<u>Decision Trees</u>

Improving components of an agent by machine learning

- The components include
  - Converting from state conditions to actions
  - Inferring information on the state from percepts
  - Knowledge of what effects actions have
  - Knowledge of the desirability of states
  - etc
- Assume no prior knowledge, everything has to be learned
- Induction: from observations develop a general rule
  - can be wrong
- Not deduction which can't be wrong
- The aim is to discover a function that
  - maps inputs (percepts of the state of the world)
  - to an output: a fact that we would like to know,
    - or an action we should take, or similar
- If the output is discrete, this is Classification
- If the output is continuous, it is called Regression

Supervised learning

- Provided with samples of inputs
  - pictures perhaps
- Together with correct Labels or classifications for them
  - such as “kitten”, “puppy”, “stop sign”
- The Training Set
- Discover a function that will correctly map
  - inputs that have not been seen before
  - to their correct labels or classifications

Unsupervised learning

- Given many samples of inputs
- Maybe detects Clusters: subsets of the inputs that share features
- Perhaps from many images it may notice a cluster that happens to be cats
  - but it would not know that they are cats,
  - just that there is this unnamed phenomenon in the world

Reinforcement learning

- A bit like the way babies learn
- Start off acting almost at random
- Receive awards or punishments based on the effects of those actions
- Learn which actions are most responsible

Back to supervised learning

- Given a training set of input-output pairs
  - $(x_1, y_1), (x_2, y_2), (x_3, y_3), (x_4, y_4), \ldots, (x_N, y_N)$
- they actually come from some unknown function $y_i = f(x_i)$
- Try to discover another function, h, that approximates f.

- That function, h, is called a Hypothesis about the world
- Or a Model of the data
- Underfitting: h does not match the data properly
- Overfitting: h is too specifically fitted to the training data
  - It gets the test inputs right, but not new real-world inputs
  - e.g. for any 12-point numeric data set,
    - you can always find an 11^{th} order polynomial that fits perfectly
    - but it will have wild swings in it
  - A simple straight line might not fit the training set so well
    - but do a better job in the real world - less noise

Big example: Should you wait for a table at a restaurant?

- The result of this function is just yes or no.
- The inputs come from 10 discrete percepts:
  - Alternate: is there an alternative near-by?
  - Bar: is there a nice bar you could wait in?
  - FriSat: true on Fridays and Saturdays
  - Hungry: are we really hungry now?
  - Patrons: none, some, or full.
  - Price: cheap, middling, expensive
  - Raining: is it raining?
  - Reservation: have we got a reservation?
  - Type: french, italian, thai, burger
  - Time: greeter's estimate of the wait time: 0-10, 10-30, 30-60, >60
- Training set is observed results of a person's actual decisions:

	Alt	Bar	FS	Hung	Patr	Price	Rain	Res	Type	Time	Output
1	yes	no	no	yes	some	exp	no	yes	fre	0-10	yes
2	yes	no	no	yes	full	che	no	no	thai	30-60	no
3	no	yes	no	no	some	che	no	no	bur	0-10	yes
4	yes	no	yes	yes	full	che	yes	no	thai	10-30	yes
5	yes	no	yes	no	full	exp	no	yes	fre	>60	no
6	no	yes	no	yes	some	mid	yes	yes	ital	0-10	yes
7	no	yes	no	no	none	che	yes	no	bur	0-10	no
8	no	no	no	yes	some	mid	yes	yes	thai	0-10	yes
9	no	yes	yes	no	full	che	yes	no	bur	>60	no
10	yes	yes	yes	yes	full	exp	no	yes	ital	10-30	no
11	no	no	no	no	none	che	no	no	thai	0-10	no
12	yes	yes	yes	yes	full	che	no	no	bur	30-60	yes

Decision trees

- (example diagram: Patr, Time, Alt, Hung, Res, Fri, Alt, Bar, Rain)
- Sometimes an expert can just give you a decision tree
  - But experts in one field are not necessarily experts in logic
  - A Knowledge Engineer has to conduct Knowledge Elicitation
  - But either way, machine learning is not involved here
- A particular sub-kind of supervised learning
  - You get all the data all at once
  - Taking into account new training items can be expensive

- Some alternatives:
    - Type first would be bad
    - But Patrons first does much better
- Learning Curve: num of training examples vs proportion correct
- How do you learn the tree from the training set?
- A seemingly simple method
    - All remaining examples yes (or all no), then done.
    - Some yes and some no: find best attribute to split them
        - and continue with the split sets of examples
    - No examples left: incomplete information
        - This combination of attrs has never been observed
        - Pick parent's most common output value
    - Some examples but no attrs left:
        - These examples have same descrs but different labels
        - Error or Noise in the data, nondeterministic, or unobservable
- But the second possibility leaves a lot unsaid. How do we find the best?

