Decision Trees

- A decision tree leads us from a set of attributes (features of the input) to some output
- For example, we have a database of customer decisions about whether to see a specific movie at a multiplex theater
- These customers have made several decisions about whether to view a movie based on many attributes:
  1. *3D*: is the movie in 3D or not?
  2. *Food*: is there in-theater food service?
  3. *Genre*: type of movie (Action, Comedy, Horror, Romance, Kids, Other)
  4. *Rating*: how the movie is rated (G, PG, R)
  5. *Price*: price range ($, $$, $$$)
  6. *New*: is this opening weekend for the movie or not?
  7. *Others*: how many other choices are there (0, 1–3, 4–6, >6)
- The function we want to learn is whether a future customer will choose a film, given some set of attributes

Decisions Based on Attributes

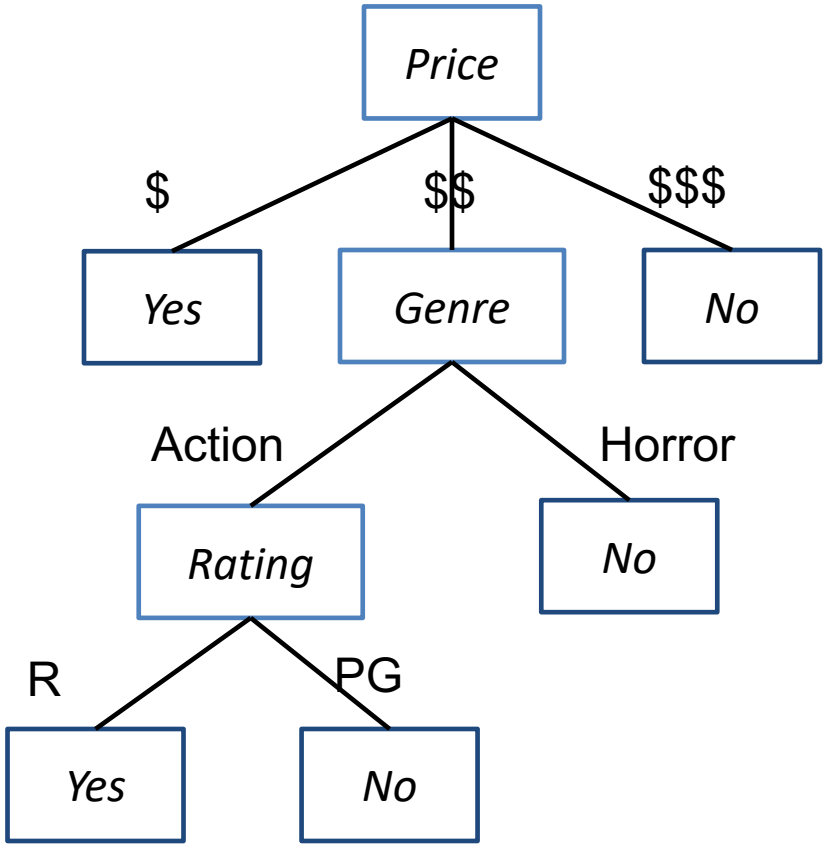
- ***Training set***: cases where patrons have decided to wait or not, along with the associated attributes for each case

Example	3D	Food	Genre	Rating	Price	New	Others	Chosen?
X_1	T	T	Action	PG	$$$	T	1–3	No
X_2	F	T	Action	R	$$	F	0	Yes
X_3	F	F	Comedy	PG	$	T	4–6	Yes
X_4	F	F	Horror	R	$$	T	1–3	No
X_5	T	F	Kids	G	$$$	F	>6	No
X_6	F	T	Action	PG	$$	F	>6	No

- We now want to learn a tree that agrees with the decisions already made, in hopes that it will allow us to predict future decisions
- In such a tree, nodes will correspond to attributes, with branches mapped to the different values of that attribute, and leaves corresponding to the decisions made by customers

Decision Tree Functions

- For the examples given, here is a "true" tree (one that will lead from the inputs to the same outputs)

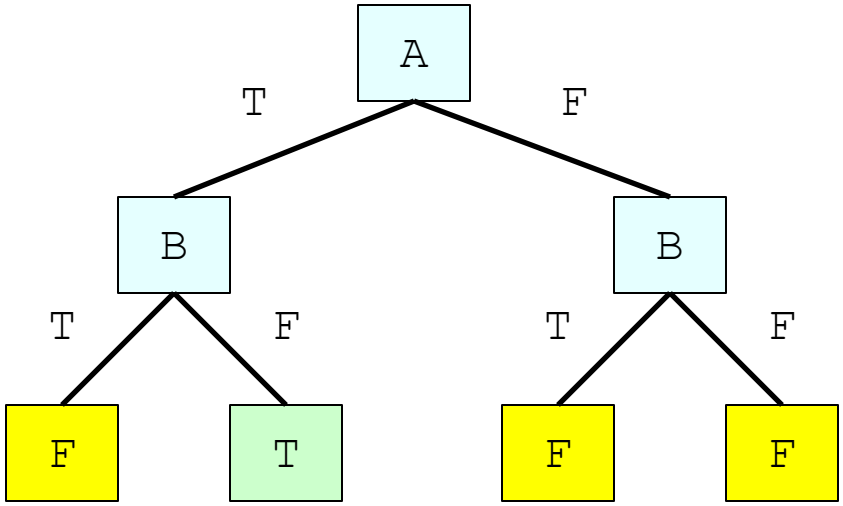


The process of constructing a tree for a data-set is another example of supervised learning.

