Aplicaciones Prácticas de Inteligencia Artificial

David Arroyo Menéndez

En este artículo se mostrarán algunas de las principales aplicaciones para las cuales se utilizan las técnicas de Inteligencia Artificial, partiendo de las técnicas más sencillas y haciendo una breve explicación de su funcionamiento y llegango a aquellas más novedosas y de reciente utilización. Así mismo, se ilustrarán las explicaciones mediante explicaciones de código y de aplicaciones libres disponibles.

1. Introducción

Este artículo ha sido titulado *Aplicaciones Prácticas de Inteligencia Artificial* y esta introducción explicará 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. La definición de Inteligencia Artificial de Rich y Knight es suficientemente simple y precisa como para ser útil a nuestros propósitos, [La IA es el estudio de cómo lograr que las computadoras realicen tareas que, por el momento, los humanos hacen mejor].

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 acercas 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 ahora un repaso para ver cuáles pueden ser esas tareas que, en principio, los humanos realizan mejor que las máquinas:

- Juegos. Durante algunos años los fabricantes de ordenadores demostraban la potencia de su tecnología jugando al ajedrez ó a las 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 ó para otras decisiones que necesitamos de un

conocimiento que no siempre poseemos. Desde hace años los sistemas basados en conocimiento ó 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, ó interfaces gráficas nos sirven para comunicarnos con éstos, 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 conseguimos crear un sistema artificial que simulara nuestro sistema neurológico y lo entrenáramos en un entorno adecuado pueda llegar 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 ó 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.

2. Juegos

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

Para acotar el problema, pensemos en un juego de tablero de 2 jugadores, por ejemplo, las tres 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, ó vacio (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 a partidas victoriosas.

- Otro enfoque sería actuar en cada turno, teniendo en cuenta ciertas reglas estáticas (ej: ciertos if). Un ejemplo de reglas en las tres 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.

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:

Ejemplo 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
```

De este modo, queda brevemente ilustrado el modo en que se pueden aplicar técnicas de Inteligencia Artificial en los juegos.

3. 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 ó 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 ó algún otro mecanismo para inferir nuevos hechos. Pero deberemos plantearnos si a partir de los hechos vamos a ir aplicando reglas (encadenamiento hacia adelante), ó si por el contrario nos interesa responder a una cuestión concreta (ej: ¿ tengo gripe?) e ir satisfaciendo subobjetivos hasta llegar a hechos que demuestren la veracidad ó 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, ó 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 siguiente:

Ejemplo 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> (reset)
CLIPS> (run)


The Engine Diagnosis Expert System

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

De este modo, se prueba un ejemplo de sistema experto cuyo objetivo es solucionar averías con el coche. El sistema va haciendo preguntas acerca de los síntomas que presenta el coche y finalmente otorga un diagnóstico. Os recomiendo que echéis un vistazo al fichero auto.clp.

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

logo de clips

4. 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, ó 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 éste 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 basa en aplicar 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 "sí" 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 ó 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 Guia Campsa (http://www.guiacampsa.com) que utiliza una neurona articial para ordenar el orden de rutas que prefieren los usuarios.

Ahora que se tiene un poco más claro para que 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. Para ello es necesario instalar un intérprete de lisp (si usamos debian y queremos cmucl) **apt-get install cmucl** y ahora se descarga 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:

Ejemplo 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 t
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
```

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, la variable d6 es un ejemplo de día para jugar al tenis ó no: **(classify 'd6 tree)**.

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.
- Russell y Norvig desarrollaron para el libro "Inteligencia Artificial: Un Enfoque Moderno" una buena cantidad de código lisp con una 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 dónde 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.

Manos dibujando de M.C. Escher

5. Agentes Inteligentes

Para Russell y Norvig 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 ó 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 ó 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 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 a 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 ó 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 ó para poder hacer simulación.

Pantallazo de jade en ejecución

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 de Java" (http://gnu.fyxm.net/philosophy/java-trap.es.html) requiere JDK 1.4 ó 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.

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.

Profundidad de M.C. Escher

6. Planificacion

Llamaremos planificación al proceso de búsqueda y articulación de una secuencia de acciones que permitan alcanzar un objetivo. Por ejemplo, si nuestro objetivo es viajar desde un pueblo perdido de Asturias y queremos llegar a Guatemala la secuencia de acciones serían los distintos transportes que se deben tomar para llegar. Otro ejemplo podría ser que tuviéramos un robot en un laberinto y nuestro objetivo fuera sacarle de él; en tal caso, nuestras acciones serían los tramos recorridos en línea recta y los giros dados por el robot.

