CPS331 Lecture: Symbolic Learning last revised October 25, 2018

Objectives:

1. To become familiar with some examples of the use of supervised, reinforcement, and unsupervised learning in symbolic AI systems.

Materials:

1. Projectable of version space learning examples (use pdf with single page full screen)
2. Projectable of Cawsey Figure 7.2
3. Projectables of Luger Table 10.1, Figures 10.13, 10.14
4. Projectable of Logsdon Figure 65
5. Projectable of partial game tree for Hexapawn
6. Projectable of Logsdon Figure 66
7. Hexapawn matchbox set
8. Davis and Lenat pp 3-7, 10-12 to read, plus projectables of pp 16, 22-23

I. Introduction

A. Today we consider how a symbol-system representation of information can be learned.

B. Recall, from the previous lecture, that we noted that learning can take place using three different paradigms. What are they? What characterizes each?

ASK

1. Supervised learning, in which a "teacher" presents the system with examples and non-examples of some concept, and the system is expected to adjust its classification scheme to conform to the classifications given by the teacher.

2. Reinforcement learning, in which the system is only told whether it succeeded or failed at a given task, but not what the correct answer is if it failed.

3. Unsupervised learning, in which the system develops its own concepts based exploration of a domain.

C. All three paradigms can be used with symbolic AI. We will look at one or two examples of each.

II. **Examples of Symbolic Supervised Learning**

A. Recall that supervised learning is particularly appropriate for classification problems. Many practical applications of supervised approaches to learning have the following general form:

1. We have a large - possibly infinite - space of possible instances which we would like our program to classify correctly

2. We can teach our program by selecting a subset of all the possible instances called a training set, giving the program the instance and the correct answer, and allowing it to develop a classification rules that gives the correct answer for every instance in the training set. Our hope is that the program will learn a classification rule that is general - i.e. applicable to all instances of the problem, not just to the ones in the training set.

3. This assumes, of course, that our space of problem instances is consistent and has some learnable properties.

a) This approach will not work if there are two different instances having the same description but different classifications.

b) It also won't work if there is no set of learnable properties that distinguish the instances. For example, the concept of "dog" is learnable by being presented with enough examples of dogs and non-dogs; but the concept "member of a set of randomly-selected objects" is not learnable, because the only way to correctly classify all of them is to see all of them!

B. We will first consider an approach that is appropriate for a subclass of classification problems

1. The problem must have two characteristics:

   a) It must be a <u>decision problem</u> - that is one where the issue involves deciding whether or not a particular instance is an example of a particular category.

      Example: in our animal identifier expert system, the individual rules were of this sort - e.g. we had the rule

      ```
      if      the animal is a carnivore
              it has a tawny color
              it has black stripes
      then    identify it as a tiger
      ```

   b) An individual must be capable of being described by a conjunction of predicates

      Example: We might have an individual described as follows:

      ```
      advertises(tony, frosted_flakes) ^ carnivore(tony)
      ^ voiced_by(tony, lee_marshall) ^ tawny(tony) ^
      daughter(tony, antoinette) ^ black_stripes(tony)
      ```

      (See the Wikipedia entry for "Tony the Tiger")

2. The goal or our learning algorithm will be to induce a rule that reliably distinguishes examples and non-examples. For example, given a suitable set of training data, we might hope to induce a rule like the following:

   ```
   carnivore(X) ^ tawny(X) ^ black_stripes(X) → tiger(X)
   ```

3. In such a case, we can use an approach known as <u>version space learning</u>.

a) The basic idea is this: at any point in the learning process, there are many rules that could potentially be used to discriminate the individuals presented thus far that belong to the class from individuals presented thus far that do not belong. Each rules takes the form of a conjunction of predicates which an object must satisfy to be regarded as a member of the class.