The resulting tree-model is non-parametric: unlike all the other models we've seen, we ***don't*** learn by fitting numerical weights to features. Instead, we simply split on specific values of those features.

Decision Trees are Expressive

A	B	A && !B
T	T	F
T	F	T
F	T	F
F	F	F



- Such trees can express ***any deterministic function*** we:
  - For example, in boolean functions, each row of a truth-table will correspond to a path in a tree
  - For any such function, there is always a tree: just make each example a different path to a correct leaf output
- ***A Problem***: such trees most often do not ***generalize*** to new examples
- ***Another Problem***: we want ***compact*** trees to simplify inference

Why Not Search for Trees?

- One thing we might consider would be to search through possible trees to find ones that are most compact and consistent with our inputs
  - Exhaustive search is too expensive, however, due to the large number of possible functions (trees) that exist
- For n binary-valued attributes and boolean decision outputs, there are 2^{2^n} possibilities.
  - For 5 such attributes, we have $4{,}294{,}967{,}296$ trees!
  - Even restricting our search to conjunctions over attributes, it is easy to get 3^n possible trees.

Building Trees Top-Down

- Rather than search for all trees, we ***build*** our trees by:
  1. Choosing an attribute A from our set
  2. Dividing our examples according to the values of A
  3. Placing each subset of examples into a sub-tree below the node for attribute A
- This can be implemented in many ways but is easily understood ***recursively***
- The main question becomes: ***how*** do we choose the attribute A to split our examples?

Decision Tree Learning

DECISION-TREE-LEARNING:

Inputs: set of examples, E
set of parent examples, P
set of features, F

$guess \longleftarrow$ most frequent output for E
if (all outputs in E the same) **or** $(F = \emptyset)$:
  return a leaf containing $guess$
else if $E = \emptyset$:
  return a leaf containing most frequent output for P
else:
  $F^\star \longleftarrow$ MOSTIMPORTANT(F, E)
  $T \longleftarrow$ new decision tree with root-feature $F^\star$
  for each value f of $F^\star$:
    $E_f \longleftarrow \{x \in E \,|\, x \text{ has feature-value } f\}$
    $sub_f \longleftarrow$ DECISION-TREE-LEARNING$(E_f, E, F - F^\star)$
    add a branch to T with label f and subtree sub_f
  return: the tree T

Base Cases

```
DECISION-TREE-LEARNING:
    Inputs: set of examples, E
            set of parent examples, P
            set of features, F

    guess ⟵ most frequent output for E
    if (all outputs in E the same) or (F = ∅):
        return a leaf containing guess
    else if E = ∅:
        return a leaf containing most frequent output for P
                ⋮
```

- The algorithm stops in three cases:

1. Perfect classification of data found and use it as a decision
2. No ***features*** left: use the most common class
3. No ***data*** left: use the most common class of ***parent*** data (this is empty when a function is called initially)

Recursive Case

DECISION-TREE-LEARNING:

Inputs: set of examples, E
set of parent examples, P
set of features, F

$\vdots$

$F^\star \longleftarrow$ MOSTIMPORTANT(F, E)
$T \longleftarrow$ new decision tree with root-feature $F^\star$
for each value f of $F^\star$:
$E_f \longleftarrow \{x \in E \mid x \text{ has feature-value } f\}$
$sub_f \longleftarrow$ DECISION-TREE-LEARNING$(E_f, E, F - F^\star)$
add a branch to T with label f and subtree sub_f
return: the tree T

Note: we ***remove*** the chosen feature, so it is never reused.

MOSTIMPORTANT(): ***rates*** features for importance in making decisions about given set of examples (only complex part)

After this attribute is chosen, we divide the data according to the values of this feature, and recursively build subtrees out of each partial data-set.

Choosing "Important" Attributes

- The precise tree we build will depend upon the order in which the algorithm chooses attributes and splits up examples
- Suppose we have the following training set of 6 examples, defined by the boolean attributes A, B, C, with outputs as shown:

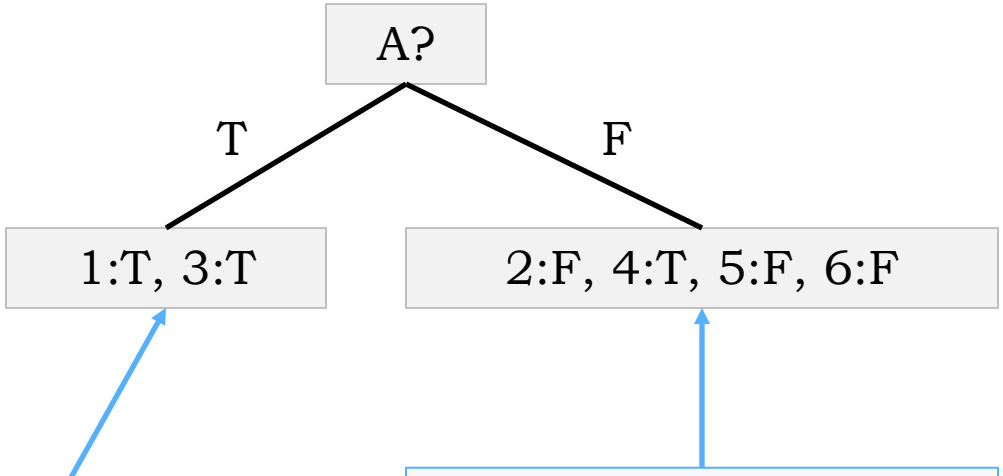
Case	A	B	C	Output
1	T	F	F	T
2	F	T	T	F
3	T	T	F	T
4	F	F	T	T
5	F	F	F	F
6	F	T	F	F

