Lecture 9: (A peak at) Machine Learning

Simon Parsons

Department of Informatics
King's College London

(Version 1.2)

Today

- Introduction
- Probabilistic Reasoning I
- Probabilistic Reasoning II
- Sequential Decision Making
- Game Theory
- Probabilistic Reasoning over Time
- Argumentation I
- Argumentation II
- (A peek at) Machine Learning
- AI & Ethics

What is machine learning?

- No agreed definition.
- Ideas have changed widely over time
- Techniques have changed widely over time.
  - For example AIMA on neural networks.
- Broadly speaking: how to use data on past situations to learn how to act in the future.

Three broad classes of machine learning

- *Supervised learning*
  - Correct answers for each instance.
  - Use these to be able to identify correct answer on new instances.
- *Unsupervised learning*
  - No correct answers available.
  - Identify patterns/relations.
- *Reinforcement learning*
  - Occasional rewards
  - Need to associate actions with the rewards they bring.

What we will look at in learning

- *Classification*
  - Supervised learning
- *Clustering*
  - Unsupervised learning
- *Reinforcement learning*
  - Simple RL model.

Supervised Learning

- Given a *training set*:

$$(x_1, y_1), (x_2, y_2), \ldots (x_N, y_N)$$

  where:

$$y_i = f(x_i)$$

  discover a function:

$$h(x) \approx f(x)$$

- x can be any value, or set of values.

Supervised Learning

- When y is a a member of a finite set, this is *classification*
- When y is a number, this is *regression*.

Supervised learning

- A set of examples/instances.
- Examples described by attribute values
    - Boolean,
    - discrete,
    - continuous,
    - ...
- Classification of examples is positive (T) or negative (F)
- Aim is to learn a function from examples to classification.

Supervised learning

(Christophe Gevrey)

A set of instances

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
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*

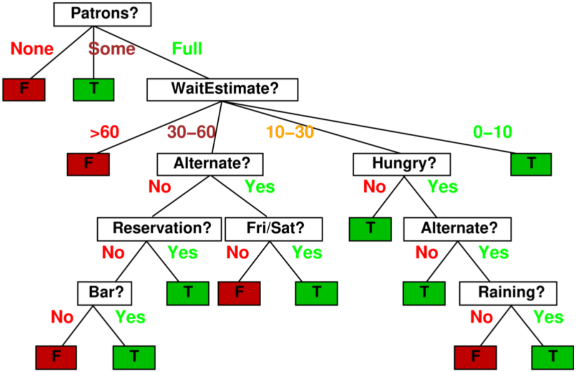
A set of instances

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
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*

What is the function?

Decision trees

- Here is the “true” tree for deciding whether to wait:

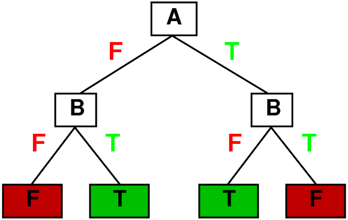


- A description of the function.

Decision trees

- Decision trees can express any function of the input attributes.
- 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



Show XOR because is hard to capture for some classifiers.

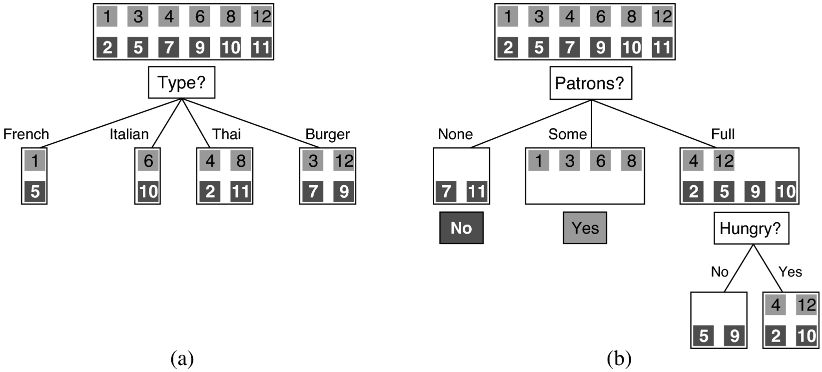
Decision trees

- Trivially, $\exists$ a consistent decision tree for any training set with one path to leaf for each example.
  - Unless there is non-determinism
- *Table lookup*
- This trivial tree probably won't generalize to new examples
- Prefer to find more compact decision trees

Decision tree learning

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

Some attributes are better than others



Decision tree learning

- Aim: find a small tree consistent with the training examples.
- Idea: (recursively) choose "most significant" attribute as root of (sub)tree.
- After we pick an attribute, one of five things can be the case.