   Example: Suppose our training data included just our description of tony plus the following

```
advertises(chester, cheetos) ^ carnivore(chester)
^ voiced_by(chester, johnny_michaels) ^
tawny(chester) ^ dark_spots(chester)
```

   (See the Wikipedia entry for “Chester Cheetah”)

   (1)Then all of the following (and many others) would be rules that would reliably distinguish the one example of a tiger we have seen from the one non-example we have seen:

```
advertises(X, frosted_flakes) → tiger(X)
voiced_by(X, lee_marshall) → tiger(X)
daughter(X, antoinette) → tiger(X)
black_stripes(X) → tiger(X)
```

   (2)Of course, all of these rules fall short of correctness - the first three would reject many legitimate tigers, and the last would accept many non-tigers as well (This is a consequence of an inadequate set of training data, of course.)

b) We will now show how the version space algorithm could discover the correct rule, given the following training data. (Note: this is somewhat contrived data that I have selected to make the example converge quickly!)

   PROJECT

(1)Examples:

(a) `advertises(tony, frosted_flakes) ^ carnivore(tony) ^ voiced_by(tony, lee_marshall) ^ tawny(tony) ^ daughter(tony, antoinette) ^ black_stripes(tony)`

(See the Wikipedia entry for “Tony the Tiger”)

(b) `friend(tigger, pooh) ^ friend(tigger, piglet) ^ carnivore(tigger) ^ voiced_by(tigger, jim_cummings) ^ tawny(tigger) ^ black_stripes(tigger) ^ created_by(tigger, milne)`

(See the Wikipedia entry for “Tigger”)

(2)Non-examples:

(a) `advertises(chester, cheetos) ^ carnivore(chester) ^ voiced_by(chester, johnny_michaels) ^ tawny(chester) ^ dark_spots(chester)`

(See the Wikipedia entry for “Chester Cheetah”)

(b) `black_stripes(harry) ^ middle_eastern(harry) ^ gray(harry) ^ carnivore(harry)`

(Harry is a striped hyena)

(c) `tawny(zoe) ^ black_stripes(zoe) ^ ungulate(zoe) ^ home(zoe, liberia)`

(Zoe is a Zebra Duiker)

4. The version space algorithm works with two sets of rules:

a) The sets are these:

(1) A most general set, consisting of rules that accept the examples and reject the non-examples seen so far, such that no rule can be made more general (by dropping a conjunct) without accepting one or more non-examples.

(2) A most specific set, consisting of rules that accept the examples and reject the non-examples seen so far, such that no rule can be made more specific (by adding a conjunct) without rejecting one or more examples.

(3) Initially, the version space consists of a most general set which contains only the rule which accepts everything, and a most specific set which contains only a rule which rejects everything.

Example: For our "tiger" example:

G = { `true -> tiger(X)` }, S = { `false -> tiger(X)` }

b) The algorithm operates in such a way that the true rule always lies somewhere between a most general rule and a most specific rule. As the algorithm operates, rules are modified or deleted until the two sets converge down to a single rule common to both sets, which is the target rule.

PROJECT: Cawsey Figure 7.2

c) The algorithm operates by presenting examples or non-examples one at a time. (Often, the order alternates between example and non-example, though not necessarily)

(1) When an example is presented

(a) Any most-general rule that rejects the example is deleted - since, by the definition of most general rule, it cannot be made more general without it accepting some non-example that has already been seen.

(b) Any most-specific rule that rejects the example is generalized by removing one or more conjuncts (but as few as possible) to make it accept the example. There may be more than one way to do this; each way results in a new most specific rule, so the most-specific rule set may grow as a result

The presentation of the first example is an exception - the most-specific rule is simply replaced by the example

(2) When a non-example is presented

(a) Any most-general rule that accepts the non-example is made more specific by adding one or more conjuncts (but as few as possible) from a most-specific rule that causes it to now reject the non-example. Typically, there is more than one way to do this; each way results in a new most general rule, so the most-general rule set grows as a result