Para formalizar el problema de la planificación existen 2 notaciones principales: ADL y STRIPS (http://www.rci.rutgers.edu/~cfs/472_html/Planning/STRIPS_plan_472.html). Ambas coinciden en utilizar la lógica para representar estado inicial, objetivo y acciones. También coinciden en que, para

aplicar una acción, es necesario cumplir unas precondiciones y, tras haber ejecutado la acción, habrá provocado unos efectos.

Para obtener la secuencia de acciones, es decir, el resultado al problema de la planificación, existen varios enfoques:

- Búsquedas en el espacio de estados que operan hacia adelante (desde el estado inicial), o hacia atrás (desde el objetivo) aplicando las acciones. También existen heurísticas (estrategias) eficaces que nos ayudan en la búsqueda. Estos enfoques funcionan bien cuando los subobjetivos son independientes.
- Si los subobjetivos no son independientes suele ser buena idea utilizar algoritmos de Planificación de Orden Parcial (POP), que exploran el espacio de planes sin comprometerse con una secuencia de acciones totalmente ordenada. Trabajan hacia atrás, desde el objetivo y añaden acciones para planificar cómo alcanzar cada subobjetivo.
- Otras estrategias prometedoras son el algoritmo GRAPHPLAN (http://www-2.cs.cmu.edu/~avrim/graphplan.html) y el algoritmo SATPLAN (http://www.cs.washington.edu/homes/kautz/satplan/).
- Cuando no se trabaja en mundos cerrados, sino en el mundo real, debemos tener en cuenta varias consideraciones. Muchas acciones consumen recursos: tiempo, dinero, materias primas, etc. por lo que debemos tener en cuenta medidas numéricas. Las redes jerárquicas de tareas (HTN) (http://www.cs.umd.edu/projects/plus/HTN/) permiten tener sugerencias sobre el dominio por parte del diseñador. La información incompleta puede ser manejada mediante planificación que utilice acciones sensoriales para obtener la información que necesita. La planificación multiagente es necesaria cuando existen otros agentes en el entorno con los que cooperar, competir, o coordinarse.

Los problemas y soluciones que se abordan en planificación tienen aplicaciones directas en gestión de tareas (workflow), control de misiones complejas (espaciales, satélites, militares, etc.), turismo (visitas a ciudades, planificar rutas, ...), procesos de enseñanza/aprendizaje, robótica (planificar caminos), ...

Para la problemática de la planificación, las licencias libres han brillado por su ausencia. No obstante, todos los algoritmos se distribuyen con su código, lo cual también es cierto en arquitecturas integradas como PRODIGY (http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/prodigy/Web/prodigy-home.html), muy usadas en planificación.

7. Procesamiento de Lenguaje Natural

Antes de abordar qué es el procesamiento del lenguaje natural (ó lenguaje humano), pensemos por un momento qué es la compresión. Según Rich y Knigth "comprender algo es transformarlo de una representación a otra, en donde la segunda representación se ha elegido para que se corresponda con un conjunto de acciones posibles que podrían llevarse a cabo, y en donde se ha diseñado la correspondencia de forma que para cada suceso se realice una acción apropiada". En definitiva, comprender algo es transformarlo en una representación que nos sea más útil para lograr nuestro objetivo.

¿Para qué podemos querer que una máquina comprenda lo que pone en un texto? Lo primero que se nos puede ocurrir es que sirva para comunicarnos con ella como si de una persona se tratara, pero para ello no solo requeriría comprender nuestras palabras, sino también las emociones con las que las expresamos, lo cual es algo más complejo. La traducción automática desde un lenguaje natural a otro es algo para lo que solo necesitamos comprender el significado neutral de las palabras. Si conseguimos que un ordenador comprenda, también nos va a ayudar en tareas de recuperación de la información; me refiero a buscadores más eficaces que sepan relacionar mejor unas palabras o frases con otras, e incluso independientemente del idioma.

Para realizar estas tareas es necesario comprender un texto; esto es lo que nos proporciona el procesamiento del lenguaje natural. Veamos las fases de las que se compone [Rich y Knight, 1994]:

- Análisis morfológico: Se analizan los componentes de las palabras individuales y se separan de las palabras los constituyentes que no forman parte de ellas, como los símbolos de puntuación
- Análisis sintáctico: Se transforman las secuencias lineales de palabras en ciertas estructuras que muestran la forma en que las palabras se relacionan entre sí. Se pueden rechazar algunas secuencias de palabras si infringen las reglas del lenguaje sobre la forma en que las palabras pueden combinarse. Ej: "niña la come mucho" se rechazaría.
- Análisis semántico: Se asigna significado a las estructuras creadas por el analizador sintáctico. Es decir, se hace una correspondencia entre las estructuras sintácticas y los objetos del dominio de la tarea. Las estructuras en las que no se pueda realizar tal correpondecia se rechazan. Ej: "Las ideas verdes incoloras duermen furiosamente" se rechazaría
- Integración del discurso: El significado de una frase individual puede depender de las frases precedentes y puede influenciar el significado de las frases posteriores. Por ejemplo, la palabra "lo" en "Jaime lo quiso" depende del contexto del discurso, mientras que la palabra "Jaime" puede influenciar el significado de frases posteriores como "Él vive en Madrid".
- Análisis de la pragmática: La estructura que representa qué se ha dicho se reinterpreta para determinar su significado actual. Ej: "¿Sabe qué hora es?" se reintepreta como petición de hora.

