Inteligencia Artificial (I)

Aplicaciones prácticas

David Arroyo Menéndez

En este artículo se mostrarán algunas de las principales aplicaciones para las cuales se utilizan las técnicas de inteligencia artificial. Así mismo, se ilustrarán estas aplicaciones mediante explicaciones de código y de aplicaciones libres disponibles.

Introducción

Este artículo ha sido titulado *Aplicaciones Prácticas de Inteligencia Artificial*, esta introducción debería explicar tal título. Por aplicación práctica entendemos utilizar alguna técnica derivada de un área del conocimiento para resolver un problema concreto que afecta a nuestra vida cotidiana. Para no complicarnos definiremos inteligencia artificial tal y como lo hacen Rich y Knight, esto es, lograr que las máquinas realicen tareas que por el momento hacen mejor los humanos.

Turing pensó que si una máquina se comporta en todos los aspectos como inteligente, entonces debe ser inteligente. Por lo que si un número alto de humanos no puede diferenciar a una máquina de una persona en una conversación es porque se comporta de manera inteligente. Esto se llamó el Test de Turing (http://plato.stanford.edu/entries/turing-test) y el primer programa en pasarlo fue Eliza, que imitaba el comportamiento de un psicoanalista en un chat de Internet. Eliza daba respuestas que eran preguntas acerca de lo que estabas diciendo, con lo cual sin ningún conocimiento real de psicología Eliza pasaba el test de Turing. Las soluciones que se aportan a las tareas que realizan mejor los humanos suponen muchas veces un compromiso entre parecer que se comporta de manera inteligente y realizar los procesos mentales que hacemos los humanos. Para ver una implementación libre de Eliza podéis hacer:

```
Esc-x doctor
```

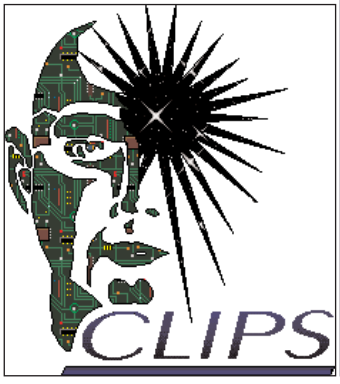
En vuestro emacs.

Demos un repaso para ver cuáles pueden ser esas tareas que los humanos realizan mejor que las máquinas:

- Juegos. Durantes algunos años los fabricantes de ordenadores demostraban la potencia de su tecnología jugando a ajedrez o damas contra campeones mundiales.
- Consejos. Los humanos solemos pedir consejo a otros humanos cuando padecemos una enfermedad, se nos estropea el coche, ante la compra de una casa o para otras decisiones que necesitamos de un conocimiento que no siempre poseemos. Desde hace años los sistemas basados en conocimiento o sistemas expertos hacen esta tarea con efectividad similar a como lo haría un experto en la materia.
- Lenguaje Natural. Hoy en día las máquinas no hablan nuestro idioma: lenguajes de programación, líneas de comando o interfaces gráficas nos sirven para comunicarnos con estos, pero el comunicarnos con ellos como nos comunicamos con personas constituye un reto que revolucionaría la informática con aplicaciones inimaginables.
- Aprendizaje. Una de las principales diferencias entre seres vivos e inertes es la capacidad de adaptarse a los cambios, la capacidad de aprender. Algoritmos de aprendizaje automático son cada día más utilizados en una cantidad creciente de aplicaciones.
- Empatía. Rara vez el ordenador se pone en el lugar del humano comprendiendo su objetivo abstracto y ayudándole a solucionarlo. ¿Pueden las máquinas hacer tal cosa?
- Sentido común. Existe una gran cantidad de conocimiento que aprendemos en nuestra más tierna infancia, que las personas conocemos como lo relativo a consideraciones del tiempo, el espacio, los materiales... y que incluso las aplicaciones que requieren cierta inteligencia, como los sistemas basados en conocimiento, rara vez incluyen.
- Pensar con neuronas. Tal vez si conseguiéramos crear un sistema artificial que simulara nuestro sistema neurológico y lo entrenáramos en un entorno adecuado pudiera llegar

Profundidad, *por Maurits Cornelis Escher.*

Logo de Clips.



a tener un nivel de inteligencia similar al nuestro. Lo cierto es que este tipo de sistemas está obteniendo muy buenos resultados en problemas que había sido muy difícil abordar por otras vías, como por ejemplo el reconocimiento de formas.