Decision tree learning

- If all remaining examples are positive, then we are done. We can answer “yes”.
- See *Some* in the example above.

Decision tree learning

- If all remaining examples are negative, then we are done. We can answer “no”.
- See *None* in the example above.

Decision tree learning

- If there are some positive and some negative examples, then choose the best attribute to split them.
- See *Full*.

Decision tree learning

- If there are no examples left, then there are no examples that fit this case.
- Ending up here means we have no data on the specific case we are asking about.
- Best we can do is return a default value.
- Plurality classification.
- Best guess based on the parent node.
- Could be majority
  ie Yes if most examples at the parent node are classified "Yes".
- Could be random pick, weighted by ratio of examples at parent node.

Decision tree learning

- If there are no attributes left, but there are still positive and negative examples, these examples have the same description but different classifications.
- Can be due to noise.
- Can be due to unobservability of attributes.
- Plurality classification.

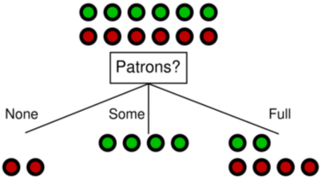
Decision tree learning

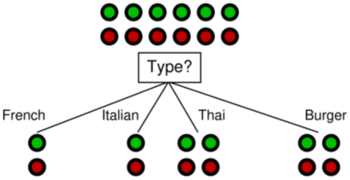
function DTL(*examples, attributes, parent_examples*) **returns** a decision tree

if *examples* is empty **then return** PLURALITY-VALUE(*parent_eamples*)
else if all *examples* have the same classification **then return** the classification
else if *attributes* is empty **then return** PLURALITY-VALUE(*examples*)
else
  best ← MOST-IMPORTANT-ATTRIBUTE(*attributes*, *examples*)
  tree ← a new decision tree with root test *best*
  for each value v_i of *best* **do**
    $examples_i \leftarrow$ {elements of *examples* with $best = v_i$}
    remain ← *attributes* − *best*
    subtree ← DTL($examples_i$, *remain*, *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—gives information about the classification

Information

- Information answers questions.
- The more clueless I am about the answer initially, the more information is contained in the answer.
- Scale: 1 bit = answer to Boolean question with prior $\langle 0.5, 0.5 \rangle$
- Information needed from an answer when prior is $\langle P_1, \ldots, P_n \rangle$:

$$H(\langle P_1, \ldots, P_n \rangle) = \sum_{i=1}^{n} -P_i \log_2 P_i$$

(also called entropy of the prior)

Calculating $\log_2$

- Your calculator probably doesn't have a $\log_2$ function.
- Instead it has log and ln.
- ln is no use here.
- log is really $\log_{10}$.
- Can compute $\log_2$ using $\log_{10}$ by:

$$log_2(x) = \frac{\log_{10}(x)}{\log_{10}(2)}$$

Information

- Suppose we have p positive and n negative examples at the root.
- Then we need:

$$H\left(\left\langle \frac{p}{p+n}, \frac{n}{p+n} \right\rangle\right)$$

  bits to classify a new example.
- For our 12 restaurant examples, $p = n = 6$ so we need 1 bit

Information

- An attribute splits the examples E into subsets E_i, each of which (we hope) needs less information to complete the classification
- Each of these subsets is a new branch.
- Let E_i have p_i positive and n_i negative examples.

$$H\left(\left\langle \frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i} \right\rangle\right)$$

bits needed to classify a new example

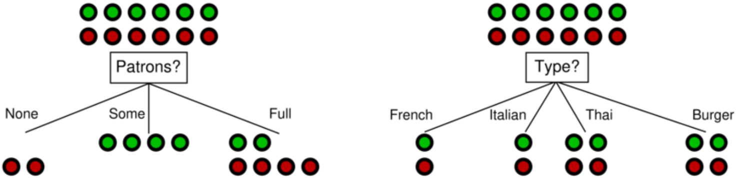
Information

- Expected number of bits per example over *all* new branches is

$$\sum_i \frac{p_i + n_i}{p + n} H\left(\left\langle \frac{p_i}{p_i + n_i}, \frac{n_i}{p_i + n_i} \right\rangle\right)$$

- Can use this value to pick “best attribute”.

