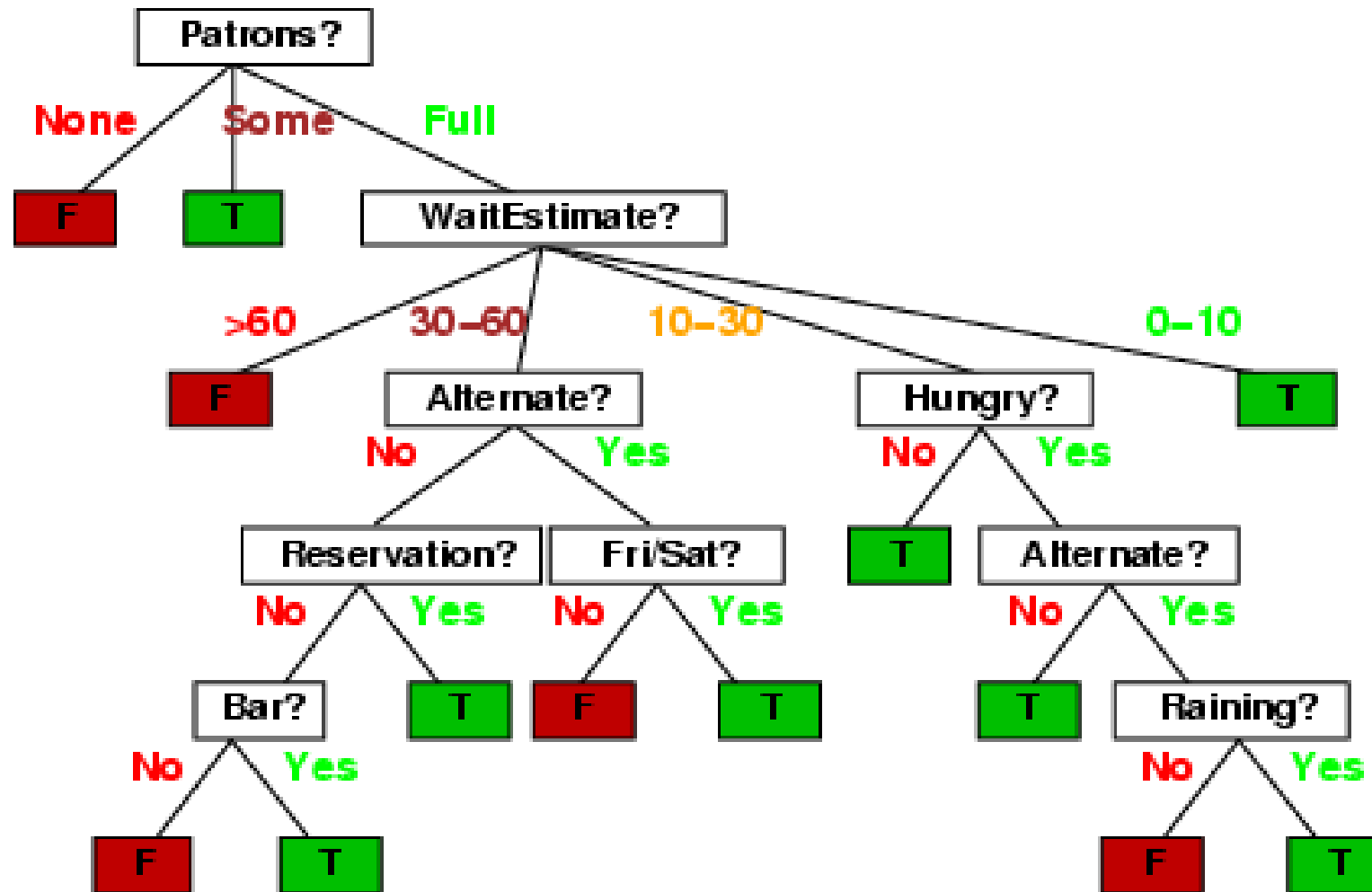
Attribute-based representations

- Examples described by **attribute values** (Boolean, categorical, continuous)
- E.g., situations where I will/won't wait for a table:

Example	Attributes										Target
	<i>Alt</i>	<i>Bar</i>	<i>Fri</i>	<i>Hun</i>	<i>Pat</i>	<i>Price</i>	<i>Rain</i>	<i>Res</i>	<i>Type</i>	<i>Est</i>	<i>Wait</i>
X_1	T	F	F	T	Some	\$\$\$	F	T	French	0-10	T
X_2	T	F	F	T	Full	\$	F	F	Thai	30-60	F
X_3	F	T	F	F	Some	\$	F	F	Burger	0-10	T
X_4	T	F	T	T	Full	\$	F	F	Thai	10-30	T
X_5	T	F	T	F	Full	\$\$\$	F	T	French	>60	F
X_6	F	T	F	T	Some	\$\$	T	T	Italian	0-10	T
X_7	F	T	F	F	None	\$	T	F	Burger	0-10	F
X_8	F	F	F	T	Some	\$\$	T	T	Thai	0-10	T
X_9	F	T	T	F	Full	\$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	\$\$\$	F	T	Italian	10-30	F
X_{11}	F	F	F	F	None	\$	F	F	Thai	0-10	F
X_{12}	T	T	T	T	Full	\$	F	F	Burger	30-60	T

- Classification of examples is **positive** (T) or **negative** (F)

Here is one tree for deciding whether to wait



Decision tree learning

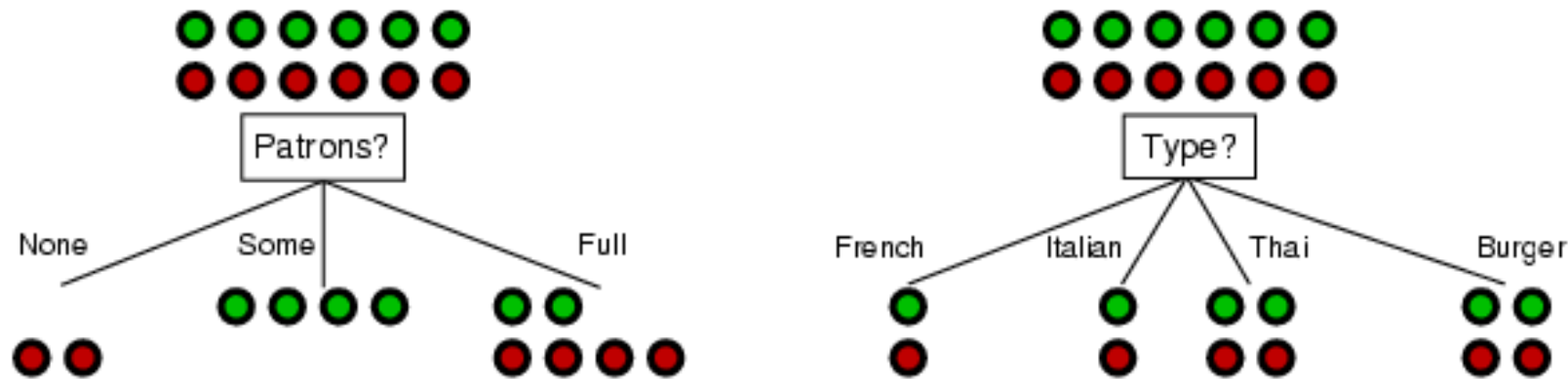
- Aim: find a small tree consistent with the training examples
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree

```

function DTL(examples, attributes, default) returns a decision tree
  if examples is empty then return default
  else if all examples have the same classification then return the classification
  else if attributes is empty then return MODE(examples)
  else
    best ← CHOOSE-ATTRIBUTE(attributes, examples)
    tree ← a new decision tree with root test best
    for each value  $v_i$  of best do
      examplesi ← {elements of examples with best =  $v_i$ }
      subtree ← DTL(examplesi, attributes − best, MODE(examples))
      add a branch to tree with label  $v_i$  and subtree subtree
    return tree
  
