MONOGRAFÍA: COMPUTACIÓN EVOLUTIVA

Una herramienta basada en algoritmos genéticos para obtener un clasificador jerárquico en aprendizaje supervisado

José Riquelme, Jesús Aguilar y Miguel Toro

Departamento de Lenguajes y Sistemas Informáticos
Facultad de Informática y Estadística.
Avd. Reina Mercedes s/n. 41012 Sevilla
e-mail: {riquelme, aguilar, mtoro}@lsi.us.es

Resumen

Presentamos una herramienta para obtener un sistema clasificador a partir de una base de datos etiquetada. El resultado que proporciona es un sistema de reglas jerárquico para dividir los datos en n-ortoedros. Esta jerarquía significa que las reglas obtenidas deben de aplicarse en un orden concreto, es decir, un punto será clasificado por la regla i-ésima sólo si no cumple las premisas de las i-1 reglas anteriores. El motor de búsqueda utilizado es un algoritmo evolutivo con codificación real. Esto presentará el inconveniente de un mayor tiempo de computación de la solución. Pero, por otra parte, proporcionará flexibilidad para obtener diversos sistemas clasificatorios con valores distintos para los parámetros opuestos de todo sistema de este tipo: número de reglas y tasa de error. Se presentan los resultados de aplicación a diversos ficheros de datos, siendo los resultados bastante buenos, tanto en relación a las tasas de error obtenidas como, sobre todo, al número de reglas necesario para ello.

Palabras clave: aprendizaje automático, algoritmos genéticos.

1. Introducción

De la gran variedad de herramientas que realizan aprendizaje supervisado (redes neuronales, vecinos más cercanos, árboles de decisión,...), el programa C4.5 (Quinlan 93) es uno de los más extendidos por varias cualidades, entre otras, su fácil utilización, el mínimo tiempo de computación, la fácil comprensión del sistema resultante... El principal inconveniente que se suele señalar del C4.5 es que la división del espacio de datos se hace en n-ortoedros, lo que motiva que para bases de datos con regiones no fácilmente aproximables por hipercubos, el número de reglas resultante puede ser excesivo. Sin embargo, pueden existir casos de ficheros de datos que aún teniendo los datos de la misma clase agrupados en hipercubos, el C4.5 daría lugar a un número excesivo de reglas. En la figura 1 podemos ver un ejemplo en dos dimensiones. A la izquierda se presenta un fichero con dos parámetros representados en abscisas y ordenadas y dos posibles clases: círculos blancos o negros. La clasificación más inmediata de estos datos es señalar que hay tres rectángulos de puntos negros y el resto son círculos blancos, lo que daría lugar a cuatro reglas. Sin embargo, el C4.5 haría una partición del espacio como la que se observa a la derecha, que además de un número mayor de reglas (13) es menos comprensible que la presentada a la izquierda.

Otra desventaja que se puede aducir del C4.5 es que sólo proporciona dos posibles sistemas de reglas: un árbol completo y otro podado. En algunos casos podría ser útil la posibilidad de que el usuario estableciera una cota máxima de error y que el sistema proporcione un clasificador con el mínimo número de reglas para el que se alcanza esa tasa de error. O viceversa, es decir, el usuario establece el

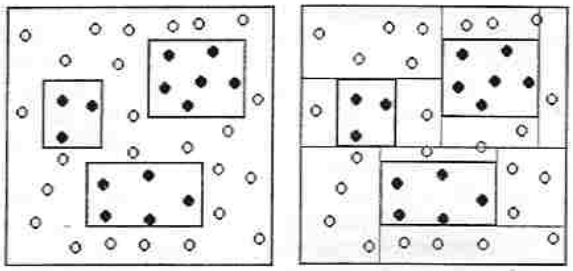

Figura 1. Clasificación jerárquica (izq.) y C4.5 (der.)

número de reglas que desea y el sistema proporciona un conjunto de reglas con la mínima tasa de error posible para ese número.