Entropy

- From Information Theory, not chemistry, sort of similar
- A measure of uncertainty
- Always called H
- Measured in bits
- Variable has only one possible value, a very very unfair coin?
    - Entropy is zero - no uncertainty
    - You learn nothing from seeing the actual result
- Two equally likely values, a fair coin
    - From the result you learn yes or no
    - Entropy is one bit
- Sixteen equally likely possibilities
    - Four bits of entropy
- Two unequal possibilities, e.g. P(heads) = 0.99 and P(Tails) = 0.01
    - There is less uncertainty
    - You learn less from seeing the actual result
    - If you just guessed you would almost always be right
    - So the entropy should be very small
- For a variable V with possible values $v_1, v_2, v_3, v_4, \ldots$
    - $H(V) = \Sigma\, P(v_i) \times \log_2(1 / P(v_i))$
    - $= - \Sigma\, P(v_i) \times \log_2(P(v_i))$
- The fair coin
    - $H(Toss) = - (0.5 \times \log_2 0.5 + 0.5 \times \log_2 0.5) = 1.0$
- The very unfair coin
    - $H(Toss) = - (0.99 \times \log_2 0.99 + 0.01 \times \log_2 0.01) = 0.08$
- Define B(p), entropy of a boolean variable V with P(true) = p
    - $B(V) = - (p \times \log_2 p + (1-p) \times \log_2 (1-p))$
- A training set with Y yes results and N no results
    - $H(Output) = B(\, Y / (Y + N)\,)$
- The result of a test (e.g. what is the value of Price?)
    - gives us some information

so it reduces our uncertainty of the overall output
so it reduces the entropy

- Going with the example table, call the training examples E_1, E_2, …, E_{12}
  Initially our set of examples is all of them, S = { E_1, E_2, …, E_{12} }
  If all examples have the same outcome, then all done
  P = number of times in S that result is yes = 6
  N = number of times it is false = 6
  For each of the possible attributes A (Alternate, Bar, etc)
    For each possible value of that attribute V (yes, no, etc)
      P_V = number of examples where A has value V
        and outcome is yes
      N_V = number of examples where A has value V
        and outcome is no
    so $P_V + N_V$ is the number of times A has value V
    and P + N is the total number of examples
    $P_{A,V} = (P_V + N_V) / (P + N)$ is the probability that A has value V
    $P_V / (P_V + N_V)$ is the prob that A being V leads to yes outcome
    so $B(P_V / (P_V + N_V))$ is entropy left after finding that A equals V
    so $P_{A,V} \times$ that is entropy in a particular case
      weighted by probability of that case happening
    Add up all those weighted entropies to find the total entropy
      that would remain after discovering the value of A
    The entropy we had before all of this was $B(P / (N + P))$
    so the entropy lost (therefore information gained) by asking
      the question A is that original entropy minus the sum
  Pick the attribute that gave the highest information gain
  Let's say that attribute has possible values V_1, V_2, …, V_N,
  Split S into subsets of the examples S_1, S_2, …, S_N, where S_i is the
    subset of examples where the chosen attribute equals V_i
  Analyse each of those subsets in exactly the same way.
- In practice
  So the *expected* entropy remaining after testing A is
    $\text{Remainder}(A) = \Sigma\, ((P_V + N_V) / (P + N)) \times B(P_V / (P_V+N_V))$
  The Information Gain is the loss in entropy
    $\text{Gain}(A) = B(Y / (Y + N)) - \text{Remainder}(A)$
  Example: Gain(Patrons) =
    $1 - ((2/12)B(0/2) + (4/12)B(4/4) + (6/12)B(2/6))$
    $= 0.541$
  Example: Gain(Type) =
    $1 - ((2/12)B(1/2) + (2/12)B(1/2) + (4/12)B(2/4) + (4/12)B(2/4))$
    $= 0$
  The *Best* attribute to test when building a decision tree
    is the one with the highest information gain.

Possible problems with decision trees

- Might overfit
- Might make irrelevant tests, making it unnecessarily big
- Pruning - remove irrelevant tests - Statistical methods - Beyond us.
- Continuous attributes (Price, Time, etc)

    - Discretise - like we did
    - Split Point test - e.g. Time > 27
- Not continuous but still too many values (e.g. Zip code)
    - Information Gain Ratio - Beyond us
- Continuous output value
    - Regression Tree
    - Decisions based on linear function of some attributes
- What if two training set examples are contradictory?