- We will consider two possible orders for the attributes when we build our tree: {A, B, C} and {C, B, A}

Choosing "Important" Attributes

- Suppose we use the order {A, B, C}: start by dividing up cases based on variable A

Case	A	B	C	Output
1	T	F	F	T
2	F	T	T	F
3	T	T	F	T
4	F	F	T	T
5	F	F	F	F
6	F	T	F	F



Here, all Outputs are the same, so we can replace this with a simple leaf node with that value.

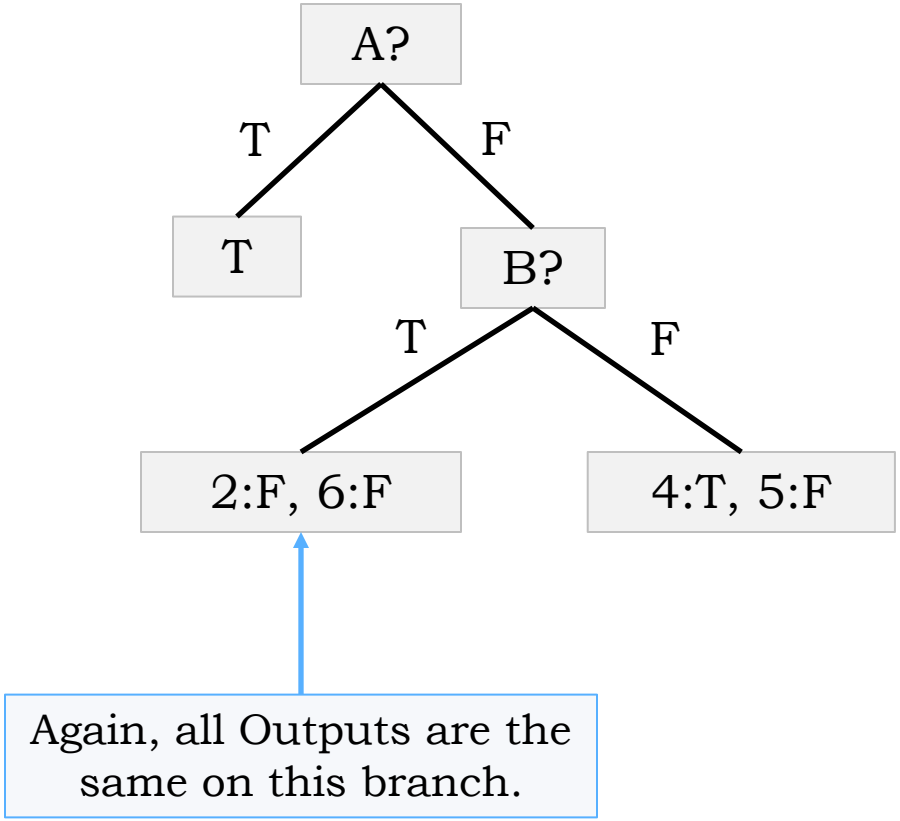
Each of these is a case for which attribute A has the right value, along with the appropriate Output value for that case.

This is an example of the ***second*** base case stopping condition of the recursive algorithm.

Choosing "Important" Attributes

- Order {A, B, C}: next, divide un-decided cases based on variable B

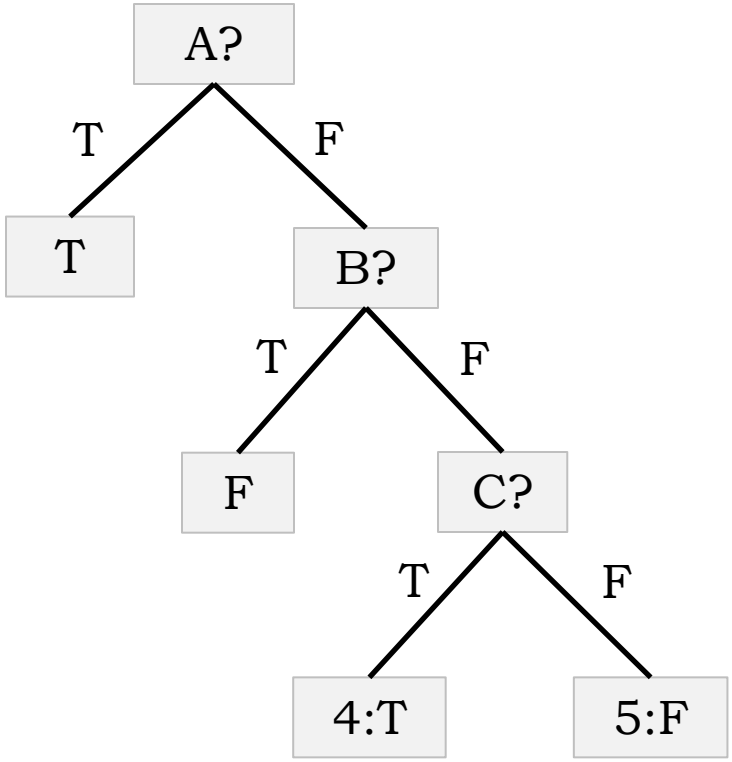
Case	A	B	C	Output
1	T	F	F	T
2	F	T	T	F
3	T	T	F	T
4	F	F	T	T
5	F	F	F	F
6	F	T	F	F



