Cooperación en la tarea de Aprendizaje

Telma Delladio Fedi

Laboratorio de Investigación y Desarrollo en Inteligencia Artificial (LIDIA)
Departamento de Ciencias de la Computación
Universidad Nacional del Sur
e-mail: td@cs.uns.edu.ar

Introducción

La cooperación en la tarea de aprendizaje puede proponerse como una herramienta útil a la hora de solucionar muchos de los problemas que esta actividad presenta. Afrontar una tarea tan compleja de manera cooperativa permitiría obtener resultados más efectivos.

La capacidad de aprender cosas nuevas es una habilidad de las entidades inteligentes y por esto el aprendizaje es un tema de interés en Inteligencia Artificial. El área de Machine Learning se dedica al estudio e investigación de estos temas. En particular un área que ha ido tomando relevancia en estos tiempos es la Programación en Lógica Inductiva o ILP (Inductive Logic Programming) que propone a la Programación en Lógica como herramienta en la tarea del aprendizaje a partir de ejemplos.

Son muchos los elementos que intervienen en el problema del aprendizaje a partir de ejemplos lo que hace que sea un problema de alta complejidad. Para una entidad, el aprendizaje a partir de ejemplos es una tarea muy difícil que requiere, entre otras cosas, una gran cantidad de información de gran calidad para poder obtener buenos resultados. Si el aprendizaje se llevara a cabo de manera cooperativa por un grupo de entidades, es de esperar que esta tarea pueda realizarse de forma más rápida y efectiva.

Problemas en el Aprendizaje a partir de Ejemplos

Se han desarrollado varios sistemas que "aprenden" conceptos por medio de la síntesis de programas lógicos a partir de un conjunto de observaciones. Esas observaciones suelen dividirse en dos conjuntos: ejemplos positivos y ejemplos negativos. Los ejemplos positivos representan instancias ciertas del concepto que se desea aprender y los ejemplos negativos representan instancias que no pertenecen ese concepto [1,2]. Luego, a partir de esas observaciones, se genera (induce) una teoría o programa lógico que sirve como definición para ese concepto. Idealmente esa definición deberá satisfacer cualquier instancia del concepto real que se está aprendiendo. La calidad del programa lógico hallado dependerá, más allá de las técnicas utilizadas, de la calidad y cantidad de ejemplos considerados, del conocimiento previo que se tenga y de las restricciones operacionales que se impongan al momento de generar el programa.

Es imprescindible que los ejemplos tanto positivos como negativos sean representativos. De no ser así es casi imposible obtener una definición cercana al concepto real que se está modelando. Cuando se dan ejemplos positivos, lo que se suele hacer es dar ejemplos de instancias que son consideradas típicas y de instancias que representen casos especiales del concepto. De manera similar cuando se dan ejemplos negativos se intenta que estos sirvan para marcar las principales diferencias con los ejemplos positivos, posiblemente representen instancias de conceptos que son muy similares (que pueden llegar a confundirse) con el concepto que se pretende aprender. Muchas veces esto hace que al momento de proponer los ejemplos al sistema ya se tenga en mente una determinada idea de como debiera ser el programa lógico que se desea generar. Por ejemplo, cuando se quiere aprender el concepto: "un elemento pertenece a una lista", mediante el predicado "*pertenece(Elemento,Lista)*" es común que se dé una secuencia de ejemplos muy similar a la siguiente:

e_1^+: `pertenece(1,[1,2]).`
e_2^+: `pertenece(2,[1,2]).`

e_3^+: pertenece(3,[1,2,3,4]).
e_1^-: pertenece(3,[1,2]).
e_2^-: pertenece(4,[1,2]).
e_3^-: pertenece(5,[1,2,3,4]).

Una definición típica del predicado "*pertenece*" consta de dos casos, uno cuando el elemento está en la cabeza de la lista, y otro cuando el elemento está en la cola de la lista. El ejemplo e_1^+ lo que hace es simplemente representar el primer caso. El ejemplo e_3^+ se refiere al caso general y el ejemplo e_2^+ representa otra instancia que responde al segundo caso más precisamente cuando el elemento está al final de la lista. El conjunto de ejemplos negativos, muestra sobre las mismas listas, instancias que no corresponden al concepto que se desea aprender. En casos como este los ejemplos provistos no solamente sirven para ejemplificar el concepto que se desea aprender sino que también dan una pauta de los distintos casos que debería cubrir el programa que se desea generar: cuando el elemento es el primero en la lista o cuando el elemento está en la cola de la lista. Es decir, ya se tiene en mente la solución que se pretende al momento de dar los ejemplos.

Otro aspecto muy importante al momento de aprender un concepto es el conocimiento previo. Cuanto más sea el conocimiento previo posiblemente será más fácil la tarea de aprendizaje. Un concepto se define a partir de otros conceptos, por esto el conocimiento previo puede ayudar a definir nuevos conceptos de manera más directa. Por ejemplo, supongamos que contamos con una base de conocimiento que solamente define las relaciones *padre* y *madre* de una determinada familia y se desea aprender el concepto "*ancestro*". Con los ejemplos adecuados posiblemente se obtenga una definición como la siguiente:

```
ancestro(X,Y) ← padre(X,Z),ancestro(Z,Y).
ancestro(X,Y) ← madre(X,Z),ancestro(Z,Y).
ancestro(X,Y) ← padre(X,Y).
ancestro(X,Y) ← madre(X,Y).
```

En cambio, si se cuenta en la base de conocimiento con la definición de la relación *progenitor*, posiblemente se podría hallar una definición más compacta para el concepto *"ancestro"*.

```
ancestro(X,Y) ← progenitor (X,Z),ancestro(Z,Y).
ancestro(X,Y) ← progenitor (X,Y).
```

Si bien, el conocimiento previo puede ayudar a aprender conceptos de manera más clara o compacta, también es cierto que aumenta la complejidad del problema. El problema de aprendizaje a partir de ejemplos (a veces llamado Problema ILP) se puede plantear como el problema de encontrar un programa lógico que junto con el conocimiento previo sirva como definición correcta (con respecto a los ejemplos) de un determinado concepto. Esto es, se busca un programa lógico que utilice posiblemente conceptos definidos en el conocimiento previo. Es fácil ver que cuantos más conceptos haya en el conocimiento previo más posibilidades habrá al momento de definir el programa lógico buscado.