```

Choosing an attribute

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



- Patrons?* is a better choice

Using information theory (ID3)

- To implement `Choose-Attribute` in the DTL algorithm
- Information Content (Entropy):

$$I(P(v_1), \dots, P(v_n)) = \sum_{i=1} -P(v_i) \log_2 P(v_i)$$

- For a training set containing p positive examples and n negative examples:

$$I\left(\frac{p}{p+n}, \frac{n}{p+n}\right) = -\frac{p}{p+n} \log_2 \frac{p}{p+n} - \frac{n}{p+n} \log_2 \frac{n}{p+n}$$

Information gain

- A chosen attribute A divides the training set E into subsets E_1, \dots, E_v according to their values for A , where A has v distinct values.

$$\text{remainder}(A) = \sum_{i=1}^v \frac{p_i + n_i}{p + n} I\left(\frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i}\right)$$

- Information Gain (IG) or reduction in entropy from the attribute test:

$$IG(A) = I\left(\frac{p}{p + n}, \frac{n}{p + n}\right) - \text{remainder}(A)$$

- Choose the attribute with the largest IG

Information gain

For the training set, $p = n = 6$, $I(6/12, 6/12) = 1$ bit

Consider the attributes *Patrons* and *Type* (and others too):

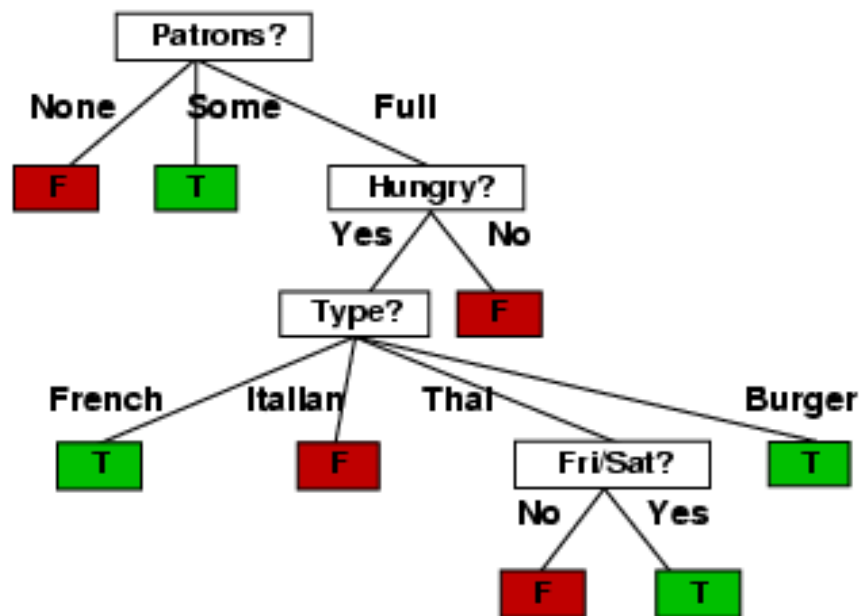
$$IG(Patrons) = 1 - \left[\frac{2}{12} I(0,1) + \frac{4}{12} I(1,0) + \frac{6}{12} I\left(\frac{2}{6}, \frac{4}{6}\right) \right] = .0541 \text{ bits}$$

$$IG(Type) = 1 - \left[\frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{2}{12} I\left(\frac{1}{2}, \frac{1}{2}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) + \frac{4}{12} I\left(\frac{2}{4}, \frac{2}{4}\right) \right] = 0 \text{ bits}$$

Patrons has the highest IG of all attributes and so is chosen by the ID3 algorithm as the root

Example contd.