En este artículo veremos que las técnicas que solucionan o se aproximan a la solución de tales problemáticas nos ayudan a resolver cuestiones de nuestra vida cotidiana y que, además, esto puede hacerse utilizando software libre.

Juegos

La mayoría de los juegos de mesa y una gran cantidad de problemas informáticos pueden resolverse mediante una adecuada modelización en estados y aplicando un algoritmo de búsqueda entre estos estados.

Para acotar el problema, pensemos en un juego de tablero de 2 jugadores, por ejemplo, las 3 en raya, las damas, el ajedrez, el othello, etc. ¿Cómo podría programarse un juego de este tipo?

Bien, en primer lugar, un tablero es una estructura de datos de tipo matriz donde cada elemento puede ser ocupado por el jugador 1, ocupado por el jugador 2 o vacío (en juegos como el ajedrez donde cada jugador tiene diferentes fichas habría que completar el enfoque). Una partida es una secuencia de estados por los que pasa un tablero. Ahora veamos cómo debería actuar nuestro sistema inteligente para ganarnos en una partida.

- Una primera aproximación podría ser tener todas las posibles partidas en memoria y aplicar solo los movimientos que han llevado solo a partidas victoriosas.
- Otro enfoque sería actuar en cada turno, teniendo en cuenta ciertas reglas estáticas (ciertos if). Un ejemplo de reglas en las 3 en raya puede ser: si tengo dos fichas alineadas entonces ocupa el último lugar vacío y gana la partida, si el oponente tiene 2 fichas alineadas ocupa el lugar vacío y evita la victoria del oponente, en cualquier otro caso mueve de manera aleatoria.
- En general, para este tipo de juegos se utiliza la estrategia minimax que imita el comportamiento humano de examinar por anticipado un cierto número de jugadas, explorando el grafo de tableros que se generarían tras un movimiento dado. En este enfoque existe una función de evaluación que da un valor a cada posible movimiento.

Listado 1 Extracto de código de tictactoe

```
# implements a simple win/block/random-move AI
  def move_ai_easy
    if (m = blockWin) >= 0
      @gamefield.field[m] = 2
    else
      # set random point
      while true
        x = rand(3)
        y = rand(3)
        if @gamefield.get(x,y) == 0
          @gamefield.set(x,y)
          break
        end
      end
    end
    @nmoves += 1
  end

  # implements a more clever AI using the negamax tree search method
  def move_ai_hard
    best = -Inf
    besti = -1
    @gamefield.player = 1

    order = [4,0,2,6,8,1,3,5,7] # see text
    0.upto(8) do |ctr|
      i = order[ctr]
      if @gamefield.field[i] == 0
        @gamefield.field[i] = 2
        value = -@gamefield.negamax
        @gamefield.field[i] = 0
        if value > best
          best = value
          besti = i
        end
      end

      break if (@nmoves == 0) && (best == 0) # see text
    end

    @gamefield.player = 2
    @gamefield.field[besti] = 2

    @nmoves += 1
  end
```

Para ilustrar la implementación de estos enfoques recomiendo instalar el paquete tictactoe que implementa las 3 en raya con 2 niveles de dificultad: easy (implementa reglas tontas) y hard (implementa minimax). Dejo el código de las funciones clave al lector (ver listado 1).

De este modo, queda ilustrado el modo en que se pueden aplicar técnicas de inteligencia artificial en los juegos.

Sistemas expertos

Los sistemas expertos resuelven problemas que normalmente son solucionados por expertos humanos.

