Class #03: Decision Trees

Tufts

Machine Learning (COMP 135): M. Allen, 11 Sept. 19

Review: Entropy

- 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*** we have about the average event
  - How much ***information*** 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***:

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

Review: Inductive Learning

- In its simplest form, induction is the task of learning a ***function*** on some inputs from ***examples*** of its outputs

- For a target function, f, each training example is a pair

$$(x, f(x))$$

  - We assume that we do not yet know the actual form of the function f (if we did, we don't need to learn)

- ***Learning problem***: find a hypothesis function, h, such that $h(x) = f(x)$ ***most of the time***, based on a training set of example input-output pairs

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 records for restaraunts
- These customers have made a number of decisions about whether to wait for a table, based on a number of 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 wait time in minutes (*0-10*, *10-30*, *30-60*, *>60*)
- The function we want to learn is whether or not a (future) customer will decide to wait, given some particular 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	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

Image source: Russel & Norvig, *AI: A Modern Approach* (Prentice Hal, 2010)

- 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

Decision Tree Functions

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

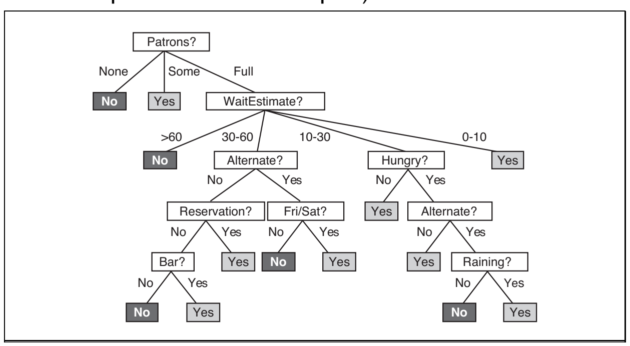
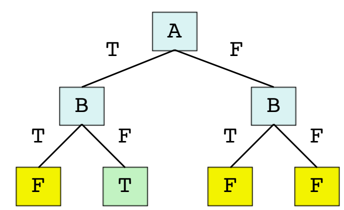


Image source: Russel & Norvig, *AI: A Modern Approach* (Prentice Hal, 2010)

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 a number of ways, but is perhaps most easily understood recursively
- The main question becomes: ***how*** do we choose the attribute A that we use to split our examples?

Decision Tree Learning Algorithm

```
function DecisionTreeTrain(data, remaining_features, parent_guess)
  guess ← most frequent label in data
  if (all labels in data same) or (remaining_features = ∅) then
    return Leaf(guess)
  else if data = ∅ then
    return Leaf(parent_guess)
  else
    F* ← MostImportant(remaining_features, data)
    Tree ←  a new decision tree with root-feature F*
    for each value f of F* do
      data_f ← {x ∈ data | x has feature-value f}
      sub_f ← DecisionTreeTrain(data_f, remaining_features − F*, guess)
      add a branch to tree with label-value f and subtree sub_f
    endfor
    return Tree
  endif
```


Base Cases

```
function DecisionTreeTrain(data, remaining_features, parent_guess)
  guess ← most frequent label in data
  if (all labels in data same) or (remaining_features = ∅) then
    return Leaf(guess)
  else if data = ∅ then
    return Leaf(parent_guess)
          ⋮
```

- The algorithm stops in three cases:

1. ***Perfect*** classification of data found: use it as a leaf-label
2. No ***features*** left: use ***most common*** class
3. No ***data*** left: use most common class of ***parent*** data

Recursive Case

```
function DecisionTreeTrain(data, remaining_features, parent_guess)
  guess ← most frequent label in data
              ⋮
    F* ← MostImportant(remaining_features, data)
    Tree ← a new decision tree with root-feature F*
    for each value f of F* do
      data_f ← {x ∈ data | x has feature-value f}
      sub_f ← DecisionTreeTrain(data_f, remaining_features − F*, guess)
      add a branch to tree with label-value f and subtree sub_f
    endfor
    return Tree
```

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:

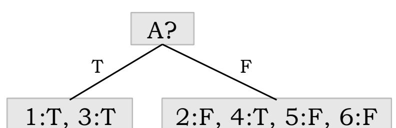
Case	A	B	C	Output
1	T	F	F	T
2	F	T	T	F
3	T	T	F	T
4	F	F	T	T
5	F	F	F	F
6	F	T	F	F

- 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.