En la definición formal del Problema ILP [1,3] se define un elemento conocido como Espacio de Hipótesis. Este espacio representa el conjunto de todas las posibles definiciones que pueden considerarse. Si la descripción buscada es un programa lógico que defina un concepto, entonces el Espacio de Hipótesis no es otra cosa que el conjunto de todos los programas lógicos (cláusulas) que podrían ser la solución al problema de aprendizaje. De esta forma, el trabajo de generar una definición para un concepto se traduce a buscar dentro de ese Espacio de Hipótesis una definición que sea correcta con respecto a los ejemplos. El principal problema es que el Espacio de Hipótesis es muy grande aún en los casos más triviales.

Existen varias formas de recorrer ese Espacio de Hipótesis y de acuerdo a esto se suele clasificar a los sistemas como Top-Down o Bottom-Up [1,3]. Los primeros comienzan con una definición muy general del concepto (generalmente la cláusula vacía), y van especializándola hasta encontrar una definición adecuada mientras que los otros comienzan con una definición muy específica y van generalizándola. En cualquiera de los dos casos cada paso de refinamiento (especialización o generalización) toma una cláusula *C* a refinar y genera un conjunto de cláusulas que son refinamientos de *C*. De ese conjunto hay que elegir un elemento para continuar la búsqueda a partir de allí. Esto obliga a que estos sistemas tengan que definir de antemano restricciones o heurísticas que permitan reducir el tamaño del Espacio de Hipótesis, es decir que las opciones en cada paso de refinamiento sean mínimas. Algunas de estas restricciones incluyen la declaración de modos y tipos [1,2] para los argumentos de los literales que pueden aparecer en una cláusula. Otras restringen la

definición de cláusulas recursivas para asegurar la terminación de cómputo en programa final generado, *etc*.

Este es uno de los principales problemas que se debe afrontar en la síntesis de programas. Sin un adecuado conjunto de heurísticas y restricciones no se podrían construir sistemas que operen de manera aceptable. Es necesario por lo tanto buscar propuestas que ayuden en el trabajo de la síntesis programas, permitiendo la construcción de sistemas más efectivos.

Cooperación en la tarea de aprendizaje.

La cooperación en la tarea de aprendizaje es una propuesta que podría servir en el desarrollo de sistemas más efectivos. El aprendizaje a partir de ejemplos es una tarea muy compleja y esa es la razón que impide encontrar soluciones claras eficientes. En general, cuando un problema es muy difícil de solucionar lo que se suele hacer es buscar la manera de solucionarlo por partes para luego integrar las soluciones parciales en una solución general. Si esto fuera posible, estos subproblemas podrían abordarse simultáneamente logrando solucionar el problema inicial de una forma mucho más eficiente.

Es posible pensar en un sistema multiagente dedicado a la solución del problema de aprendizaje a partir de ejemplos. En este caso, el problema podría dividirse en la generación de definiciones parciales del concepto que se desea aprender para luego unificar esas definiciones parciales en una definición general.

En el caso general, el problema ILP admite que la tarea de aprendizaje comience con una teoría o programa no necesariamente vacío. Esto es posible cuando se utilizan mecanismos de aprendizaje en la *revisión o corrección* de teorías o programas lógicos.

Como ya se dijo, la cantidad y calidad de información provista a un sistema de aprendizaje será de suma importancia al momento de aprender un concepto. Si se considera un sistema multiagente, cada integrante puede contar con un conocimiento previo y con un conjunto de observaciones propias. Esto le puede permitir a cada uno, generar definiciones parciales del concepto que se pretende aprender en conjunto. Cada una de estas definiciones parciales, no solamente será útil para quien la haya generado sino también puede serlo (al menos en un principio) para el resto de los integrantes del sistema. Si uno de los integrantes prueba que una determinada definición parcial es *consistente* con sus observaciones, podría adoptarla para acelerar el aprendizaje.

La ventaja de un esquema como este, no solamente está en el paralelismo que puede obtenerse con varias entidades resolviendo en conjunto el mismo problema. Son muchos y variados los mecanismos de aprendizaje que se han propuesto [1,2,3,4]. Es posible entonces, que cada integrante del sistema implemente un mecanismo de aprendizaje diferente con ventaja potencial que esto significa. Esto posibilitaría contar con las ventajas de cada mecanismo, el cual podría adaptarse a las propiedades de cada agente (conocimiento previo, observaciones). Una elección adecuada permitiría balancear las ventajas y desventajas de cada uno de ellos logrando así complementar las deficiencias que presentan por separado.

Referencias

[1] F. Bergadano and D. Gunnetti. *"Inductive Logic Programming: from Machine Learning to Software Engineering"*, The MIT Press, 1996.

[2] I. Bratko. *"Prolog Programming for Artificial Intelligence"*, Addison Wesley, 2nd edition, 1990.

[3] S-H. Nienhuys-Cheng and Ronald de Wolf. *"Foundations of Inductive Logic Programming"*, Springer - 1997.

[4] S. J. Russell and P. Norvig. *"Artificial Intelligence. A Modern Approach"*, Prentice-Hall, 1995.