El primer inconveniente se puede resolver mediante el establecmiento de un *sistema de reglas jerárquico*, es decir, las reglas deben aplicarse en un orden concreto. La estructura de estas reglas será de la forma:

Si premisa entonces clase
  sino Si premisa entonces clase
    sino Si premisa entonces clase ....
      .....
        sino clase.

La ventaja de este sistema es, sobre todo, la reducción en el número de reglas necesarias para clasificar correctamente determinadas bases de datos como la presentada en la figura 1.

El segundo inconveniente puede solucionarse si el sistema de reglas es obtenido mediante un proceso de búsqueda no determinista que permita configurar los dos principales parámetros de todo sistema clasificatorio: complejidad (en este caso, número de reglas) y precisión (en este caso, % de error sobre el fichero de entrenamiento o de test). Esto presenta dos desventajas respecto del C4.5, la primera es el tiempo de computación y la segunda que la solución no es única. La primera significa que, mientras el C4.5 tarda escasos segundos en hallar sus árboles de decisión, nuestra herramienta tardará varios minutos (dependiendo del tamaño del fichero). Desde nuestro punto de vista, la segunda objeción podría ser más importante si los resultados de diferentes ejecuciones dieran lugar a soluciones muy distintas. Como se verá en el epígrafe de aplicación, los resultados obtenidos minimizan este posible inconveniente. Para ello era necesario un algoritmo de búsqueda que sobre espacios complejos diera soluciones satisfactorias. El método que proponemos es un algoritmo genético (Goldberg 89) con codificación real de los individuos (Michalewicz 94).

El objetivo es diseñar un sistema que, mediante el uso de los algoritmos genéticos, sea capaz de extraer un conjunto de reglas de decisión a partir de una base de datos ya clasificada. Las herramientas más conocidas basadas en esta técnica de optimización son GABIL (De Jong 93) y GIL (Janikow 93). Ambas tienen en común una codificación binaria de los individuos, presentando reglas para parámetros con valores discretos o continuos discretizados en intervalos de clase. La diferencia entre ambos radica en que mientras un individuo en GABIL es una sola regla, GIL codifica varias reglas en un individuo. Nuestra principal aportación en este caso es trabajar con valores continuos y denotar las premisas por la pertenencia de un parámetro a intervalos de clase no prefijados de antemano.

Por último, señalar que en (Aguilar 97) se presentó un antecesor de esta herramienta que implementaba un algoritmo genético con codificación binaria y que proporcionaba un sistema de reglas sin posibilidad de error, es decir, el motor de búsqueda no permitía reglas con algún error, lo que penalizaba en exceso la simplicidad del sistema de reglas, objetivo prioritario de este trabajo.

2. Descripción del sistema

2.1 El entorno y su codificación

La información del entorno vendrá dada por un fichero de datos con una estructura similar a la de cualquier herramienta de aprendizaje supervisado. Es decir, una tupla formada por un número indeterminado pero fijo de parámetros con valores reales que tendrá asociada una clase discreta que

define el tipo. El número de clases posibles es también indeterminado. El algoritmo genético utiliza codificación real, es decir, un individuo está formado por una n-tupla de reales, incluyendo la clase que toma valores reales comenzando por 1.0 y sumando una unidad a las sucesivas. Nuestro objetivo es que la premisa de cada regla venga dada por la pertenencia de un dato a un n-ortoedro definido por un límite inferior y superior para cada parámetro. Para ello la representación de un individuo toma la siguiente forma:

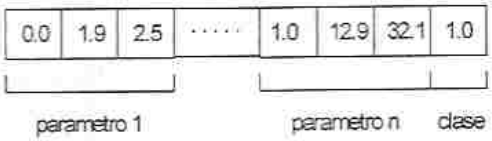

Figura 2. Codificación de los individuos

Para cada parámetro se define una terna compuesta por:

* Un valor perteneciente al conjunto {0.0,1.0,2.0}, que identifica el tipo de operación sobre los dos valores siguientes:

  - 0.0 equivale a "Si p1≤1.9 ...".
  - 1.0 equivale a "Si p1≥2.5 ...".
  - 2.0 equivale a "Si 1.9≤p1≤2.5 ...".