Un sistema experto se compone de una base de conocimiento del dominio en cuestión, mecanismos de razonamiento para aplicar conocimiento a los problemas que se proponen, mecanismos para explicar a los usuarios el razonamiento utilizado a la hora de ofrecer una respuesta y mecanismos de aprendizaje y adquisición de nuevo conocimiento.

Para crear una base de conocimiento es necesario contar con al menos un experto humano del dominio en cuestión; se puede adquirir su conocimiento a partir de entrevistas o mediante interfaces amigables para que lo introduzca. Pero también es muy importante una adecuada modelización de su conocimiento.

Los mecanismos de representación del conocimiento que se pueden encontrar son: lógica, redes, marcos y reglas. No obstante, lo más utilizado es una adecuada combinación de marcos y reglas.

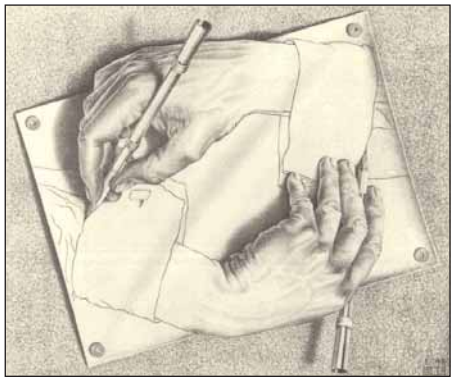
Es posible representar el conocimiento mediante hechos o instancias y reglas o algún otro mecanismo para inferir nuevos hechos. Pero deberemos plantearnos si a partir de los hechos vamos a ir aplicando reglas (encadenamiento hacia delante), o si por el contrario nos interesa responder a una cuestión concreta (por ejemplo: ¿tengo gripe?) e

Listado 2 **Ejemplo de uso de clips**

```
[darroyo@turing:/usr/share/doc/clips-doc/examples]$ clips
CLIPS> (load "auto.clp")
Defining deffunction: ask-question
Defining deffunction: yes-or-no-p
Defining defrule: normal-engine-state-conclusions +j
...
CLIPS> (run)
CLIPS> (reset)
CLIPS> (run)


The Engine Diagnosis Expert System

Does the engine start (yes/no)?
```



Manos dibujando, *por Maurits Cornelis Escher.*

ir satisfaciendo subobjetivos hasta llegar a hechos que demuestren la veracidad o falsedad de la frase (encadenamiento hacia atrás). También es conveniente plantear: qué reglas ejecutar antes, cuáles están listas para ser ejecutadas, si ejecutar las reglas cuyos antecedentes se actualizaron recientemente o si ejecutar primero reglas específicas. Todos estos mecanismos de control del razonamiento pueden ser tenidos en cuenta a la hora de construir nuestro sistema experto.

El software libre nos proporciona buenas herramientas para el desarrollo de sistemas expertos. Algunas de ellas son: clips (http://www.ghg.net/clips/CLIPS.html) y babylon (http://www-cgi.cs.cmu.edu/afs/cs/project/ai-repository/ai/areas/expert/systems/babylon/0.html).

Para ilustrar este apartado recomiendo instalar clips, que está empaquetado para un buen número de sistemas. En Debian basta con hacer:

```
apt-get install xclips clips clips-doc
```

xclips es un interfaz gráfico para utilizar clips; recomiendo su uso, para saciar la curiosidad del cacharreo. Desde línea de comandos podemos ejecutar lo que se muestra en el listado 2.

De este modo, podemos probar un ejemplo de sistema experto cuyo objetivo es solucionar averías del coche; el sistema va haciendo preguntas acerca de los síntomas que presenta el coche y finalmente otorga un diagnóstico. Para ejecutarlo el primer (run) no me funcionó; probé a hacer un reset y luego volver a hacer run y ya funcionaba. Os recomiendo que le echéis un vistazo al fichero auto.clp.