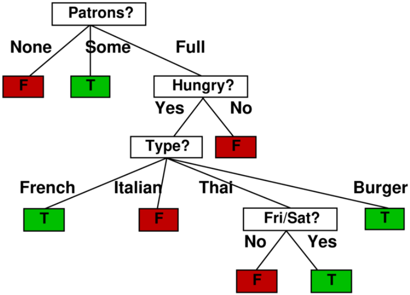
Information



- For *Patrons*?, expected value is 0.459 bits.
- For *Type*, expected value is (still) 1 bit
- Thus *Patrons*? is better than *Type*.
- In general, choose the attribute that minimizes the expected remaining information needed.

Back to the example

- Decision tree learned from the 12 examples:

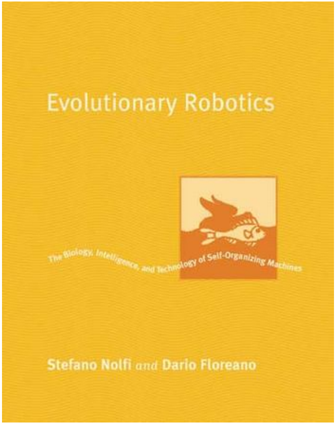


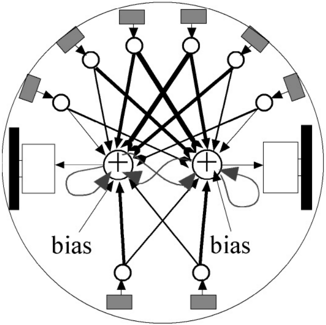
- Substantially simpler than "true" tree—a more complex hypothesis isn't justified by small amount of data

DTL one last time

- The basic algorithm given above is a version of the CART algorithm.
- Using entropy/information to make the choice about which attribute to choose gives the ID3 algorithm.
- C4.5 is similar, but adds the ability to handle unknown values, and does some post-processing to simplify the tree.

Example





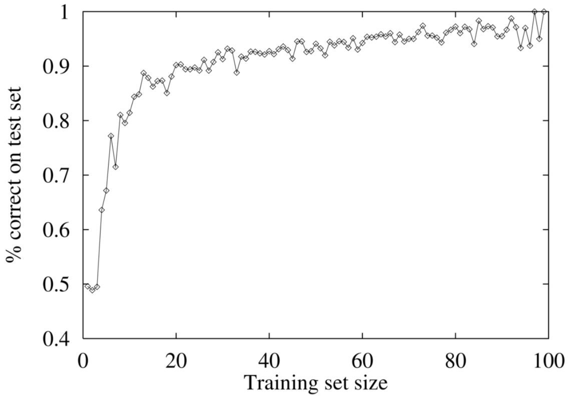
(disal.epfl.ch)

Performance measurement

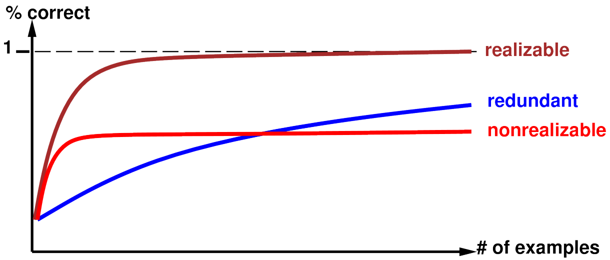
- How do we know that $h \approx f$?
- Try h on a new test set of examples
  (use same distribution over example space as training set)
- Learning curve = % correct on test set as a function of training set size

Learning curve

- Learning curve for the restaurant example.



Learning curve



- Learning curve depends on
  - realizable (can express target function) vs. non-realizable
    non-realizability can be due to missing attributes or restricted hypothesis class
  - redundant expressiveness (e.g., loads of irrelevant attributes)

Cross-validation

- What we just described for testing is holdout cross-validation.
  - Disadvantage that it doesn't use all the data.
  - However we split the data we have as training and test sets we can bias the results.
    Not enough training data or bias because the test data is small.