* Los dos valores siguientes son las cotas para cada operación sobre el parámetro. Con el significado siguiente: si el operador es ≤ sólo tiene sentido el primer valor, si el operador es ≥, el segundo valor es el usado y en el tercer caso los dos valores indican la cota por arriba y abajo del parámetro.

* El último valor identifica a la clase. Existirán tantos valores posibles como clases haya, es decir, si hay 5 clases el valor pertenecerá al conjunto {0.0, 1.0, 2.0, 3.0, 4.0}.

2.2 El algoritmo

El algoritmo del sistema se esquematiza de la siguiente forma:

```
inicializar
repetir
    evolución
    seleccionar_regla
    adaptar_entorno
hasta condición_de_parada
```

En *inicializar* se calcula número de datos del fichero de entrenamiento, el número de parámetros de cada dato y el numero de clases, calculándose el máximo y mínimo de cada parámetro. El proceso *evolución* es un algoritmo evolutivo clásico que se encarga de buscar una regla que con la codificación anterior, minimice una determinada función de ajuste sobre el fichero de entrenamiento. El módulo *seleccionar_regla* escoge el mejor individuo del proceso evolutivo, transformándolo en una regla. Esta regla es utilizada en *adaptar_entorno* para eliminar del fichero de entrenamiento los datos que cumplan su premisa (aunque no cumplan la conclusión). De esta manera, el fichero de entrenamiento es reducido para la siguiente iteración. La *condición_de_parada* puede ser hasta que el fichero de entrenamiento haya sido cubierto en su totalidad, hasta un número determinado de reglas o hasta una determinada precisión.

2.3 Operadores genéticos

Como se ha señalado el módulo *evolución* presenta un algoritmo evolutivo cuyas principales características son: elistismo, en el sentido de que el mejor de cada generación es replicado en la siguiente y una serie de hijos son obtenidos mediante copias de los padres, elegidos aleatoriamente pero en función de su bondad (método de la ruleta de la fortuna). El resto de la siguiente generación se obtiene mediante cruces que son sometidos con una determinada probabilidad a mutación.

Se utilizan dos tipos de cruce que se aplican alternativamente dependiendo de una probabilidad. Por un lado, se ha conservado el cruce aleatorio multipunto (Aguilar 97) modificado para una codificación real. Por otro lado, se ha utilizado un cruce real diseñado ex profeso que utiliza tres tipos ponderados en función de la importancia que puedan tener en la solución, los cuales son seleccionados aleatoriamente. Los tipos son:

* cruce segmentado intermedio: obtiene un valor aleatorio en el segmento definido por los dos valores que ocupan la misma posición en cada individuo seleccionado (se aplica en el 40% de los casos, aproximadamente).

* cruce segmentado forzado al mínimo: obtiene un valor aleatorio en el segmento definido por el límite inferior del rango y el menor de los dos valores que ocupan la misma posición en cada individuo

seleccionado (se aplica en el 30% de los casos, aproximadamente).

* cruce segmentado forzado al máximo: obtiene un valor aleatorio en el segmento definido por el mayor de los dos valores que ocupan la misma posición en cada individuo seleccionado y el límite superior del rango (se aplica en el 30% de los casos, aproximadamente).

La mutación hace crecer la región cubierta por el individuo para que pueda abarcar más puntos de la base de datos. Dos tipos de mutación se han utilizado:

* mutación incremental: si se trata del valor derecho del parámetro se incrementa en una cantidad pequeña (1% del rango del parámetro); y si se trata del izquierdo, se decrementa en una cantidad similar.

* mutación forzada al límite: con una probabilidad muy pequeña (5% del total de mutaciones), se mutará a uno de los límites inferior o superior dependiendo, del mismo modo, de si se trata del valor izquierdo o derecho del parámetro, respectivamente.