(b) Any most-specific rule that accepts the non-example is deleted. since by the definition of most specific rule it cannot be made more specific without rejecting some example that has already been seen.

d) Applying this process to our example

PROJECT evolution of G and S sets

(1) Initially: G = { `true` }
S = { `false` }

(2) After presenting the example tony

```
G (unchanged)      { true }
S =     { advertises(X, frosted_flakes) ^
          carnivore(X) ^
          voiced_by(X, lee_marshall) ^
          tawny(X) ^
          daughter(X, antoinette) ^
          black_stripes(X) }
```

(a single rule with six conjuncts)

(3)After presenting the non-example chester

```
G = { advertises(X, frosted_flakes),
   voiced_by(X, lee_marshall),
   daughter(X, antoinette),
   black_stripes(X) }
```

(four different rules)

S (unchanged)

```
   {advertises(X, frosted_flakes) ^
   carnivore(X) ^
   voiced_by(X, lee_marshall) ^
   tawny(X) ^
   daughter(X, antoinette) ^
   black_stripes(X) }
```

(4)After presenting the example tigger

```
G = { black_stripes(X) }

S = { carnivore(X) ^
   tawny(X) ^
   black_stripes(X) }
```

(a single rule with three conjuncts)

(At this point, there is one rule in the most-specific set, and it happens to be the desired one, but we don't know this yet)

(5)After presenting the non-example harry:

```
G = { tawny(X) ^
    black_stripes(X) }
```

(a single rule with two conjuncts)

S (unchanged)

```
  { carnivore(X) ^
   tawny(X) ^
   black_stripes(X) }
```

(6)After presenting the non-example zoe

```
G = { carnivore(X) ^
      tawny(X) ^
      black_stripes(X) }
```

S (unchanged)

```
{ carnivore(X) ^
  tawny(X) ^
  black_stripes(X) }
```

(7)Our algorithm has now converged on the correct rule - one having three conjuncts.

5. In this example, we have used training data selected by a human teacher. But version space learning can also be used with a random set of training data that includes both examples and non-examples of the concept to be learned. (E.g., in this case, a database of information about animals)

6. The version space algorithm has one crucial limitations: it is limited to situations where the desired rule has the form of a conjunction of descriptive predicates.

   a) In the case of our animal identifier expert system, the version space algorithm could be used to induce the various animal classification rules given a database of descriptive information.

   b) However, it could <u>not</u> be used to deduce the rule for mammal, because it is actually a disjunction of two rules;

if	the animal has hair
then	it is a mammal

if	the animal gives milk
then	it is a mammal

C. Another paradigm for supervised symbolic learning involves inducing a decision tree from a large pool of data. This is not limited to simple example / non-example categorization as was true with version spaces.

1. An example: data on loan applicants

   PROJECT Luger table 10.1

2. Some examples of decision trees that correctly classify this data:

   PROJECT Luger Figures 10.13, 10.14

   While both correctly classify the data, the second is simpler.

3. There is a well-known algorithm known as ID3 that "learns" a decision tree from a set of raw data. It seeks to find the simplest possible tree by using information content considerations to select which piece of information to make the root of the tree - e.g., for this set of data, ID3 would produce the second tree rather than the first.

   (This preference for the simplest tree is based on a principle widely used in the sciences known as Occam's Razor - first articulated by William of Occam in 1323)

   "It is vain to do with more what can be done with less ... Entities should not be multiplied beyond necessity".

   a) The basic idea is this: at each stage in the development of the decision tree, we consider each of the possible properties that might be used to divide the data and choose the one that yields the greatest information gain - where information is defined using a rigorous definition from information theory.

      If ID3 were applied to the table projected earlier, it would select "Income" as first property to be selected first, because this property yields the greatest information gain when the raw data is partitioned based on this property.