Cross-validation

- Better is k-fold cross validation.
- Split data into k equal subsets. Learn on $k - 1$ sets and test each result on the remainder.
  Repeat k times.
- Average test set score is a better estimate of the error rate than a single score.
- Common values of k are 5 and 10, both giving error estimates that are very likely to be accurate.

KING'S College LONDON

Cross-validation

- The extreme case is when $k = n$, the number of data points.
- Leave-one-out cross validation.

Overfitting

- One thing that cross-validation checks for is over-fitting.
- If a classifier overfits the training data, it is too specialised to that data.
- Classifies the training data very well.
- Classifies other data badly.
- A good learner does well on the data it hasn't seen.

Unsupervised Learning

- We have the same set of instances as in supervised learning.
- But we don't have the classification.
- Look for patterns in the data.
- One approach is to look for groups within the data.

  Clustering

Clustering

- Given a set of examples:

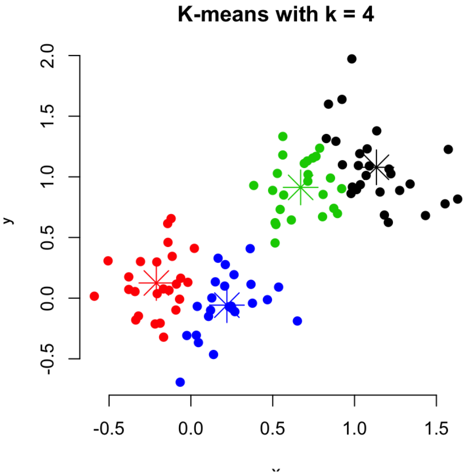
$$X = \{x_1, \dots x_N\}$$

  identify K different groups.
- Each x_i should be in the group that it matches most closely.
- Identify each group by its centre/mean, μ_j.
- The clusters z^i are then made up of the points closest to the μ_j.

KING'S College LONDON

Clustering

- For example:



(http://www.sthda.com)

K-means

- How do we find the clusters in the data?
- One approach: K-means

K-means

- Pick random values for μ_k.
- Then repeat:
  1. Assign each x_i to its closest cluster centre:

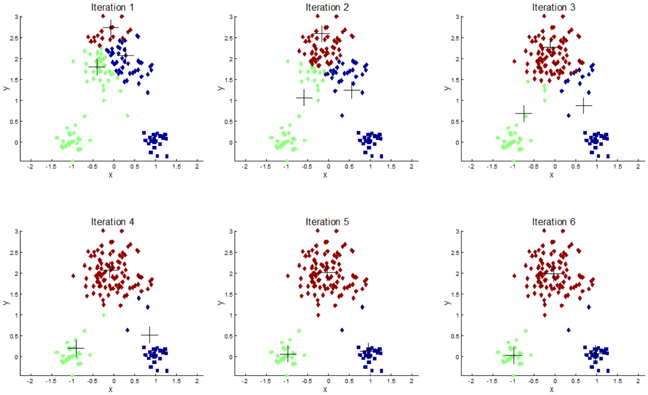
$$z^i = \arg\min_{\mu_k} distance(x_i, \mu_k)$$

  2. Update each cluster centre as the mean of the points assigned to that cluster:

$$\mu_k = \frac{1}{|\{j : z^j = k\}|} \sum_{i \in \{j : z^j = k\}} x_i$$

  until converged.
- Can use a range of metrics to define distance.

K-means

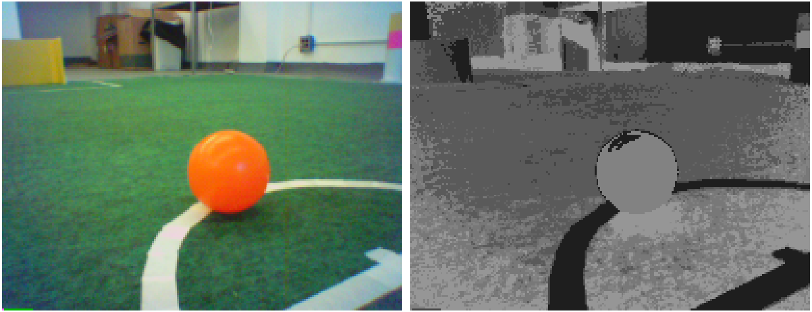


(Andrei Pandre)

K-means

- Simple.
- Often works well.
- No guarantee that clusters will converge.
- Can get poor clusterings.
- To improve:
  - Re-run with random restart.
  - Or use `kmeans++` to pick good initial cluster centres.