Espero que este apartado nos haya servido para entender que una máquina puede modelar el conocimiento de un experto en al menos un dominio limitado y que esto es útil para recibir asesoría o consejos para los que en condiciones normales precisaríamos de un experto humano.

Software adaptativo

Cuando aprendemos a programar, enseguida aprendemos ciertos principios básicos que hacen que nuestro software aumente su valor de uso: reutilización, abstracción, que sea usable para el usuario final... Sin embargo, no nos suelen preparar para que el software se adapte de manera automática a los cambios en las necesidades de usuario, en sus objetivos o en el entorno.

Según Peter Norvig (http://www.norvig.com), director de calidad de búsqueda de Google, "el software adaptativo usa información disponible acerca de cambios en su entorno para mejorar su comportamiento". Es decir, el software tiene capacidad de adaptarse al comportamiento del usuario sin que este lo pida de una manera explícita, entendiendo como usuario a una persona u otro sistema.

El software adaptativo ha tenido su caldo de cultivo en la minería de datos (*data mining*), esto es, la extracción no trivial, implícita, previamente desconocida y potencialmente usable de información de interés en grandes cantidades de datos. El proceso de extraer esa información se consigue aplicando algoritmos de aprendizaje automático.

Todo esto que puede estar pareciendo un poco de ciencia ficción está teniendo una fuerte eclosión en aplicaciones populares para el usuario final. Fijémonos, por ejemplo, en el problema del spam; actualmente evolution, mozilla, kmail y otros lectores de correo están introduciendo facilidades para separar el spam del ham, es decir, el correo electrónico no deseado del deseado. Para ello normalmente se aplica un algoritmo de aprendizaje automático denominado Naive Bayes. Muchos sitios web también están incorporando estos métodos para incrementar ventas o visitas. Es conocido el caso de Amazon, que utiliza filtrado colaborativo para encontrar usuarios con perfiles de compra similares y recomendar productos personalizados. Un ejemplo un poco más hispano es la Guía Campsa (http://www.guiacampsa.com), que utiliza una neurona artificial para ordenar el orden de rutas que prefieren los usuarios.

Ahora que tenemos un poco más claro para qué puede ser útil hoy el aprendizaje automático, se verá cómo crear un sistema que aprenda a decidir si es un buen día para jugar al tenis en función de una serie de atributos: fuerza del viento (*wind*), humedad (*humidity*), temperatura (*temperature*) y previsión del cielo (*outlook*). Para ello vamos a utilizar el algoritmo ID3 implementado en lisp por Tom Mitchell, uno de los padres del aprendizaje automático. Así es que nos descargamos un intérprete de lisp (si usamos Debian y queremos cmucl):

```
apt-get install cmucl
```

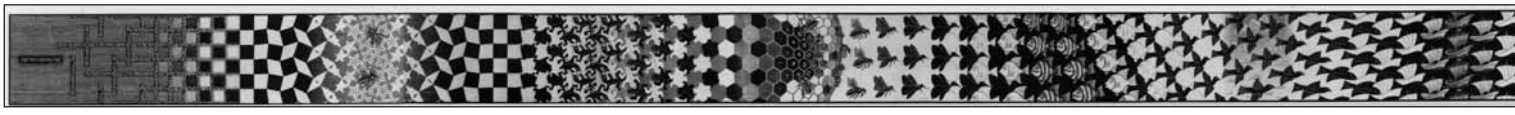
Y ahora el programa lisp en cuestión:

```
wget -c http://www-2.cs.cmu.edu/afs/cs/
project/theo-11/www/decision-trees.lisp
```

Finalmente, se ejecuta la traza que viene en el propio fichero (ver listado 3).

Este algoritmo de clasificación va construyendo un árbol de clasificación escogiendo en cada paso el atributo que tiene una mayor entropía, es decir, el atributo que tiene valores más variados. Después de haber recibido los ejemplos de entrenamiento ya podemos saber si, por ejemplo, d6 es un ejemplo de día para jugar al tenis o no: (*classify 'd6 tree)*.