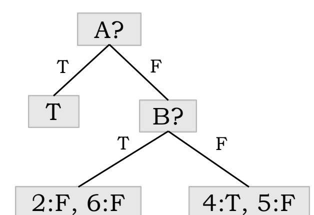
This is an example of the ***second*** base case stopping condition of the recursive algorithm.

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

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

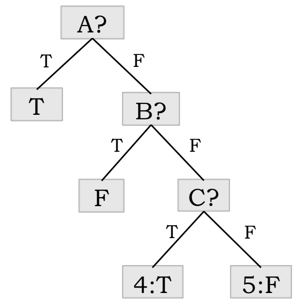


Again, all Outputs are the same on this branch.

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

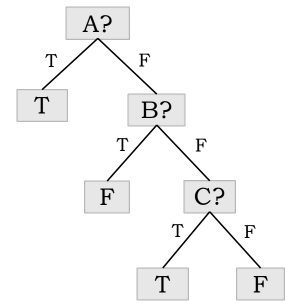


Now, we can replace the last nodes with the relevant decision Output.

Choosing "Important" Attributes

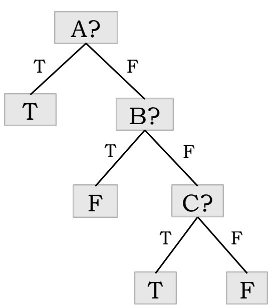
- Order {A, B, C}: the final decision tree for our data-set

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

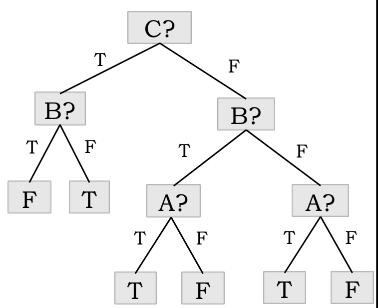


Choosing "Important" Attributes

- 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}



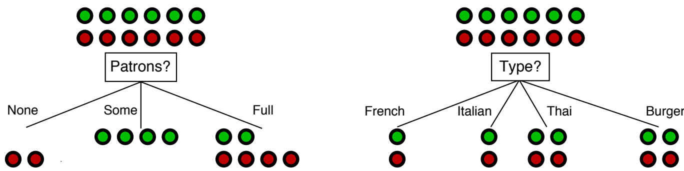
{C, B, A}

Choosing "Important" Attributes

- The Daumé text suggests one test for importance, based upon 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 the majority category
    3. Count how many total correct you would get this way
- We will examine ***another*** approach, based on information theory (you will implement ***both*** in your first program)

Another Approach

= waits
= doesn't wait



- Intuitively, a good choice of the attribute to use is one that 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 make our decision
  - Often, a single attribute only tells us part of what we need to know, so we prefer those that tell us the most
  - In the example, Patrons gives us more information than Type, since some values of the first attribute predict decision ***perfectly***, while no values of 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:

$$\begin{aligned} H(Examples) &= -(P(Pos)\log_2 P(Pos) + P(Neg)\log_2 P(Neg)) \\ &= -(\frac{p}{p+n}\log_2\frac{p}{p+n} + \frac{n}{p+n}\log_2\frac{n}{p+n}) \end{aligned}$$

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 own 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:

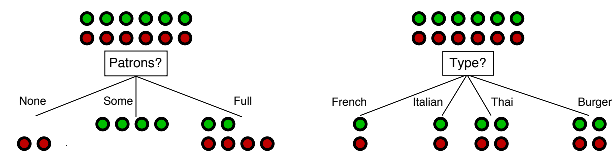
$$Remainder(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} H(E_k)$$

- 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

= waits
= doesn't wait



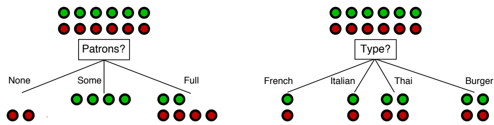
- Now we can be precise about how *Patrons* gives us more information than *Type*:

$$\begin{aligned} H(Examples) &= -(\frac{6}{12}\log_2\frac{6}{12} + \frac{6}{12}\log_2\frac{6}{12}) \\ &= -(\frac{1}{2}\log_2\frac{1}{2} + \frac{1}{2}\log_2\frac{1}{2}) \\ &= -(-\frac{1}{2} + -\frac{1}{2}) = 1.0 \end{aligned}$$