Hasta ahora hemos estado hablando del procesamiento del lenguaje escrito, si quisiéramos realizar procesamiento del lenguaje oral, necesitaríamos conocimiento adicional sobre fonología, así como suficiente información adicional para manejar las posibles ambigüedades que pudieran surgir.

Existen varias herramientas libres para tratar el procesamiento del lenguaje natural. Podemos encontrar malaga-bin y mmorph. Sin embargo, creo que para juguetear y/o iniciarse en este campo es más adecuado nltk (http://nltk.sourceforge.net/) (Natural Language Toolkit), escrito en python y con una licencia copyleft de creative commons (http://creativecommons.org/licenses/by-nc-sa/1.0/).

pantallazo de nltk en ejecución

8. Computación Neuronal

Bien, hasta aquí hemos visto que las máquinas pueden imitar comportamientos humanos para un gran número de tareas. De hecho, incluso pueden aplicarse algoritmos de aprendizaje que, para ciertas tareas como el spam, dan muy buenos resultados. La siguiente cuestión entonces es ¿pueden las máquinas pensar tal y como lo hacemos los humanos?

La naturaleza nos ha dotado con un sistema de cómputo basado en una enorme red de neuronas. Si las máquinas pudieran simular tal sistema de cómputo ¿no estarían entonces en condiciones de pensar tal y como lo hacemos las personas?

En primer lugar, pensemos qué es una neurona. Una neurona está formada por el cuerpo celular y diferentes prolongaciones: el axón, por el que transitan los impulsos nerviosos o potenciales de acción desde el cuerpo celular hacia la siguiente célula, y la/s dendritas, con número y estructura variable según el tipo de neurona, y que transmiten los potenciales de acción desde las neuronas adyacentes hacia el cuerpo celular. Se podría decir que las dendritas son entradas y el axón es la salida.

Ahora bien, ¿cómo se decide si tras recibir entradas se produce o no una salida, y qué salida se produce? Pues bien, la salida es la suma ponderada de las entradas seguida de una función umbral. Esto puede parecer un diodo: si el potencial de las entradas alcanza un cierto valor, entonces se produce la salida, y sino no. Sin embargo, difiere en ser ponderado. Se refiere a que no todos los valores de las entradas van a tener la misma importancia de cara a decidir si se produce o no la salida.

Si nuestra neurona no es más que un diodo con entradas ponderadas, entonces juntando unos diodos con otros en forma de red ¿hará el tipo de procesos que realizamos los humanos como, por ejemplo, aprender? Es curioso que para aprender solo es necesaria una neurona, y el aprendizaje consiste en modificar el valor de los pesos (la importancia de cada entrada) aplicando un algoritmo a nuestro valor de salida. ¿Y cual será ese algoritmo? Un psicólogo llamado Donald O. Hebb enunció una regla que dice que "las conexiones que unen diferentes neuronas que se encuentran activas en un instante dado se fortalecen". Se refiere a que si la neurona está transmitiendo, las entradas que est´n activas tendrán más importancia de la que ya tenín. Y esto es más ó menos la esencia de la computación neuronal, especialmente la no supervisada (que no requiere de un tutor), y esta es la que nos encontramos en la naturaleza.

Pero en la práctica las redes neuronales artificiales no han creado en las máquinas comportamientos similares a los de las personas. Sin embargo, lo cierto es que los problemas que tradicionalmente son más complejos para las máquinas (visión artificial, reconocimiento del habla, etc.), se resuelven mejor utilizando redes neuronales que mediante otras perspectivas simbólicas.

La computación neuronal tiene como tarea genérica la clasificación. Por ejemplo, pensemos que queremos averiguar en qué imágenes está Wally y en qué im´genes no. Nuestras entradas serían las intensidades RGB de cada pixel de cada una de las imágenes, y tendría una única salida que nos diría si está o no.

Las redes neuronales tienen un modelo inherentemente distribuido, puesto que un procesador puede simular una neurona. Así mismo, son autoprogramables. La programación pasa por elegir un número de neuronas, las conexiones entre las mismas, un número de salidas, un número de entradas y una configuración inicial de pesos. Después bastaría con alimentarla correctamente y, si el aprendizaje es supervisado, darle la adecuada realimentación humana

Como siempre, para profundizar recomiendo jugar con el software libre disponible. Lo que he encontrado ha sido Fast Artificial Neural Network Library (fann) (http://fann.sourceforge.net/intro.html), Genesis (http://www.genesis-sim.org/GENESIS/) y, aunque no totalmente libre, también creo importante destacar SNNS (http://www-ra.informatik.uni-tuebingen.de/SNNS/).