Example

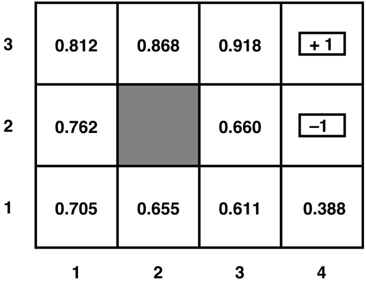


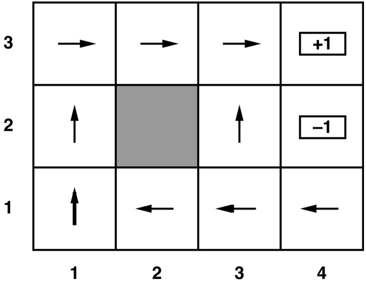
- What happens when we don't have any examples?
- Just know when we succeed or fail.
- This is the domain of reinforcement learning (RL).
- Since actions are non-deterministic, a natural framework to study this in is that of Markov decision processes.

RL in a nutshell

(Pendleton Ward/Cartoon Network)

Passive reinforcement learning





- Remember this world which we solved as an MDP?

Passive learning

- Now imagine that the agent doesn't know the transition model:

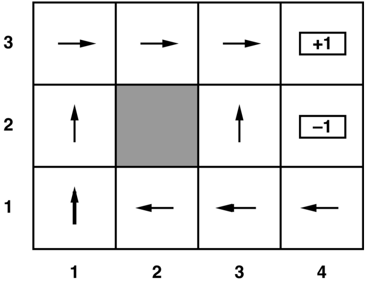
$$P(s'|s, a)$$

  and it doesn't know the reward function

$$R(s)$$

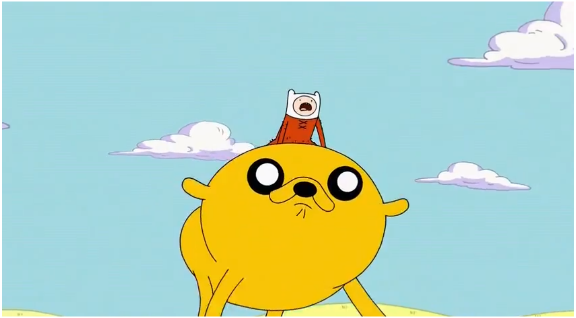
- How can it decide what to do?
- Needs to learn the transition model and reward.

Passive learning



- Agent learns utility $U^{\pi}(s)$ by carrying out runs through the environment, following some policy π.

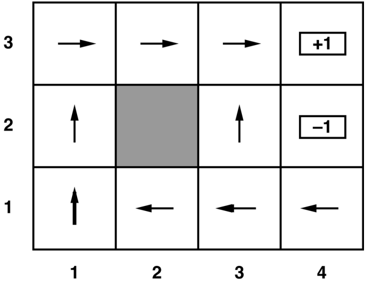
Passive learning



(Pendleton Ward/Cartoon Network)

- In passive reinforcement learning the agent's policy is fixed.
- Agent doesn't make a choice about how to act.

Passive learning



- As in Lecture 1, a run is a sequence of states and actions that continues until the agent reaches the terminal state:

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow$$
$$(1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \ldots$$

- Note the rewards attached to each state.

Passive learning

- The utility $U^{\pi}(s)$ of a state s under policy π is the expected sum of the (discounted) rewards obtained when following π.

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(S_t)\right]$$

  where S is the state reached at t from s when executing π.
- So if we run the policy for long enough, you will compute the utility of the states from the onward rewards.

Direct utility estimation

- We can estimate the utility of a state by the rewards generated along the run from that state.
- Direct utility estimation.
- Each run gives us one or more samples for the reward from a state.

