Concept Learning by Experiment

Claude Sammut

School of Electrical Engineering and Computer Science
Unlverlaty of New South Wales
P.O. Box 1, Keneington, N.S.W., Auetrelle, 2033.

1. Introduction

Mich of the emphasis in current research on concept learning and rule Induction is based on two assuaptione. Firstly, all that needs to be known to learn a concept can be obtained directly from exaaplea given to It, without reference to previously learnt knowledge. Secondly, a sufficient nuaber of exaaplee to learn the concept is available, and all exaaplee are presented slaultaneously.

Relatively little attention has been given to developing systems which are able to improve their performance over time by using knowledge that has been learnt before. Two exceptions are (Winston70a] and [Cohen78a] •

The usual task for a learning program is: Given a set of positive Instances and a set of negative Instances, produce a concept description which distinguishes between these two sets. The program passively accepts its input and otherwise does not interact with the environment. Furthermore, It Is expected that the concept will be learnt in one session.

A program has been developed which learns concepts by searching a knowledge base which is augaented each time a new concept is learnt. A concept description aay be treated as a program which may be executed. The output will be the description .of an object which is an instance of the concept. A trial concept may be tested by executing the description as an experiment to see if the desired result is produced.

2. Representing Concepts

Suppose a trainer gives the program a description of an object which characterises the concept to be learnt. For example, an instance of "on-top-of" is,

S1 - <shape: sphere; colour: red>
B1 =<shape: box; colour :green>
E1= <top: S1; bottoa: Bl>

Objects ere described by a list of attribute/value pairs.

The description language used is based on first order predicate calculus (with quantifiers). It evolved froa DL [Banerj 169a] and CODE [Cohen78a]. A description of El as a concept may be,

```
[X1:
    ∃X2, X3: top of X1 is X2          (1)
           & bottom of X1 is X3       (2)
           & shape of X2 is sphere    (3)
           & colour of X2 is red      (4)
           & shape of X3 is box       (5)
           & colour of X3 is red      (6)
    ]
]
```

A concept description may include references to previously learnt concepts, for example, "flat(X3)". "Flat" is a concept already learnt and is true if applied to the shape of X3 which, in the example, is the box Bl.

The equivalence relation "is" need be the only built in relation known to the system. All other relations can be constructed using "is" and referring to other concepts.

3. The Generalization Process

Suppose tha following concepts are already known,

```
flat = [X1: shape of X1 is box
          | shape of X1 is table
       ]
has-shape  = [X1: shape of X1 is sphere
                | shape of X1 is pyramid
                | flat(X1)
              ]

has-colour = [X1: colour of X1 is red
                | colour of X1 is blue
                | colour of X1 is green
              ]
phys-obj   = [X1: has-shape(X1)
                & has-colour(X1)
              ]
```

"has-shape" and "hat-colour" specify the range of values acceptable at shapea and colours, "flat" is a subset of shapes, "phys-obj" states that a physical object is a thing which has shape and colour.

If we examine the description of E1 we see that stateaent (3) Batches the first disjunct of "has-shape". That is, SI is sn instance of objects which possess the property "shape". Thus a generalisation of (3) is

has-shape(X2) (7)

Similarly, (4) can be generalized to

has-colour(X2) (8)

Now, statement. (7) and (8) together match the statements in "phys-obj", therefore we can deduce

phys-obj(X2) (9)

We can go through the same process with X3:

flat(X3)	(10)
has-shape(X3)	(11)
haa-colour(X3)	(12)
phys-obj(X3)	(13)

Note that we have not tested whether any of these generalisations are relevant to the concept which the trainer wants the program to learn. In the following section we describe how the program tests Its generalisations.

4. Using Concept Descriptions as Programs

Consider the following description of "on-top-of":

```
[X1:
        [∃X2, X3: top of X1 is X2
                & bottom of X1 is X3
                & phys-obj(X2)
                & flat(X3)
                & phys-obj(X3)
]       ]
```

If I assert that [X: on-top-of(X)] , that is, X is an "on-top-of" situation, then the program will construct X. In other words the program will find an instance of "on-top-of". It does this by attempting to prove the assertion by s process similar to resolution theorem proving.

During the learning process, when the program wishes to test its trial concept, it generates an instance of this trial and shows it to the trainer. The description of "on-top-of" given above is the correct one. However, suppose that in the process of making generalizations, the program tries phys-obj(X3) without the qualification "flat(X3)". In this case, the object which is constructed may have a pyramid on the bottom which is not allowed.

5. An Example of Learning

Given E1 as a positive instance, the program tries to produce a generalised description of "on-top-of". The first generalisation will Involve the replacement of statement (3) by (7). In order to test its generalisation, it proposes to the trainer an alternative "on-top-of" situation;

S2 - <shape: pyramid; colour: red>
E2 - <top: S2; bottom: Bl>

Note that when a statement such as (3) has been replaced, the object construction algorithm ensures that S2 will not be a sphere. Since this a good example of on-top-of, the generalisation is correct.

In fact the specification of X2 can be generalized to any physical object. However, if the program allows X3 to have any shape, then the following object may be constructed:

P1 • <shape: pyramid; colour: green>
E3 - <top: S1; bottom: Pl>

With a pyramid on the bottom, E3 does not describe a stable structure, therefore the trainer rejects this alternative*

The program now makes a new generalisation which attempts to explain why a pyramid cannot support another object. That is the generalisation must be restricted so that only objects which may support other objects are allowed. This restriction process looks at the statements which have been replaced by the incorrect generalization to determine how to specify the subset. In this example it is necessary to return "flat(X3)" to the concept description.

6. Conclusion

The learning process may be regarded as a form of automatic programming. The algorithm demonstrated here, although simple, is capable of learning quite complex concepts, including list reversal and other abstract recursive concepts. Once "reverse" is known, the system can actually perform a list reversal on given input data.

Since each disjunct of a concept is learnt separately, it is possible to build the complete description over a period of time, possibly interrupted by another learning task.

References

Banerji69a.
R. Banerji, Theory of Problem Solving - An Approach to Artificial Intelligence, American Elsevier, New York (1969).

Cohen78a.
B. L. Cohen, A Theory of Stuctural Concept Formation and Pattern Recognition, Ph.D. Thesis, Dept. of Computer Science, University of N.S.W (1978).

Winston70a.
P. H. Winston, Learning Structural Descriptions From Examples, Ph.D Thesis, MIT Artificial Intelligence Laboratory (1970).