Choosing "Important" Attributes

- Order {A, B, C}: last, divide un-decided cases based on variable C

Case	A	B	C	Output
1	T	F	F	T
2	F	T	T	F
3	T	T	F	T
4	F	F	T	T
5	F	F	F	F
6	F	T	F	F



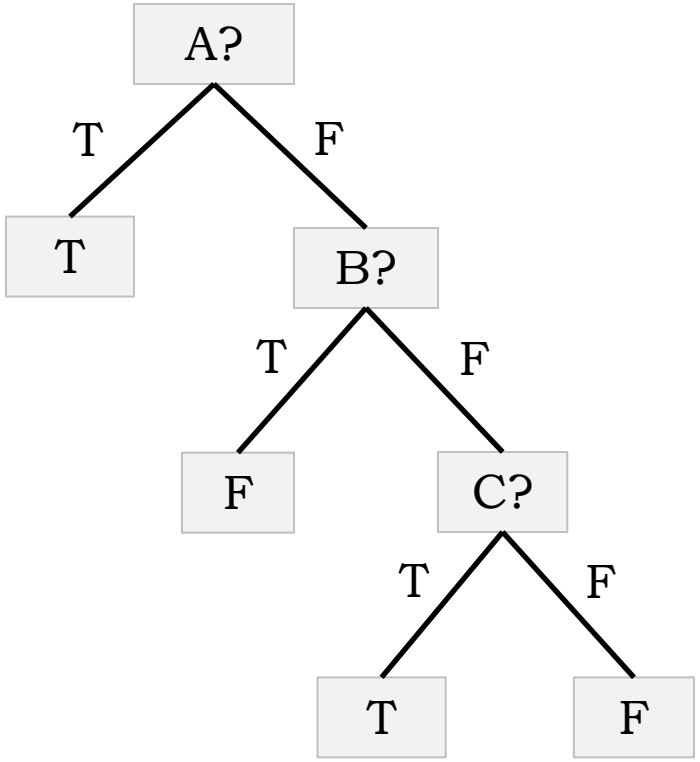
Now, we can replace the last nodes with the relevant decision Output.

Tufts

Choosing "Important" Attributes

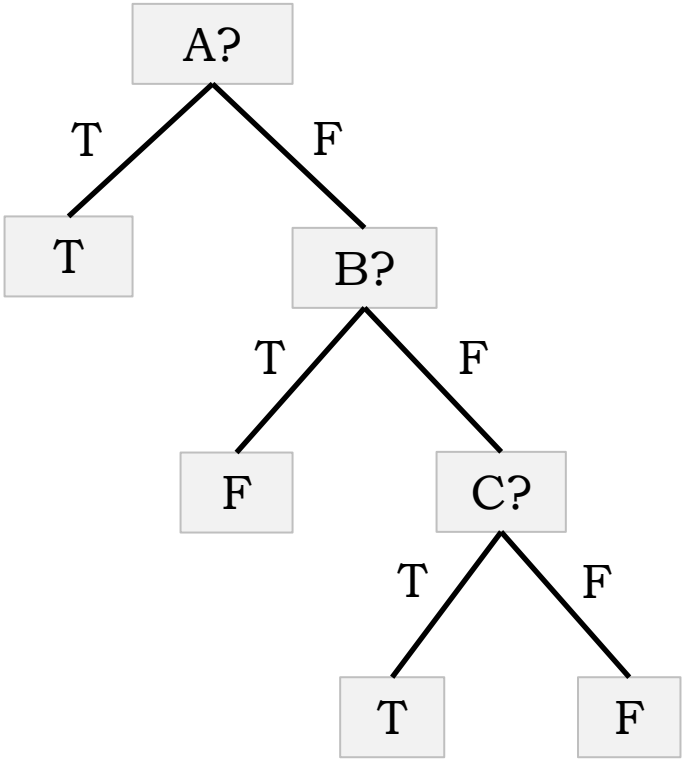
- Order {A, B, C}: the final decision tree for our data-set

