Rule Learning

Many successful expert systems have been implemented using rule-based techniques. But the rule-based approach is not without its problems. Some of these problems we will discuss in the next lecture. But here we address a problem that seems to threaten the whole approach: the difficulty of knowledge engineering.

Rules are little nuggets of knowledge. It was originally thought that knowledge engineers could interview domain experts and easily elicit such nuggets of knowledge. But this proved not to be true. The first handful of rules might be easily acquired, but it becomes increasingly hard to obtain more rules to improve the coverage of the knowledge base. Different experts may disagree. Experts may be unable to articulate their knowledge. The brake this put onto the development of expert systems became known as the *knowledge acquisition bottleneck*.

The problem of maintaining these knowledge bases is just as hard. Because rules chain together, they cannot be inserted, modified or deleted, without consideration of the 'knock-on' effects.

It might be that a degree of automation could streamline both the knowledge engineering and maintenance. The options investigated have included:

- Tool support. Programs have been written that help knowledge engineers to manage the knowledge engineering process. These tools might check the syntax or the consistency of the knowledge that gets entered, for example. Such tools help, but they don't make a big enough improvement.
- Natural language understanding. Rudimentary systems exist that can extract meaning from dialogues and text. Such systems might help us to interview domain experts or might extract knowledge from manuals and other documents. However, this technology has, as yet, too many limitations.
- Machine learning. Rules can be extracted from databases of, e.g. example symptoms/diagnoses.

We look at the last of these in more detail.

1 Supervised Inductive Learning

1.1 Supervised Learning

The learning task that we face is basically the same as the one we looked at when we were considering how to build classification systems using Artificial Neural Nets. (But the learning algorithm is quite different.) We start by recapping some of the concepts that we covered previously.

We're looking at *supervised learning*. The learner will obtain some data related to what it is trying to learn, which we call the *training set*, and, in supervised learning, the training set contains examples of inputs and their corresponding outputs.

In the simplest case, which is all that we will look at in this lecture, the task might be, e.g., to learn some rules that decide whether a patient has a bacterial infection or not, or whether a loan applicant should be granted a loan or not. Therefore, the target outputs in the training set will be simply **true** or **false** . Those in which the target output is **true** are called *positive examples*; those in which the target output is **false** are called *negative examples*. (We must beware! The training set may be *noisy*! In other words, some of the examples may be misclassified with the wrong target output.)

1.2 Inductive Learning

Here's a (rough) logical formulation of the task. Suppose we've some *background knowledge*, B, and suppose we've some *examples*, E — some are positive examples, E^+; others are negative examples, E^- . The learning task is to find some hypothesis, H, such that:

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

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

What do we require of H? We require that B, E and H aren't contradictory:

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

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

$$B \not\models H$$

In other words, H is 'new' knowledge. We also require that, without H, the examples wouldn't follow from the existing knowledge:

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

But this is not enough.

In inductive learning, we don't only require that H explain the examples in the training set, the *seen examples* ($B \wedge H \models E^+$). We also require H to 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 $H =$ "*all* birds fly". It doesn't prove that the next bird we see will fly; it's just good evidence. Induction involves making leaps, which may turn out, when we see more training examples, to be wrong. H must generalise without overgeneralising.

H must be chosen from among many candidates. The set of all hypotheses from which the learner can choose is called the *hypothesis space*. Hypothesis spaces are typically huge, even by AI standards.

Some learning algorithms entertain *multiple* candidate hypotheses at the same time; as more data comes in, they eliminate candidates from the set. The problem with these algorithms can be that representing and manipulating the (very large) set of candidate hypotheses is expensive in time and space.

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. We can see this as a search through the hypothesis space, moving from hypothesis to hypothesis.

In this search, how do we compute the successors of a hypothesis? Two basic processes may be at work:

- *Generalisation:* if a hypothesis fails to cover a positive example, then it is too specific and it needs generalising.
- *Specialisation:* if a hypothesis covers a negative 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. Usually a mix of both processes is needed.

2 Rule Induction

We'll look at a machine learning algorithm for inducing rules from positive and negative examples. In our case, we can imagine that the examples have come from a company database and give us data about individual customers of that company. In each example, we have some attributes about the individual. They just happen to all be boolean-valued attributes, but in general they could be numeric-valued or use some other data type. The attributes are: whether the individual has a job or not, whether the individual has substantial assets (e.g. a house) or not; whether the individual has a substantial income or not, and whether the individual has a substantial bank balance or not. Positive examples are ones where our company granted the individual a loan; negative examples are ones where our company chose not to grant the individual a loan.