Choosing Variables Using the Information Gain

= waits

= doesn't wait



- Now we can be precise about how *Patrons* gives us more information than *Type*:

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

$$= 1.0 - (\frac{2}{12}\,H(E_1) + \frac{4}{12}\,H(E_2) + \frac{6}{12}\,H(E_3))$$

Thus, since we have:

$$H(E_1) = -(\frac{0}{2}\log_2\frac{0}{2} + \frac{2}{2}\log_2\frac{2}{2}) = 0$$

$$H(E_2) = -(\frac{4}{4}\log_2\frac{4}{4} + \frac{0}{4}\log_2\frac{0}{4}) = 0$$

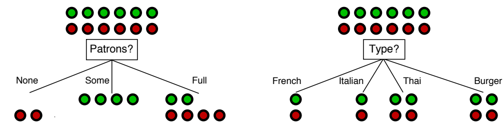
$$H(E_3) = -(\frac{2}{6}\log_2\frac{2}{6} + \frac{4}{6}\log_2\frac{4}{6}) \approx 0.918$$

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

Choosing Variables Using the Information Gain

= waits

= doesn't wait



- Now we can be precise about how *Patrons* gives us more information than *Type*:

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

$$= 1.0 - (\frac{2}{12}\,H(E_1) + \frac{2}{12}\,H(E_2) + \frac{4}{12}\,H(E_3) + \frac{4}{12}\,H(E_4))$$

Thus, since we have:

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

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

And so we would choose to split on $Patrons$, since:

$$Gain(Patrons) = 0.541 \;>\; Gain(Type) = 0$$

Learning with Information Gain

- If we use this information gain concept of information to rate the IMPORTANCE of an attribute, and always split based on the one that gives us the greatest gain, we can learn the following, more compact tree for the restaurant example:

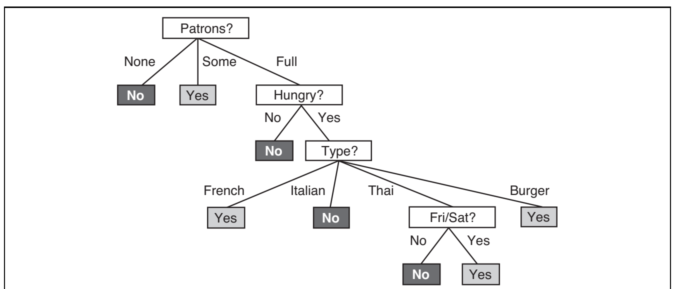


Image source: Russel & Norvig, *AI: A Modern Approach* (Prentice Hal, 2010)

Performance of Learning

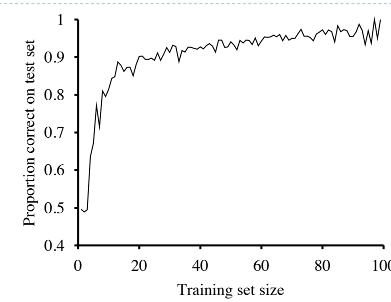


Image source: Russel & Norvig, *AI: A Modern Approach* (Prentice Hal, 2010)

- If we start with a set of 100 random examples of the restaurant problem, we can see that the accuracy of the learning increases relative to the size of the training set

Information Gain and Other Heuristics

- A couple 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?
- For the first, there are any number of ways we might break ties between attributes with the same information gain:
  - Deterministically (e.g., 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)
- For the 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 actually do better (indeed, Daumé suggests another such heuristic)

This Week

- Information Theory & Decision Trees
  - Some material in these slides drawn from Russel & Norvig, *Artificial Intelligence: A Modern Approach* (Prentice Hal, 2010)
- Readings:
  - Blog post on Information Theory (linked from class schedule)
  - Chapter 1 of the Daumé text (linked from class schedule)
- Office Hours: 237 Halligan
  - Tuesday, 11:00 AM – 1:00 PM