Machine Learning

- Learning takes place when a system modifies its knowledge with the goal of performing better in future
- An essential capability for 'generally intelligent agents':
  - we can program in only so much knowledge
  - this knowledge is inevitably partial in a complex, changing world
  - for autonomy, learning is essential

Machine Learning in Industrial Systems

- Knowledge acquisition for, e.g., diagnosis, fault-finding, design systems
  - acquisition bottleneck: experts may disagree and be unable to articulate their knowledge
  - maintenance problem: knock-on effects of changes
- Possible solutions:
  - tool support (to check syntax, consistency, etc.) — helps a little
  - natural language understanding (to elicit knowledge from dialogue/text) — not advanced enough
  - learning, e.g. from example diagnoses
- Machine learning is also being used in other ways, e.g. data mining, scientific discovery

The training set

- *Training set:* The learner will obtain some data related to what it is trying to learn
  - *Supervised learning:* training set contains examples of inputs and their corresponding outputs (outputs, e.g., from a 'teacher')
  - *Reinforcement learning:* training set contains example states and actions along with some reward or other evaluation for the action, but it does not indicate the correct action
  - *Unsupervised learning:* the training set is just inputs with no hint at all of the correct output; e.g. you're given lots of data and you try to find relationships/ clusters in the data
- Beware: the training set may be *noisy*!

A (Rough) Logical Formulation of the Task

- Suppose you've some *background knowledge*, B
- Suppose you've some *examples*, E — usually ground facts; some positive, E^+; others negative, E^-
- The learning task is to find some hypothesis, H:

$$B \wedge H \models E^+$$

$$B \wedge H \not\models E^-$$

Continued

What do we require of H?

- Require that B, E and H aren't contradictory

$$B \wedge H \wedge E \not\models \Box$$

- In *inductive learning*, also require that the learned knowledge doesn't already follow from the existing knowledge

$$B \not\models H$$

  H is 'new' knowledge and, without it, the examples wouldn't follow from the existing knowledge

$$B \not\models E^+$$

Inductive learning

- Suppose $B = \{$bird(tweety), bird(chirpie), bird(croakie), giraffe(georgie), giraffe(gertie)$\}$

- Suppose $E = \{$flies(tweety), flies(chirpie), $\neg$flies(georgie)$\}$

- Observe that $B \not\models E^+$. What should H be? We want something so that $B \wedge H \models E^+$

- The computationally easiest proposal is

  $H = E = \{$flies(tweety), flies(chirpie), $\neg$flies(georgie)$\}$

- Yet this seems to miss the point about learning: learning usually isn't simply memorisation

- (Give an example of human learning where $H = E$)

More on induction

- H must explain the examples in the training set, the *seen examples* ($B \wedge H \models E^+$)

- But H must also make good predictions on *unseen examples*; learning involves generalisation

- Induction is usually *unsound* inference
  Just because the last 100,000 birds we saw could fly doesn't prove the hypothesis that *all* birds fly; it doesn't prove that the next one we see will fly; it's just good evidence.

- (On occasion, induction *is* sound. When?)

The Hypothesis Space

- H must be chosen from among many candidates

- The set of all hypotheses from which the learner can choose is the *hypothesis space*

- Hypothesis spaces are huge, even by AI standards

- Choosing H so that $B \wedge H \models E^+$ is then only one criterion for the choice

- H must also generalise, without overgeneralising; it must make (mostly) accurate predictions on *unseen examples* (ones not in the training set)

- $H = E$ doesn't generalise at all;
  $\forall x$ flies(x) overgeneralises;
  $\forall x($bird$(x) \rightarrow$ flies$(x))$ is just right

The Way Machine Learning Algorithms Work

- Some algorithms entertain *multiple* candidate hypotheses at the same time; as more data comes in, eliminate candidates from the set
  — representing and manipulating the (very large) set is problematic

- So most learning algorithms form a *single* hypothesis and then modify it so that it does a better and better job of accounting for the data
  — a search through the hypothesis space, moving from hypothesis to hypothesis

Continued

- How do we compute the successors of a hypothesis in this search?
  - *Generalisation:* if a hypothesis fails to cover a +ve example, then it is too specific and it needs generalising
  - *Specialisation:* if a hypothesis covers a -ve example, then it is too general and it needs specialising;

- Some algorithms start with the most specific hypothesis and repeatedly generalise

- Others start with the most general hypothesis and repeatedly specialise

Rule Learning (Inductive)

- Training set consists of many +ve and -ve examples, each specifying the value of a number of attributes

- E.g. from a company database we have data about individuals: some attributes and whether they were granted a loan or not