Case	A	B	C	Output
1	T	F	F	T
2	F	T	T	F
3	T	T	F	T
4	F	F	T	T
5	F	F	F	F
6	F	T	F	F

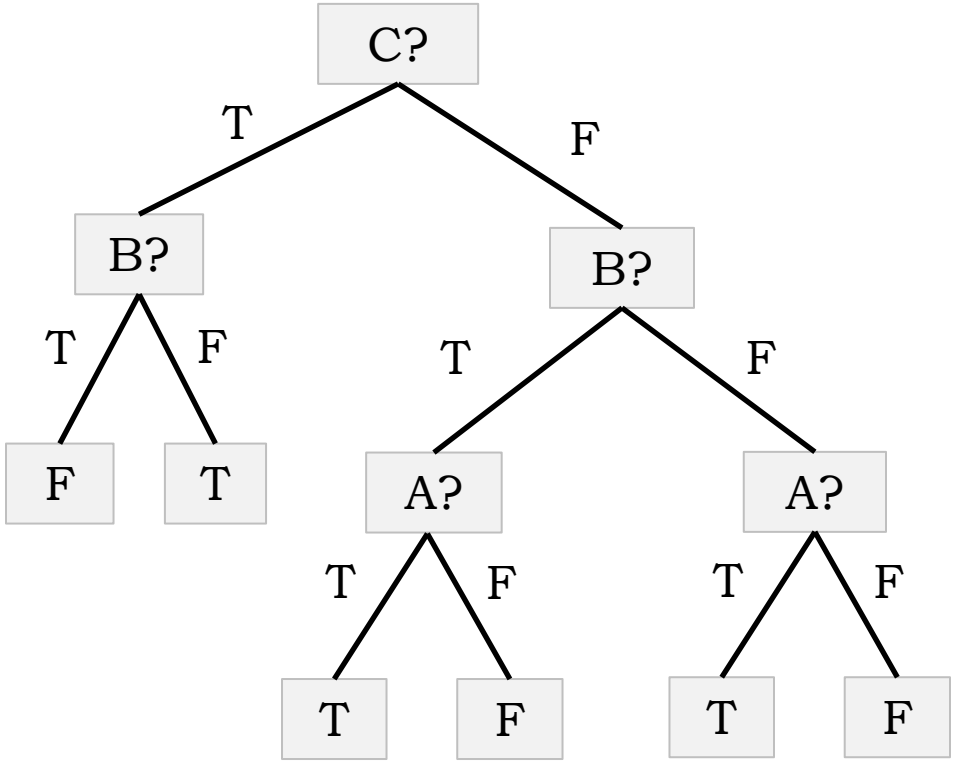


Choosing "Important" Attributes

- If we reverse the order of attributes and do the same process, we get a different, somewhat larger tree (although both will give same decision results on our set)



{A, B, C}



{C, B, A}

Tufts

Choosing "Important" Attributes

- The reading from Daumé suggests one test for importance based on a simple ***counting*** method
- Consider each remaining attribute:
  1. Divide data-set according to possible values of that attribute
  2. For each subset, assign all data to the majority category
  3. Count how many total correct you would get this way
- Let's examine another, more sophisticated approach, using information theory

Information Theory

- Claude Shannon created information theory in his 1948 paper, “A mathematical theory of communication”
- A theory of the amount of information that communication channels can carry
- It has implications in networks, encryption, compression, and many other areas
- Also, the source of the term “bit” (credited to John Tukey)

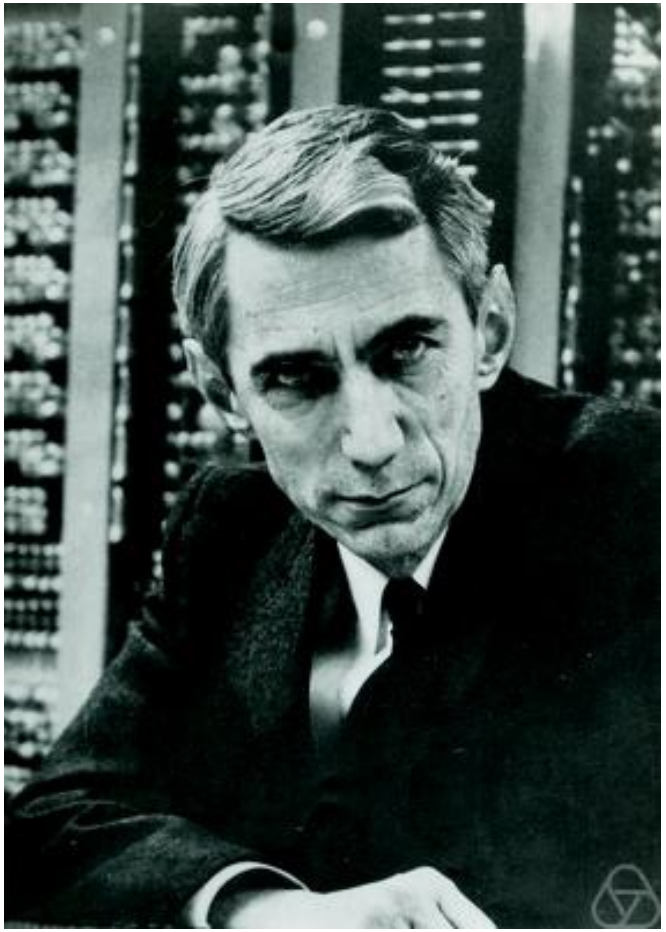


Photo source: Konrad Jacobs
(https://opc.mfo.de/detail?photo_id=3807; license)

Information Carried by Events

- Information is relative to our ***uncertainty*** about an event
  - If we ***do not know*** whether an event has happened or not, then learning that fact is a ***gain*** in information
  - If we ***already know*** this fact, then there is ***no information*** gained when we see the outcome
- Thus, if we have a fixed coin that always comes up tails, actually flipping it tells us ***nothing*** we don't already know
- Flipping a fair coin ***does*** tell us something, on the other hand, since we can't predict the outcome ahead of time