Direct utility estimation

- Given the run:

$$(1,1)_{-0.04} \to (1,2)_{-0.04} \to (1,3)_{-0.04} \to (1,2)_{-0.04} \to \\ (1,3)_{-0.04} \to (2,3)_{-0.04} \to (3,3)_{-0.04} \to (4,3)_{+1}$$

  a sample reward for $(1,1)$ from the run above is the sum of the rewards all the way to a goal state.
  - 0.72 in this case.
- The same run will produce two samples for $(1,2)$ and $(1,3)$.
  - 0.76 and 0.84
  - 0.8 and 0.88
- (Here we set the discount to 1).

Passive learning

- As the agent moves it can calculate a sample estimate of $P(s'|s, \pi(s))$
- Each time it moves it creates a new sample for one state.
- Given:

$$(1,1)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow (1,3)_{-0.04} \rightarrow (1,2)_{-0.04} \rightarrow \\ (1,3)_{-0.04} \rightarrow (2,3)_{-0.04} \rightarrow (3,3)_{-0.04} \rightarrow (4,3)_{+1}$$

we get:

$$\begin{aligned} P((1,2)|(1,1), Up) &= 1 \\ P((1,2)|(1,3), Right) &= 0.5 \\ P((2,3)|(1,3), Right) &= 0.5 \\ &\vdots \end{aligned}$$

Direct utility estimation

- Over time, the agent builds up estimates of:

3	0.812	0.868	0.918	$\boxed{+1}$
2	0.762		0.660	$\boxed{-1}$
1	0.705	0.655	0.611	0.388
	1	2	3	4

and $P(s'|s, \pi(s))$, for every s, s' for the given $\pi(s)$.

Passive learning

- What does a solution look like?
- A list of states s_i.
- Each state has a utility estimate associated with it $U(s)$.
- Each state has an action associated with it, $\pi(s)$.
- Each state action pair has a probability distribution:

$$\mathbf{P}(S'|s, \pi(s))$$

  over the states S' that it gets to from s by doing $\pi(s)$.
- (May not encounter every state.)

Passive learning

- How does an agent decide what to do?
- Then the agent just computes each step using one-step lookahead on the expected value of actions.
- Picks the action a with the greatest expected utility.
- Its data on actions will be limited because it has only been trying $\pi(s)$.

Passive learning

- Has to vary π if it wants to learn the full space.
- But is this worth it?
- After all, once we have an idea of how to act to get to the goal, is more learning justified?
- Tradeoff *exploration* and *exploitation*

Tradeoff

Tradeoff

- But explore less over time.

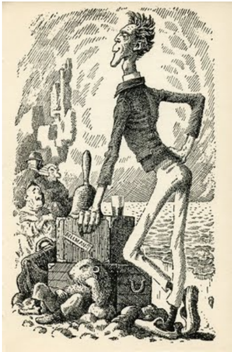
Problem with direct utility estimation

- Treats utilities of states as independent.
- But we know that they are connected — obey the Bellman equation.
- Ignoring the connection means that learning may converge slowly.
- So another approach to utility estimation: adaptive dynamic programming.
- Still doing passive reinforcement learning.

Adaptive dynamic programming

- We can improve on direct utility estimation by remembering the Bellman equation for a fixed policy:

$$U^{\pi}(s) = R(s) + \gamma \sum_{s'} P(s'|s, \pi(s))U^{\pi}(s')$$



(Mervyn Peake)

- The utility of a state is the reward for being in that state plus the expected discounted reward of being in the next state.
- This is the formula from page 76 of the notes for Lecture 4.

Adaptive dynamic programming

- Still passive learning so we have π.
- Since we are using the fixed policy version of the Bellman equation we don't have the $\max$ that makes the original so hard to solve.
- Can just plug results into an LP solver
  - As we discussed when talking about policy iteration.
- Updates all the utilities of all the states where we have experienced the transitions.
- Updated values are estimates, and are no better than the estimated values of utility and probability.

Adaptive dynamic programming

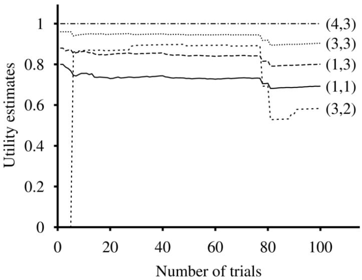
- Can also use value iteration to update the utilities we have for each state.
- Update using:

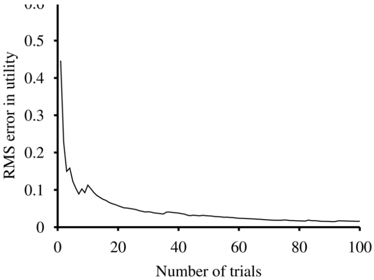
$$U_{i+1}(s) \leftarrow R(s) + \gamma \sum_{s'} P(s'|s, \pi(s)) U_i(s')$$

  to update utilities.
- Recall that we do this in modified policy iteration also.
- Back in Lecture 4 we called this “approximate policy evaluation”.