Individual	Job	Assets	Income	Balance	Loan
1	T	F	F	T	F
2	F	F	T	F	F
3	T	T	F	T	T
4	F	T	T	T	T
5	F	T	T	F	F
6	T	T	T	F	T
7	T	T	T	T	T
8	T	F	T	F	F
9	T	T	F	F	F

A rule learning algorithm

- We'll learn propositional rules, i.e.

  $$\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \text{Loan}$$

  where α_i come from Job, Assets, Income and Balance

- Specialisation: given a rule, make it more specific, i.e. cover fewer instances, by adding an atom to the antecedent of a rule
  — used when the rule covers some -ve examples and you want to prevent it from covering them

- Generalisation: given a set of rules, make them more general, i.e. cover more instances, by adding another rule
  — used when there are uncovered +ve examples and you want to cover them

The algorithm

```
CurrTrainingSet := TrainingSet;
CurrRuleSet := {};
while there are +ves in CurrTrainingSet
   that are not covered by CurrRuleSet
{  CurrRule := true -> Loan
   while there are -ves in  CurrTrainingSet
     that are covered by CurrRule
   {  Select an atom and add it to the
         antecedent of CurrRule
   }
   CurrRuleSet := CurrRuleSet plus CurrRule
   CurrTrainingSet := CurrTrainingSet minus
      +ves covered by CurrRule
}
```

- May not give 'optimal' result
  — At end, could try simplifications, e.g. try dropping rules/atoms to see if this has any effect
- Get an expert to scrutinize learned rules

Judging Machine Learning Algorithms

- *The improvement in subsequent performance:*
  - E.g. after learning, is the system faster/more space efficient/more likely to find optimal solutions
  - E.g. after learning using the training set, assume we have another set of examples of inputs and their correct outputs, the *test set*; when the system predicts outputs for these inputs, what percentage does it correctly predict?
  - A need for good methodology: cross-validation is needed to ensure that you didn't just hit lucky with this particular training set and test set
  - NB Be alert to the possibility that learning worsens performance (e.g. more knowledge may mean higher branching factor which may mean slower search)

Continued

- *Transparency:* Is what has been learned comprehensible?
  E.g. if you show what has been learned to a human expert, does s/he find it understandable/ plausible? Some representations are not transparent at all
- *Complexity:* time and space efficiency of the learning algorithm; time and space efficiency of the system after learning

Noise

- There is a possibility that the training set is *noisy*
  —- examples are misclassified
- E.g. (serious case) two identical examples with different labels
  — either we're missing a distinguishing attribute or we have noise
- E.g. (less serious) examples are wrongly classified but they don't actually contradict other examples
- Modify the algorithm:
  — be satisfied with a rule when it covers mostly +ves (only a few -ves covered)
  — be satisfied with the rule set when it covers most of the +ve examples (only a few +ves not covered)

Over-fitting

- *Over-fitting:* learning spurious distinctions
- E.g. suppose you've learned a rule that covers 99 out of 100 examples; if we generate a second rule to cover the 100th example, we've probably over-fitted
  — the 100th example is probably noise
  — avoid over-fitting in these circumstances using the techniques on the previous slide
- E.g. due to a quirk of the training set, an irrelevant attribute might appear relevant and form the basis of a distinction in the rules that we learn
  — avoid over-fitting in these circumstances by making sure the training set is large enough and by pruning/scrutinising the learned knowledge

Making the algorithm more powerful

- Make the representation more expressive. E.g. allow the predicates in the +ve Horn clauses to have arguments
- This now gives two ways to specialise: add an atom to antecedent, or replace an argument by something more specific (e.g. replace a variable by a constant)
- But now, an infinite number of next hypotheses: an infinite number of atoms and infinite number of ways of replacing a variable
- Must restrict, e.g. very common is to disallow function symbols

FOIL

- Essentially the algorithm we have described. But when specialising a rule, the following restrictions:
  - only allow predicates that appear in B or the consequent of the rule (... so, can learn recursive rules but can't 'invent' new predicates)
  - arguments must all be variables but at least one argument of each predicate must appear earlier in the antecedent or in the consequent
  - allow special predicates such as equality/arithmetic, whose arguments may be variables that are already in the rule or constants in B or E
  - sophisticated statistical heuristic (among other things, prevents antecedent getting longer than the number of examples it covers)
- Learned large number of simple Prolog programs

Golem

- Searches from specific to general, instead of general to specific
- Uses a form of inverse resolution
  - can 'invent' new predicates
  - but needs to be highly constrained
- Very successful
  - learned rules for satellite fault diagnosis, finite element mesh design, biological classification of river water quality, Prolog programs, structure-activity prediction for drug design (published) and protein secondary-structure prediction (published)