Metamorfosis II, *por Maurits Cornelis Escher.*



Os animo a modificar un poco el código cambiando los ejemplos de entrenamiento, para familiarizarse con el mecanismo. El lector interesado en librerías de aprendizaje automático puede visitar las siguientes referencias, que además son libres:

- Torch (http://www.torch.ch/): Librería escrita en C++ con licencia BSD, implementa una gran cantidad de algoritmos de aprendizaje: redes neuronales, K-nearest-neighbors, modelos de Markov, clasificadores bayesianos...
- Weka (http://www.cs.waikato.ac.nz/~ml/weka/): Librería escrita en Java con licencia GPL, es bastante popular en el mundo académico y tiene un interfaz gráfico bastante amigable.
- Bow (http://www-2.cs.cmu.edu/~mccallum/bow/): Herramienta escrita en C con licencia GPL para el análisis estadístico de textos, clasificación automática de documentos desarrollada en Carnegie Mellon.
- Para el libro *Inteligencia Artificial: Un Enfoque Moderno,* Russell y Norvig desarrollaron un buena cantidad de código lisp con licencia similar a una BSD, entre el que hay bastantes algoritmos de aprendizaje automático (http://www.di.unipi.it/~simi/AIMA/doc/overview-LEARNING.html). Si usas debian puedes descargártelos mediante:

```
apt-get install cl-aima
```

El software adaptativo permite solucionar problemas donde el conocimiento estático no llega. El caso del spam es tal vez el más popular; no obstante, sistemas expertos, sistemas web, firewalls, encaminadores, etc., están incluyendo enfoques adaptativos para mejorar la eficacia de sus sistemas.

El objetivo de este apartado ha sido vislumbrar el modo en que puede aprender una máquina y la aplicación que tiene este aprendizaje de máquinas en el software que utilizamos en nuestra vida cotidiana.

Agentes inteligentes

Para Norvig & Russell un agente es cualquier cosa capaz de percibir su medioambiente mediante sensores y actuar en ese medio mediante actuadores. Todo agente tiene una función u objetivo. Por ejemplo, un agente humano de bolsa tiene el objetivo de comprar y vender acciones respondiendo a los estímulos iniciados por su cliente y captados por sus sentidos. Una aspiradora tiene la función de aspirar cuando capta que ha sido encendida y no aspirar cuando es apagada.

Un agente inteligente o racional trata de maximizar el valor de una medida de rendimiento, dada la secuencia de percepciones que ha observado hasta el momento.

Repitamos lo dicho ejemplificándolo. Un agente inteligente tiene un objetivo abstracto (ej: "ofrecer a un usuario información interesante"); tiene una forma de evaluar si esa información es interesante (ej: "el usuario lee la información sugerida"); tiene unos actuadores (ej: "una caja html donde presenta enlaces interesantes"), y tiene unos sensores (ej: "un conjunto de sitios web para recoger información y filtrar la que sea interesante y el conjunto de clicks que puede hacer o no el usuario de todos esos sitios web"). La pregunta ahora es ¿cómo mejorar ese rendimiento?

Para Peter Norvig la programación estructurada tiene asociadas las aplicaciones basadas en entrada/salida, la programación orientada a objetos las aplicaciones basadas en eventos y la programación adaptativa las aplicaciones basadas en agentes inteligentes. Es decir, la respuesta a nuestra pregunta es: usamos aprendizaje automático para mejorar el rendimiento.