Amount of Information

- From N. Abramson (1963): If an event e_i occurs with probability p_i, the amount of information carried is:

$$I(e_i) = \log_2 \frac{1}{p_i}$$

- (The base of the logarithm doesn't matter, but if we use base-2, we are measuring information in bits)

- Thus, if we flip a fair coin, and it comes up tails, we have gained information equal to:

$$I(Tails) = \log_2 \frac{1}{P(Tails)} = \log_2 \frac{1}{0.5} = \log_2 2 = 1.0$$

Biased Data Carries Less Information

- While flipping fair coin yields 1.0 bit of information, flipping one that is biased gives us ***less***
- If we have a ***somewhat*** biased coin, then we get:

$$\mathcal{E} = \{Heads, Tails\}$$
$$\mathcal{P}_2 = \{0.25, 0.75\}$$
$$I(Tails) = \log_2 \frac{1}{P(Tails)} = \log_2 \frac{1}{0.75} = \log_2 1.33 \approx 0.415$$

- If we have a ***totally*** biased coin, then we get:

$$\mathcal{P}_3 = \{0.0, 1.0\}$$
$$I(Tails) = \log_2 \frac{1}{P(Tails)} = \log_2 \frac{1}{1.0} = \log_2 1.0 = 0.0$$

Entropy: Total Average Information

- Shannon defined the entropy of a probability distribution as the ***average amount*** of information carried by events:

$$\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$$

$$H(\mathcal{P}) = \sum_i p_i \log_2 \frac{1}{p_i} = -\sum_i p_i \log_2 p_i$$

- This can be thought of in a variety of ways, including:
  - How much ***uncertainty*** do we have about the average event
  - How much ***information*** do we get when an average event occurs
  - How many bits, on average, are needed to ***communicate*** about the events (Shannon was interested in finding the most efficient overall encodings to use in transmitting information)

Entropy: Total Average Information

- For a coin, *C*, the formula for entropy becomes:

$$H(C) = -(P(Heads)\log_2 P(Heads) + P(Tails)\log_2 P(Tails))$$

- A fair coin, $\{0.5, 0.5\}$, has ***maximum*** entropy:

$$H(C) = -(0.5\log_2 0.5 + 0.5\log_2 0.5) = 1.0$$

- A somewhat biased coin, $\{0.25, 0.75\}$, has ***less***:

$$H(C) = -(0.25\log_2 0.25 + 0.75\log_2 0.75) \approx 0.81$$

- And a fixed coin, $\{0.0, 1.0\}$, has ***none***:

$$H(C) = -(1.0\log_2 1.0 + 0.0\log_2 0.0) = 0.0$$

A Mathematical Definition

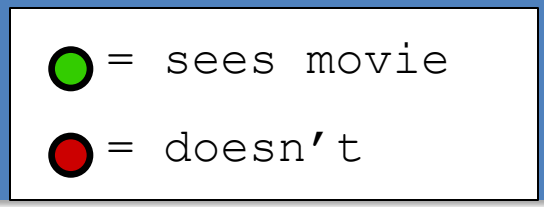
$$H(\mathcal{P}) = -\sum_i p_i \log_2 p_i$$

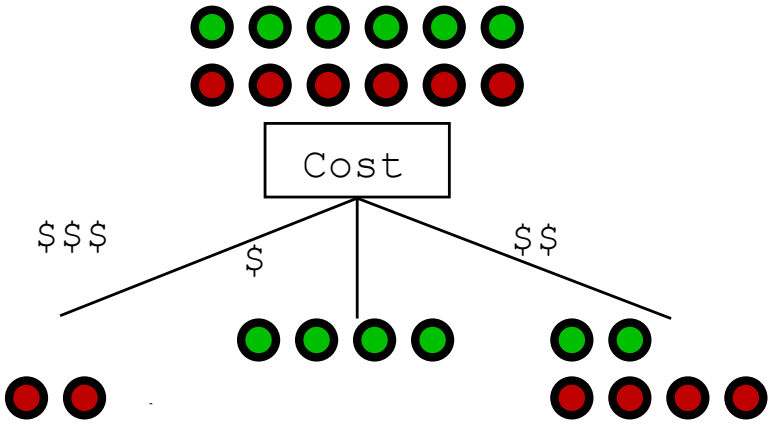
- It is easy to show that for any distribution, entropy is always greater than or equal to 0 (***never negative***)
- ***Maximum*** entropy occurs with a uniform distribution
  - In this distribution, if we have k possible outcomes, the probability of each is the same: $p_i = 1/k$
  - In such cases, entropy (in bits) is $\log_2 k$
- Thus, for any distribution possible, we have:

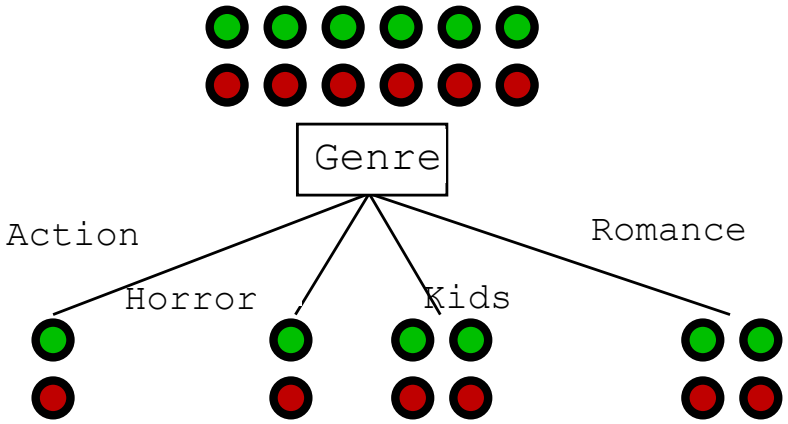
$$\mathcal{P} = \{p_1, p_2, \ldots, p_k\}$$

$$0 \leq H(\mathcal{P}) \leq \log_2 k$$

Back to Decision Trees: An Information-Theoretic Approach







- Intuitively, a good choice of attribute gives us the ***most information*** about how output decisions are made.
  - Ideally, it would divide our outputs perfectly, telling us everything we needed to know to decide.
  - Often, a single attribute only tells us part of what we need to know, so we prefer those that tell us the most.
  - In this example, `Cost` gives us more information than `Type` since some values of the first attribute predict the decision ***perfectly***, while no values of the second do the same.

Entropy for Decision Trees

- For a binary (yes/no) decision problem, we can treat a training set with p positive examples and n negative examples as if it were a random variable with two values and probabilities:

$$P(Pos) = \frac{p}{p+n} \qquad P(Neg) = \frac{n}{p+n}$$

- We can then use the definition of entropy to measure the information gained by finding out whether an example is positive or negative:

$$H(Examples) = -(P(Pos)\log_2 P(Pos) + P(Neg)\log_2 P(Neg))$$

$$= -(\frac{p}{p+n}\log_2\frac{p}{p+n} + \frac{n}{p+n}\log_2\frac{n}{p+n})$$

Information Gain

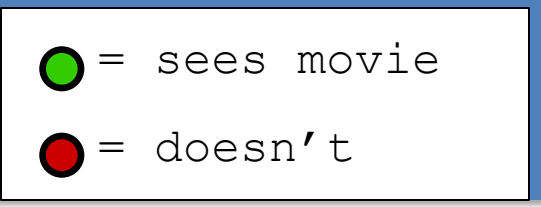
- When we choose an attribute A with d values, we divide our training set into sub-sets $E_1, \ldots, E_d$
  - Each set E_k has its number of positive and negative examples, p_k and n_k, and entropy $H(E_k)$
- The total ***remaining entropy*** after dividing on A is thus:

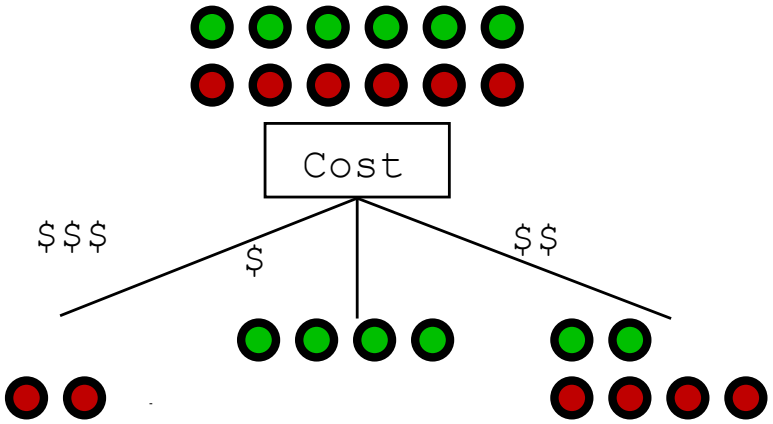
$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p + n} H(E_k)$$

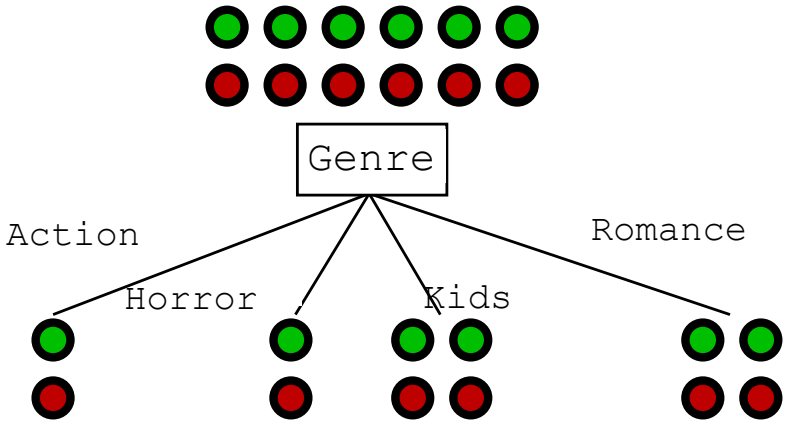
- And the total information gain (entropy reduction) if we do choose to use *A* as the dividing-branch variable is:

$$Gain(A) = H(Examples) - Remainder(A)$$

Choosing Variables Using the Information Gain







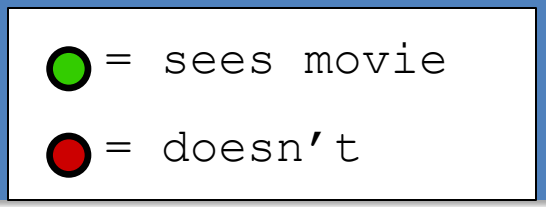
- Now we can be precise about how `Cost` gives us more information than `Genre`:

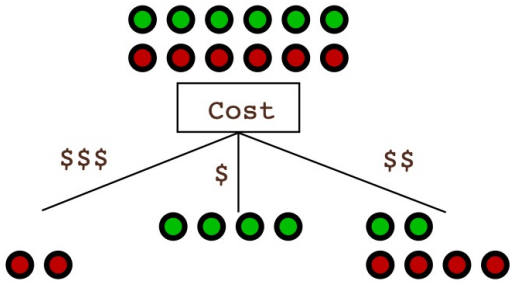
$$H(Examples) = -(\frac{6}{12}\log_2\frac{6}{12} + \frac{6}{12}\log_2\frac{6}{12})$$

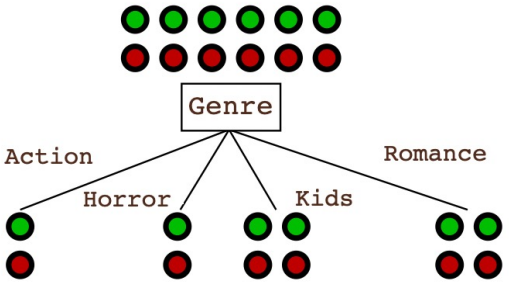
$$= -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2})$$

$$= -(-\frac{1}{2} + -\frac{1}{2}) = 1.0$$

Choosing Variables Using the Information Gain







- Now we can be precise about how `Cost` gives us more information than `Genre`:

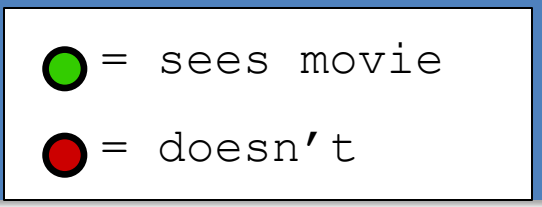
$$Gain(Cost) = H(Examples) - Remainder(Cost)$$
$$= 1.0 - (\frac{2}{12}H(E_1) + \frac{4}{12}H(E_2) + \frac{6}{12}H(E_3))$$

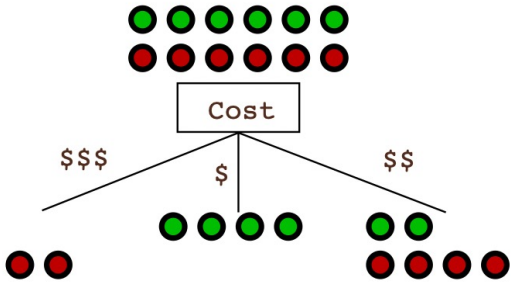
Thus, since we have:

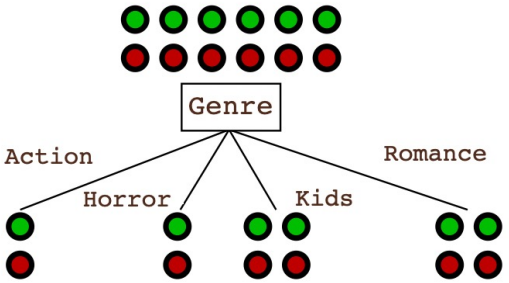
$$H(E_1) = -(\frac{0}{2}\log_2\frac{0}{2} + \frac{2}{2}\log_2\frac{2}{2}) = 0$$
$$H(E_2) = -(\frac{4}{4}\log_2\frac{4}{4} + \frac{0}{4}\log_2\frac{0}{4}) = 0$$
$$H(E_3) = -(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6}) \approx 0.918$$

$$Gain(Cost) = 1.0 - \frac{0.918}{2} = 0.541$$

Choosing Variables Using the Information Gain







- Now we can be precise about how `Cost` gives us more information than `Genre`:

$$Gain(Genre) = H(Examples) - Remainder(Genre)$$
$$= 1.0 - (\frac{2}{12}H(E_1) + \frac{2}{12}H(E_2) + \frac{4}{12}H(E_3) + \frac{4}{12}H(E_4))$$

Thus, since we have:

$$H(E_1) = H(E_2) = H(E_3) = H(E_4) = 1.0$$

$$Gain(Genre) = 1.0 - 1.0 = 0$$

And so we would choose to split on `Cost`, since:

$$Gain(Cost) = 0.541 > Gain(Genre) = 0$$

Information Gain and Other Heuristics

- A couple of questions could be raised about the use of information gain to choose attributes in a tree:
  - What do we do when there is a ***tie***?
  - Are there ***other*** measures we could use instead?
- First, there is any number of ways we might break ties between attributes with the same information gain:
  - Deterministically (e.g., the first attribute we consider)
  - Non-deterministically (e.g., a "coin flip" in case of ties)
  - Based upon some other heuristic (e.g., choosing those that give us the largest number of set decisions)
- Second, it is important to note that information gain is only a measure that works in ***many cases***—that doesn't mean there might not be something else we could use in specific instances that would do better (Daumé suggests another such heuristic)