- Decision tree learned from the 12 examples:



- Substantially simpler than “true” tree---a more complex hypothesis isn’t justified by small amount of data

Let's see how a decision tree might be implemented in Python using scikit-learn

<u>Color</u>	<u>Size</u>	<u>Price</u>	<u>Sales</u>	<u>SalesLevel</u>
Black	S	10	200	Med
Black	M	10	400	High
Black	L	10	375	High
Red	S	9	120	Low
Red	M	9	225	Med
Red	L	9	200	Med
Red	S	16	100	Low
Red	M	16	120	Low
Red	L	16	100	Low
Green	S	10	80	Low
Green	M	10	190	Med
Green	L	10	120	Med

This dataset represents
sales both as a continuous
variable and as a binned
categorical

Represent numerically for
use in learn

<u>Color</u>	<u>Size</u>	<u>Price</u>	<u>Sales</u>	<u>SalesLevel</u>
Black	S	10	200	Med
Black	M	10	400	High
Black	L	10	375	High
Red	S	9	120	Low
Red	M	9	225	Med
Red	L	9	200	Med
Red	S	16	100	Low
Red	M	16	120	Low
Red	L	16	100	Low
Green	S	10	80	Low
Green	M	10	190	Med
Green	L	10	120	Med

<u>ColorIndex</u>	<u>SizeIndex</u>	<u>Price</u>	<u>SalesLevelIndex</u>
1	1	10	2
1	2	10	3
1	3	10	3
2	1	9	1
2	2	9	2
2	3	9	2
2	1	16	1
2	2	16	1
2	3	16	1
3	1	10	1
3	2	10	2
3	3	10	2

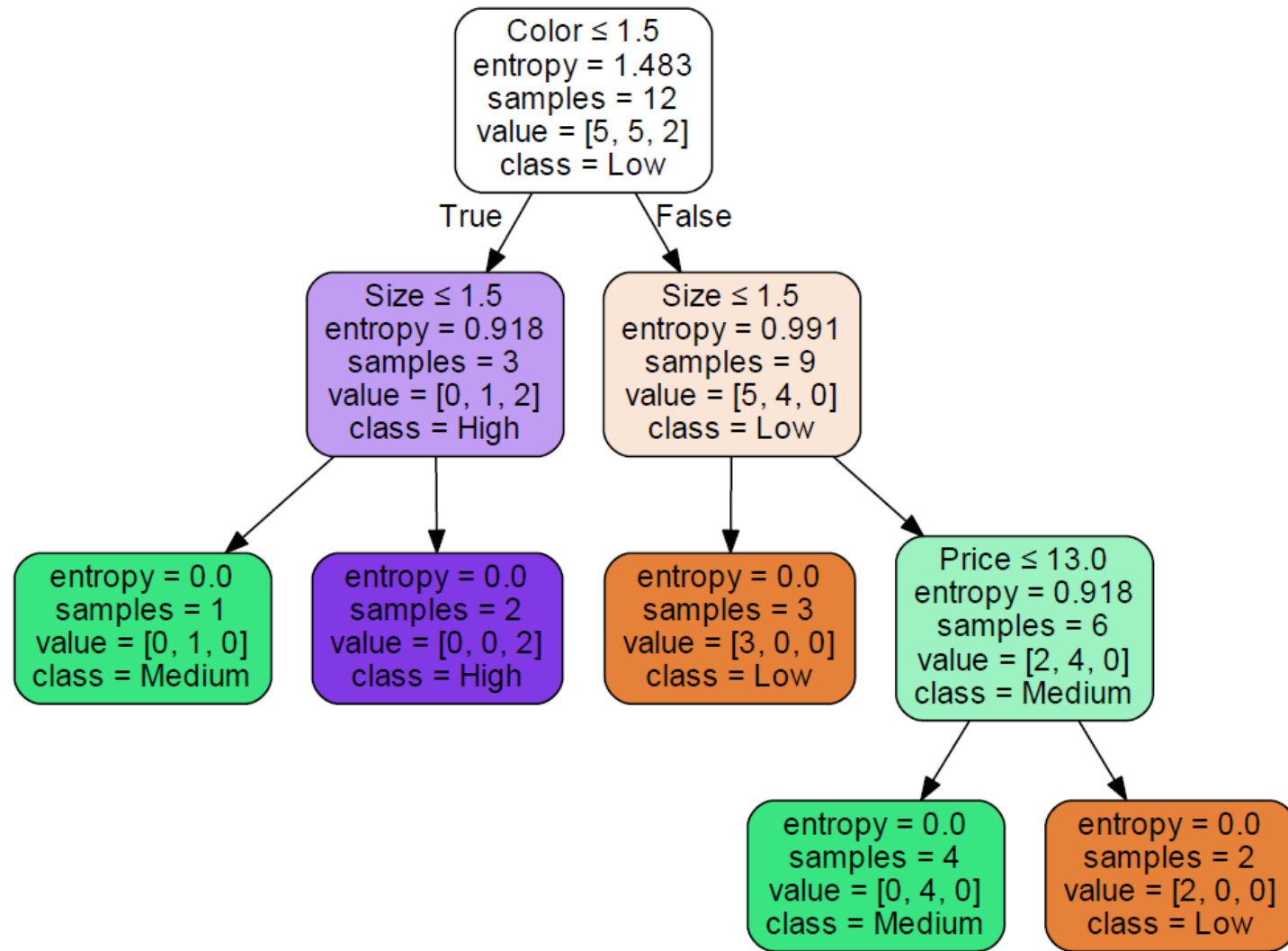
```
"""
```

```
TestID3 - Creed Jones created on Feb 12 2020  
A simple exploration of decision tree classifiers
```