No obstante, el deseo de desarrollar software adaptativo no es la única razón para utilizar una metodología de programación orientada a agentes. Los agentes tienen su campo de cultivo en la Inteligencia Artificial

Listado 3 — Traza de ID3 dada

```
[18:41][darroyo@turing:~/programacion/lisp]$ lisp
CMU Common Lisp CVS release-19a 19a-release-20040728 + minimal debian patches,
running on turing
With core: /usr/lib/cmucl/lisp.core
Dumped on: Fri, 2004-10-08 11:51:05+02:00 on turing
For support see http://www.cons.org/cmucl/support.html Send bug reports to the
debian BTS.
or to pvaneynd@debian.org
type (help) for help, (quit) to exit, and (demo) to see the demos

Loaded subsystems:
    Python 1.1, target Intel x86
    CLOS based on Gerd's PCL 2004/04/14 03:32:47
* (load "decision-trees.lisp")

; Loading #p"/home/darroyo/programacion/lisp/decision-trees.lisp".
Warning:  Declaring *DATA* special.
T
* *training.examples*

(D14 D13 D12 D11 D10 D9 D8 D7 D6 D5 D4 D3 D2 D1)
* (print.entity 'd6)

(PLAY.TENNIS? NO WIND STRONG HUMIDITY NORMAL TEMPERATURE COOL OUTLOOK RAIN)
(PLAY.TENNIS? NO WIND STRONG HUMIDITY NORMAL TEMPERATURE COOL OUTLOOK RAIN)
* (setq tree (id3 *training.examples*
                  'play.tennis?
                  '(outlook temperature humidity wind)))
Warning:  Declaring TREE special.

(OUTLOOK (SUNNY (HUMIDITY (NORMAL YES) (HIGH NO))) (OVERCAST YES)
 (RAIN (WIND (STRONG NO) (WEAK YES))))
* (print.tree tree)
OUTLOOK
 = SUNNY
     HUMIDITY
       = NORMAL => YES
       = HIGH => NO
 = OVERCAST => YES
 = RAIN
     WIND
       = STRONG => NO
       = WEAK => YES
NIL
* (classify 'd6 tree)

NO
```

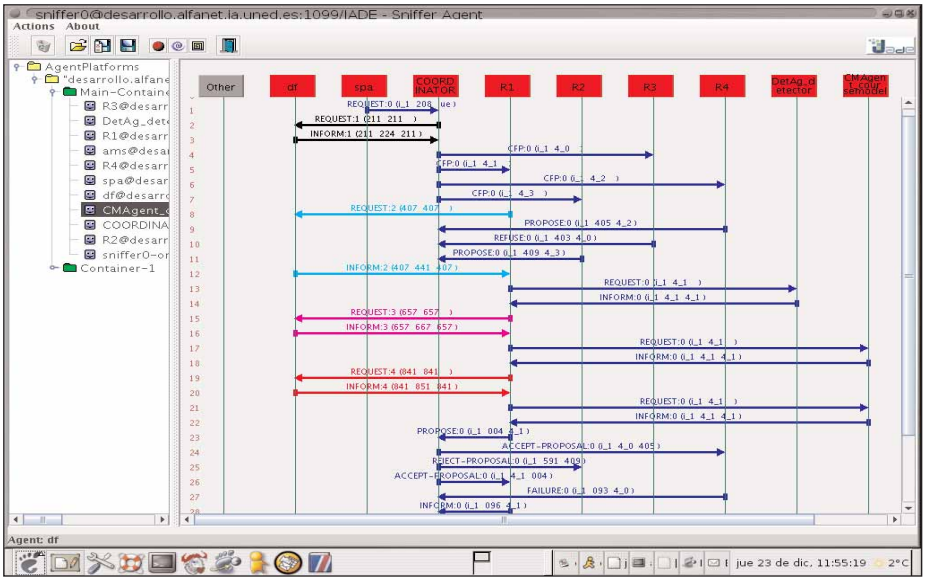
Distribuida (IAD) que, como su nombre indica es la rama de la inteligencia artificial que trata de resolver de manera distribuida sus problemas, aprovechando así las ventajas propias de la programación distribuida: robustez, paralelismo y escalabilidad.

Desde un punto de vista de ingeniería de software este paradigma también supone una evolución de las necesidades de reutilización y encapsulamiento del código. Partiendo de la programación orientada a objetos, el mundo está compuesto por elementos llamados objetos que tienen atributos a los que es posible aplicarles métodos y estos pueden abstraerse a clases y estas clases pueden abstraerse en otras clases de las que heredan métodos y/o atributos o de las que se componen. Sin embargo, este modelo del mundo es incompleto, pues en el mundo también existen agentes con capacidades de aprendizaje y autonomía.