Finalizar diciendo que, si bien los algoritmos de redes neuronales no son otra cosa que algoritmos de aprendizaje automático, debido a las implicaciones filosóficas que tienen he decidido tratarlo aparte.

Pantallazo de fann en ejecución

9. Robótica

[Russell y Norvig, 2004]Los robots son agentes físicos que realizan tareas mediante la manipulación física del mundo. Para ello se dotan de efectores, como pinzas, ruedas, brazos mecánicos, etc. También se equipan de sensores que les permiten percibir el entorno, como visores, sistemas de ultrasonidos, giroscopios, etc.

El interés de la IA en la robótica se centra en los agentes inteligentes que manipulan el mundo físico. No obstante, también es un campo de gran interés para aplicar otras ideas de IA como planificación o visión artificial (aplicar algoritmos de aprendizaje para el reconocimiento de formas).

Las áreas de aplicación de la robótica son múltiples: industria, agricultura, transporte, entornos peligrosos (donde no debería haber humanos), exploración (ej: viajes al espacio), salud (ej: cirugía gracias a la precisión milimétrica), entretenimiento, ...

Para experimentar en este área respetando nuestras libertades es posible encontrar programas divertidos para la simulación de comportamientos robóticos, como por ejemplo gnurobots (http://directory.fsf.org/GNU/gnurobots.html) o robocode (http://robocode.alphaworks.ibm.com/home/home.html). Este tipo de programas pueden llegar a dar mucho juego en la IA, recordad que los bots del quake han sido materia de tesis doctoral (http://www.kbs.twi.tudelft.nl/Publications/MSc/2001-VanWaveren-MSc.html).

También es interesante salirnos un poco del software y meternos en algún proyecto de hardware abierto, como Microbótica. (http://www.microbotica.es/web/ha.htm))

Imagen de un robot de juguete tomado de http://www.worstdomainever.com

10. Conclusiones

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 ó 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 sea en 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 permite modelar el conocimiento de un experto con herramientas integradas. Seguidamente, se mostró que el software puede ser mejorado de una manera autónoma y que el enfoque de agentes es bueno para resolver ese problema. Seguidamente se han visto otras cuestiones: Planificación, procesamiento de lenguaje natural, redes neuronales y robótica son áreas que permiten entendernos mejor como humanos y que al mismo tiempo mejoran la productividad de las sociedades.

De hecho, podemos decir que la inteligencia artificial aporta soluciones para casi todas las cuestiones que nos caracterizan como humanos e inteligentes, sin embargo, aún dista bastante para que realice dichas tareas de la misma manera y con la misma eficacia que lo hacen las personas. Sin embargo, las soluciones aportadas para resolverlos tienen aplicaciónes interesantes en múltiples áreas de nuestra vida cotidiana que nos hacen la vida más fácil.

Espero que este artículo ayude a despertar el interés por la IA entre los usuarios de software libre y, a que la comunidad que existe alrededor de la inteligencia artificial se interese aún más por el software libre.

Metamorfosis 2 de M.C. Escher

11. Referencias

Algunas referencias para profundizar en el aprendizaje de la inteligencia artificial.

- Rich, E. y Knight K. (1994). Inteligencia Artificial. Madrid: McGraw Hill
- Russell, S. J. y Norvig P. (2004). Inteligencia Artificial. Un Enfoque Moderno. Segunda ed. Madrid: Pearson Educación S.A.

- Howto de inteligencia y vida artificial (http://linuxselfhelp.com/HOWTO/AI-Alife-HOWTO.html)
- Wikipedia (http://www.wikipedia.org/)
- Página personal de Peter Norvig (http://www.norvig.com/)
- Dentro del linux documentation project encontramos este howto de inteligencia y vida artificial (http://linuxselfhelp.com/HOWTO/AI-Alife-HOWTO.html)

12. Agradecimientos

Este artículo habría sido muy distinto sin las aportaciones de un buen número de miembros del grupo adenu (http://adenu.ia.uned.es/adenu/) y de Ramiro Pareja cuyas sugerencias y apoyos fueron muy valiosas.

Un afectuoso agradecimiento a todo el equipo de Mundo Linux (http://digital.revistasprofesionales.com/mundolinux/) por la oportunidad de escribir este artículo y de poder hacerlo aplicando una licencia libre y muy especialmente a Margarita Padilla.

Finalmente, agradecer a los lectores del mismo a quienes deseo que hayan disfrutado tanto leyéndo como yo escribiéndolo.