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

To begin with we'll assume that the data is not noisy.

We want to learn rules that can be used to classify unseen individuals into those to whom we should award a loan and those who should be denied a loan.

The rules we will learn are ones that use propositions (predicates with no arguments).

$$P_1 \wedge P_2 \wedge \ldots \wedge P_n \Rightarrow \mathit{Loan}$$

where P_i come from *Job*, *Assets*, *Income* and *Balance*.

The algorithm will build up a set of rules by using both specialisation and generalisation:

- Specialisation: Given a rule, we may need to make it more specific, i.e. so that it covers fewer examples. We do this by adding an atom to the antecedent of a rule. We'll do this when the rule covers some negative examples and we want to prevent it from covering them.
- Generalisation: Given a set of rules, we may need to make them more general, i.e. so that they cover more examples. We do this by adding another rule. We'll do this when there are uncovered positive examples and we want to cover them.

Here's 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
}
```

In the lecture, we'll trace the execution of this algorithm on the data given earlier.

At this point, you should re-read the material in our lecture on Classification about how to evaluate learning algorithms. The discussion there about training sets (the examples), test sets and cross-validation is relevant here. This way we can get some idea of the accuracy of the rules when they make predictions on unseen data.

There is no guarantee that this algorithm will find an 'optimal' set of rules. E.g. it is not guaranteed to find a minimal set of rules. So it is common to include some sort of post-processing of the rules. One thing that can be done automatically is to try out a few simplifications: try dropping rules or atoms and see how this effects coverage of the examples. Perhaps a better thing to do is to get an expert to scrutinize the learned rules. The expert can act as a 'sanity check' and might also be able to suggest simplifications. This is possible because rules are such a 'transparent' notation. Contrast this with what we said about neural networks.

3 More Powerful Rule Induction

There is a possibility that the training set is *noisy*: examples may be misclassified. How can we modify the algorithm to handle noise? Actually this is easy:

- Be satisfied with a rule when it covers mostly positives and only a few negatives. Don't add any more atoms at this point.
- Be satisfied with the rule set when it covers most of the positive examples and only a few positives are not covered. Don't add any more rules at this point.

Considerable research effort has also been devoted to devising variants of this algorithm that can learn rules in which the predicates do have arguments. In some ways, all this does is increase the number of ways in which we can specialise a rule: we can add an atom to antecedent (as at present in the algorithm), or we can replace an argument by something more specific (e.g. replace a variable by a constant symbol). The trouble with this is that there is now an infinite number of next hypotheses. This is because there is an infinite number of atoms and infinite number of ways of replacing a variable with something more specific. So a lot of research has gone into working out ways to constrain this more.

Systems that are representative of this more powerful form of rule induction include:

- Foil, which learned large number of simple Prolog programs from example input/outputs.
- Golem, which 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).

Exercise (Past exam question)

1. **Briefly** explain what is meant in A.I. by the term *classification*.

2. A novice knowledge engineer is building a system for carrying out classification. S/he asks you whether s/he should use a neural network or a rule-based system.

   What would you want to find out from this knowledge engineer and how would it affect your advice?

3. A university Computer Science department has collected data that it thinks can be used to predict whether a prospective student will pass first-year Computer Science exams (*willPass*). For each past student, the data records whether the student passed Leaving Certificate exams in *maths*, *physics*, *latin*, *french* and *english*:

Student	*maths*	*physics*	*latin*	*french*	*english*	*willPass*
1	F	T	T	F	F	T
2	T	T	F	F	T	T
3	T	F	T	F	T	T
4	F	T	F	T	F	F
5	F	F	F	T	T	F
6	T	F	F	F	T	F
7	T	F	F	T	T	T
8	F	F	F	T	T	F

   Assume that the data is not noisy.

   The rule induction algorithm covered in lectures is being applied to this data. Show **in detail** how the algorithm proceeds. Explain each step. Stop after you have learned the second rule, just prior to moving on to learning the third rule.

4. State, in general terms, how the rule induction algorithm can be modified to handle *noisy* data.

5. Suppose that you want to predict, not just whether a student will pass first-year Computer Science exams, but how well s/he will perform (*firstClass*, *secondClass*, *thirdClass*, *pass* and *fail*). So the data collected might now look like this:

Student	*maths*	*physics*	*latin*	*french*	*english*	*performance*
1	F	T	T	F	F	*secondClass*
2	T	T	F	F	T	*firstClass*

   etc.

   State **briefly** how you think we might use or modify the rule induction algorithm so that we can learn rules from this data.