También podemos usar agentes inteligentes para entender mejor el conocimiento o para poder hacer simulación.

Hay bastante software denominado como software de agentes (http://linuxselfhelp.com/HOWTO/AI-Alife-HOWTO-6.html) para GNU/Linux; sin embargo, no siempre se entiende la filosofía de agentes que subyace. La herramienta para desarrollar agentes más extendida y utilizada es JADE (http://jade.tilab.com/) gracias a sus buenas herramientas gráficas, documentación, soporte, licencia LGPL... por desgracia para quienes no queremos caer en la trampa (http://gnu.fyxm.net/philosophy/java-trap.es.html, ver *Mundo Linux* 67) está implementado en java y requiere JDK 1.4 o posteriores.

Para ampliar información acerca de teoría de agentes recomiendo *Multiagent Systems: A Survey from a Machine Learning Perspective* (http://www-2.cs.cmu.edu/afs/cs/usr/pstone/public/papers/97MAS-survey/revised-survey.html). Aunque leer *Adaptive Software* (http://www.norvig.com/adapaper-pcai.html) es un excelente complemento que aclara muchas ideas.



Jade en ejecución.

Tal vez este apartado no debería estar en este artículo, ya que no hay una aplicación práctica a los agentes inteligentes, debido a que es una cuestión más metodológica que facilita, entre otras cosas, poder hacer adaptación.

Reflexiones, conclusiones, referencias y agradecimientos

En este artículo se ha visto que existe una cierta correlación entre software libre e inteligencia artificial debido a que existen multitud de herramientas libres para tratar las diferentes problemáticas que se plantean. En mi humilde opinión no creo que sea casual que el mismo Richard Stallman trabajara en el laboratorio de inteligencia artificial del MIT poco antes de fundar el proyecto GNU. También he tratado de desmitificar ciertos aspectos de la IA que la hacen parecer algo mágico o carente de utilidad real y dejando entrever que es uno de los principales motores de la evolución de la informática. No es casual que las principales áreas de investigación de cualquier universidad estudien temas de IA, como tampoco es casual que las empresas punteras en tecnología tengan muy en cuenta este área de la informática.

En el artículo, primeramente se ha mostrado cómo esos juegos con los que cada día nos divertimos llevan en su motor algoritmos de IA. Después se ha revisado el modo en que la IA nos permite modelar el conocimiento de un experto con herramientas integradas. Finalmente se ha mostrado que el software puede ser mejorado de una manera autónoma y que el enfoque de agentes es bueno para resolver ese problema.

Han quedado bastantes temas que deberán ser tratados en un próximo artículo: planificación, procesamiento del lenguaje natural, sentido común, robótica, computación neuronal... Mientras tanto, os invito a experimentar sobre los temas tratados haciendo uso de la selección de Referencias que se muestran en el cuado adjunto.

Y, para terminar, agradecer a Jesús González Boticario (http://www.ia.uned.es/~jgb/) y Olga Santos sus comentarios y sugerencias.

Referencias

- Dentro del *linux documentation project* encontramos este howto de inteligencia y vida artificial: http://linuxselfhelp.com/HOWTO/AI-Alife-HOWTO.html
- La claridad de ideas de Peter Norvig y la cantidad de código aportado le hacen un sitio imprescindible: http://www.norvig.com
- En Ourproject se está formando un grupo que pretende ser un laboratorio de inteligencia artificial y software libre: http://www.ourproject.org

Sobre el autor

David Arroyo Menéndez (http://davidam.no-ip.info) es Ingeniero Técnico en Informática de Sistemas por la UNED, donde estudia el segundo ciclo de dicha titulación. Actualmente trabaja en el Proyecto Europeo de Investigación Alfanet (http://alfanet.ia.uned.es).

Licencia