In particular, the information gained by using each of the properties as the root of the tree is

(1) Gain(Income): 0.564 bits

(2) Gain(Credit History): 0.266 bits

(3) Gain(Debt): 0.063 bits

(4) Gain(Collateral): 0.206 bits

(For details, see Luger *Artificial Inteligence* p. 412 ff)

b) Our book developed an example of the development of a decision tree where the choice of properties was obvious and a rigorous information-theoretic analysis was not needed.

D. Both version space learning and ID3 require that two things be true of the training set.

1. It must be consistent. Clearly, if the training set contains two entries with identical descriptions but different classifications, no correct rule or decision tree exists; hence, of course none can be learned.

2. It must include enough cases to ensure convergence. For example, suppose our tiger data did not include the final non-example zoe (a tawny, striped, non-carnivore that I frankly had to look pretty hard for!). Then, when our learning algorithm had processed all the training data, the version space would be:

```
G =   { tawny(X) ^
        black_stripes(X) }

S =   { carnivore(X) ^
        tawny(X) ^
        black_stripes(X) }
```

- i.e. we would not yet have learned that being a carnivore is essential to being a tiger, and therefore we would know how to classify an instance like zoe.

3. Of course, when training with random data, we cannot know - in general - whether a training set will result in convergence to a correct rule without actually trying it. However, there is a theory called PAC learning which can help us know what size training set is likely to produce a close enough classification scheme, which is often sufficient in practice. (The development of this theory was a reason the person who developed this theory - Leslie Valiant - was awarded the Turing Prize in 2010)

   a) PAC stands for "probably approximately correct".

   b) We say a classification scheme is approximately correct to within some degree of error if it classifies most instances correctly. For example, an algorithm that classifies 99% of instances correctly would be approximately correct to a 1% error.

   c) We say that a training set size is probably successful within some degree of failure if a training algorithm using a randomly selected training set of this size succeeds in converging in most cases. For example, if, for a given random sample size, the training procedure converges to an approximately correct result 98% of the time, we say it is probably successful to within a 2% failure rate.

   d) It has been shown that we can get an approximately correct algorithm, to within an error of e, with a failure of rate of d, by using a randomly-chosen set of samples of size

      $(1/e)\ (\ln|H| + \ln 1/d)$ where H is the number of hypotheses

      For example, if we wanted to induce a classification scheme to distinguish tigers from non-tigers (H = 2) with a 99% rate of correctness 98% of the time, ($e = .01$ and $d = .02$) we would need a random sample set size of 461! (Now you see why I hand-tailored the example!)

E. Supervised learning is often used for problems involving classification or prediction (as in our two examples.) With neural (connectionist) networks, it is also used for control problems - but more on this later.

III. Examples of Symbolic Reinforcement Learning

A. An obvious place where reinforcement learning is useful is for two-player games - but this is certainly not the only place. (We will look at some other uses for reinforcement learning when we look at Genetic Algorithms.)

B. To illustrate how learning of this kind can be accomplished, we will look at an extremely simple (actually trivial) illustration of the use of reinforcement learning with the game of Hexapawn.

1. Hexapawn was described by Martin Gardner in The Unexpected Hanging and Other Mathematical Diversions (1969)

2. Hexapawn is played on a 3 x 3 board, using 6 pieces that can move and capture the way chess pawns do - hence the name “hexapawn”.

   PROJECT - Logsdon figure 65

   a) The object of the game is to do one of the following:

      (1) Advance one pawn to the opposite side of the board

      or

      (2) Capture all of the opponent’s pieces

      or

      (3) Create a situation in which the opponent cannot move

b) Because of symmetries, there are actually few game positions possible.

PROJECT - Partial game tree

c) It turns out that the player who moves second can always win.

3. It is possible to build a simple learning machine that learns to play perfect hexapawn using matchboxes and colored beads.

   a) We use one matchbox to correspond to each of the 24 positions (after eliminating symmetries) that the second player can confront in the game.

   b) The matchbox is labelled with a drawing of the board and colored arrows representing each of the possible moves (eliminating symmetries).