Adaptive dynamic programming

- Results:





Adaptive dynamic programming

- Still passive learning, so a solution is as before:
- A list of states s_i.
- Each state has a utility estimate associated with it $U(s)$.
- Each state has an action associated with it, $\pi(s)$.
- Each state action pair has a probability distribution:

$$\mathbf{P}(S'|s, \pi(s))$$

over the states S' that it gets to from s by doing $\pi(s)$.

After learning

- Now, to get the utilities, the agent started with a fixed policy, so it always knew what action to take.
- It used this to get utilities.
- Having gotten the utilities, it could use them to choose actions.
  - Just picks the action with the best expected utility in a given state.
- However, there is a problem with doing this.

Problems

- Might not yet have experienced the bad effects of an action:

(Calista Condo/South Jersey Times)

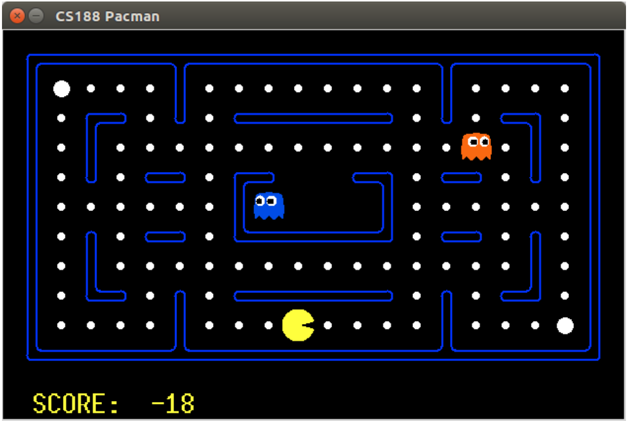
- Maybe your autonomous car learnt that running a red light saves time.

KING'S College LONDON

Problems

- Of course, this kind of over-reliance on not-full-explored state/action spaces is what people do all the time.
- There is no way to be sure that the action your reinforcement learner is picking doesn't have possible bad outcomes.
- $\Rightarrow$ Lots more work on reinforcement learning.
- (But we don't have time to look at it.)

RL Example



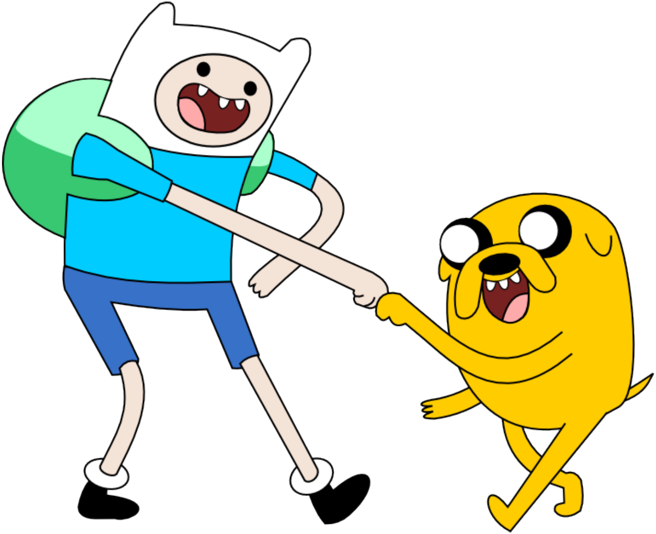
(ai.berkeley.edu)

RL Example

http://www.cs.utexas.edu/~pstone/research.shtml

- Learning robot gait.

Mathematical!



(Pendleton Ward/Cartoon Network)

Summary

- Supervised learning: find a simple function/hypothesis approximately consistent with training examples
  Decision trees
  - Also discussed crossvalidation as a way to measure performance.
- Unsupervised learning.
  K-means clustering
- Reinforcement Learning: learn rewards and transition probabilities.
  Direct utility estimation
  Adaptive dynamic programming