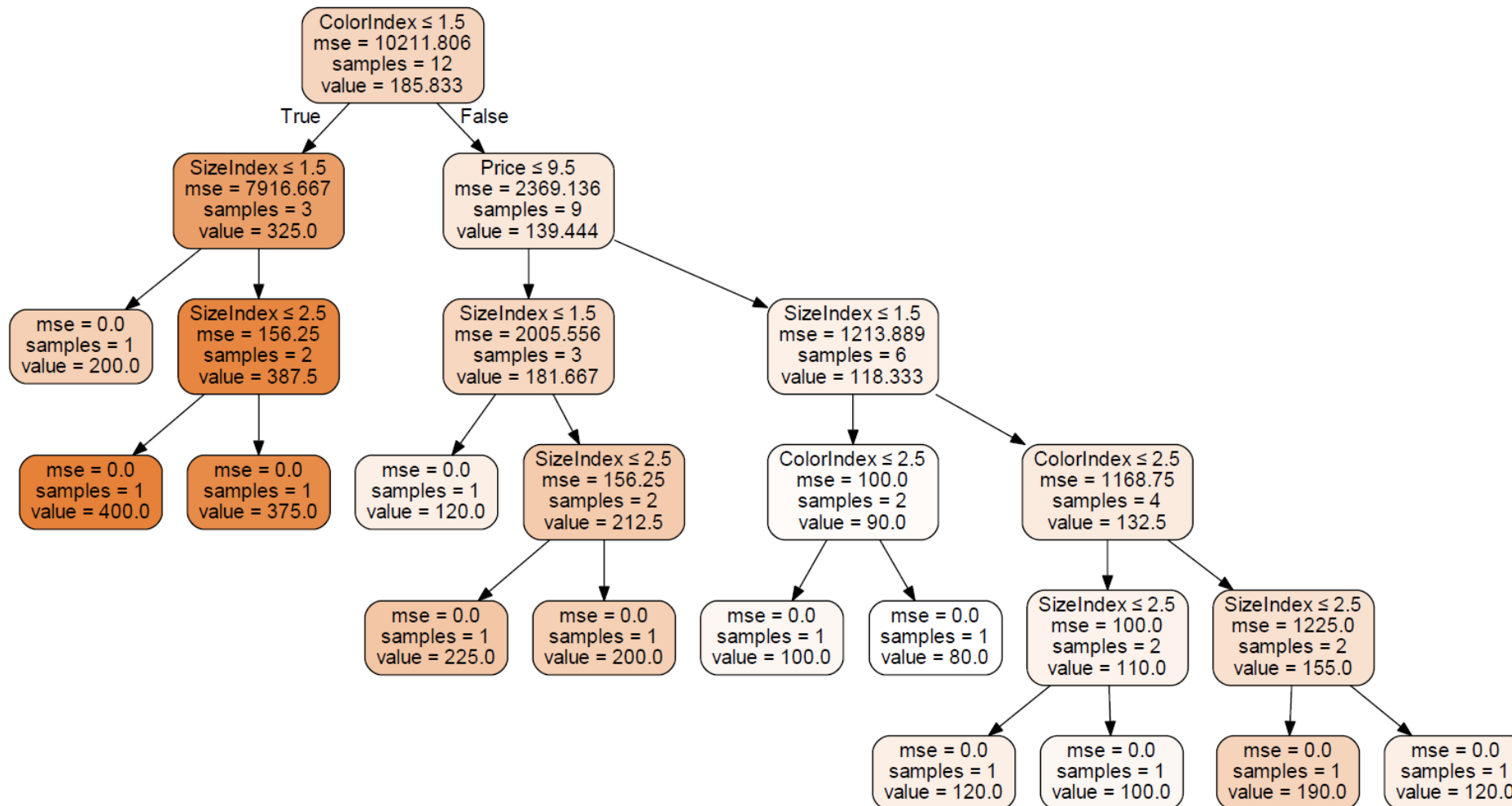
```
"""
```

```
from sklearn import tree                                # the decision tree functionality  
import pydot                                           # to write the tree as a pdf image  
  
shirts = [[1,1,10], [1,2,10], [1,3,10], [2,1,9], [2,2,9], [2,3,9], [2,1,16],  
          [2,2,16], [2,3,16], [3,1,10], [3,2,10], [3,3,10]]  
F = [2, 3, 3, 1, 2, 2, 1, 1, 1, 1, 2, 2]  
shirtNames = ["ColorIndex", "SizeIndex", "Price"]  
targetNames = ["Low", "Medium", "High"]  
  
clf = tree.DecisionTreeClassifier(criterion="entropy") # create a tree object to do classification  
clf = clf.fit(shirts, F)                             # train it on this data  
  
dot_data = tree.export_graphviz(clf, out_file=None,    # merely to write the tree out  
                                feature_names=shirtNames,  
                                class_names=targetNames,  
                                filled=True, rounded=True,  
                                special_characters=True)  
(graph,) = pydot.graph_from_dot_data(dot_data)  
graph.write_png("shirtstree.png")
```

The resulting tree will properly classify any sales data that is consistent with the training set, using the predict() method



We can also use a regression tree to estimate the actual sales figure



```
"""
TestID3 - Creed Jones created on Feb 12 2020
A simple exploration of decision tree classifiers - using pandas for Excel I/O
"""

from sklearn import tree
import graphviz
import pandas as pd

pathName = "C:\\\\Data\\"
dataFrame = pd.read_excel(pathName + 'Shirts.xlsx', sheet_name='rawData')

X = dataFrame.drop(["Sales", "SalesLevel"], axis=1)      # remove target from modeling variables
y = dataFrame.SalesLevel

targetNames = ["Low", "Medium", "High"]

clf = tree.DecisionTreeClassifier(criterion="entropy")

clf = clf.fit(X, y)

dot_data = tree.export_graphviz(clf, out_file=None,
                                feature_names=X.columns,
                                class_names=targetNames,
                                filled=True, rounded=True,
                                special_characters=True)
graph = graphviz.Source(dot_data)
graph.render("shirts")
```


Today's Objectives

Decision Tree models

- Concept
- Entropy
- Information Gain
- The ID3 algorithm for constructing efficient trees
- Examples
- Tree learning in Python