   PROJECT Logsdon Figure 66

4. The performance and learning process is described by Gardner as follows

   ”Pick up the matchbox that shows the position on the board. Shake the matchbox, close your eyes, open the drawer, remove one bead. Close the drawer, put down the box, note the color of the bead, find the matching arrow and move accordingly ... Continue this procedure until the game ends. If the robot wins, replace all beads and play again. If it loses, punish it by confiscating only the bead that represents its *last* move. Replace the other beads and play again. If you should find an empty box .... it means the machine has no move that is not fatal and it resigns. In this case, confiscate the bead of the preceding move.” (Quoted in Logsdon, *Computers and Social Controversy*, p. 235-237).

5. DEMO several games with matchbox set

6. We won't demonstrate this to completion, but it turns out that eventually this procedure will result in learning to play perfect Hexapawn. (Gardner reports an experiment in which this took 36 games, with the robot losing 11 times.)

C. Now, let's look at a couple of more challenging examples. As you recall, one of the keys to the success of a program that plays a challenging two-player game is the static evaluation function.

1. Often, such functions are hand-tailored by someone with expertise in the game.

2. But another approach that has been used with considerable success is makes use of a form of reinforcement learning.

D. An early example of a use of this approach is actually one of the earliest AI successes: Arthur Samuel's experiments (over a period of years) with checkers-playing programs.

1. Samuel identified a 38 of features that might be part of a static evaluation function.

   Example: A feature called EXCH which measured the relative exchange advantage of the player whose turn it is to move - i.e. the number of squares into which the player could move which would force an exchange of pieces with the opponent.

2. One of his programs learned weights for the various features as part of an overall evaluation function - i.e. which are most important, which are not.

3. Another of his programs learned which 16 of these features are most important for determining the relative value of a position.

4. Some of Samuel's programs learned by playing against other copies of the same program, others by playing against humans, and

others by comparing their performance against “book games” of masters (a form of supervised learning).

5. Pamela McCorduck quotes the following comment by Robert W. Nealey, a former Connecticut checkers champion and a nationally known player, concerning a game played in 1962:

   “Up to the 31st move, all our play had been previously published, except where I evaded “the book” several times in a vain effort to throw the computer’s timing off. At the 32-27 loser and onwards, all the play is original with us, so far as I have been able to find. It is very interesting to me that the computer had to make several star moves in order to get the win, and that I had several opportunities to draw otherwise. That is why I kept the game going. The machine, therefore, played a perfect ending without one misstep. In the matter of endgames, I have not had such competition from any human being since 1954, when I lost my last game.” McCorduck, *Machines Who Think*, p. 153

IV. Examples of Symbolic Unsupervised Learning

A. The final sort of learning we will consider is one in which the program itself generates new knowledge categories, without reliance on a “teacher” or an informed critic..

B. Here, a classic work is a program by Douglas Lenat called AM, which actually discovered a number of interesting mathematical concepts, such as the idea of prime numbers.

   1. READ DAVIS AND LENAT PP. 3-7; 10-12

   2. PROJECT: DAVIS AND LENAT PAGES 16, 22-23

   3. Note on the name - footnote on page 7.

C. One place where this kind of learning is used extensively is called data mining. Since a team will be addressing this later, we won't discuss it further now.

V. **Conclusion**

A. Learning remains a very difficult problem. Despite the successes that have been achieved in a few areas, it remains very much a research area.

B. Perhaps the biggest problem is what Winston calls Martin's Law: “You can't learn anything unless you almost know it already”. [This provides a rationale for incorporating a domain theory into a learner, as is done with explanation-based learning.]

C. For our next major topics, we will look at two altogether different paradigms for learning:

1. Genetic algorithms are inspired by viewing evolution as a process by which a species “learns” how to succeed in an environment.

2. Connectionist (neural) networks are modeled after the structure of brain. One of their great attractions is that they provide a framework that makes learning much more natural.