2.4 Función de adaptación

La función de adaptación o bondad f que el algoritmo evolutivo minimiza para cada individuo i viene dada por el esquema

$$f(i) = \frac{1}{1 + ND - g(i)}$$

donde ND es el número de datos del fichero de partida (variante en cada adaptación) y g se calcula para cada individuo penalizando el número de errores y premiando el de aciertos sobre los datos que van quedando después de las sucesivas adaptaciones. Por último, a la función g se le añade un sumando que representa el tanto por uno de volumen que el individuo abarca respecto del volumen total de la base de datos original. Con ello, pretendemos premiar a los individuos que, teniendo igual valor de ajuste, abarquen más espacio, pues éstos encontrarán nuevos puntos que cubrir.

2.5 Factor de relajación

Las bases de datos utilizadas como entrenamiento, normalmente, no presentan zonas claramente diferenciadas en regiones n-ortoédricas, con lo cual, la obtención de un sistema de reglas totalmente coherente implica un elevado número de reglas. El sistema que presentamos es capaz de obtener un sistema de reglas totalmente coherente (sobre los datos de entrenamiento), sin embargo, a veces, es aconsejable reducir el número de reglas. Esto acarrea un decremento en la tasa de coherencia del sistema de reglas, aunque el número de reglas sea bastante inferior. Por ejemplo, si la base de datos presenta una distribución de las clases en regiones difíciles de inscribir por hipercubos, la propiedad anterior se hace casi necesaria, pues es prácticamente imposible clasificar el espacio en n-ortoedros, o en el mejor caso obtener un número de reglas excesivo. De esta manera, si se consiguiera un sistema de reglas de cardinalidad menor a pesar de cometer algunos errores, podríamos pensar que hemos ganado claridad en el modelo en lugar de perder fiabilidad en el sistema de reglas. Por tanto, puede ser interesante introducir un factor de relajación que permita cometer algunos errores para reducir el número de reglas. El factor de relajación f_r será un porcentaje del número de aciertos en cada momento, de forma que, para un individuo que tiene en un instante dado k aciertos, se permite que cometa $\frac{f_r * k}{100}$ errores de clase, es decir, la cantidad de puntos que pueden ser de clase diferente perteneciendo a la región que el individuo abarca. Por ejemplo, si se permite un factor de relajación del 10%, un individuo que cubra 83 puntos de una determinada clase podrá cometer 8 errores como máximo, o sea, 8 puntos de clase diferente a la primera. Ese coeficiente de relajación se comporta como una cota superior del error cometido por el sistema de reglas sobre la base de datos de entrenamiento.

Teniendo en cuenta que el sistema permite limitar el número de reglas que se desee obtener y también puede fijarse una cota superior del error, dos criterios posibles están disponibles a la hora de encontrar el sistema de reglas adecuado. Así el usuario puede obtener un sistema de reglas para una base de datos especificando o bien el número máximo de reglas que desea; o bien, especificando el porcentaje máximo de error que pretende cometer sobre los datos de entrenamiento.

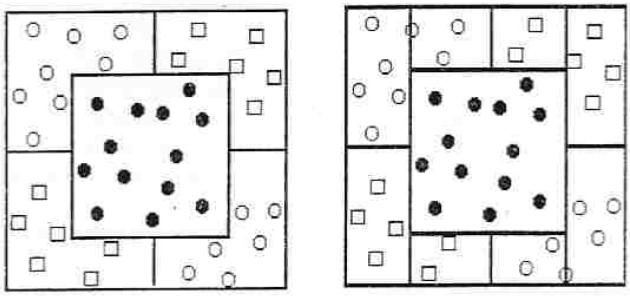

Figura 3. Base de datos Ladrillos

Con ésto se consiguen resultados muy satisfactorios, pues el usuario controla directamente la tasa de coherencia que desea obtener. Por ejemplo, si se obtiene con un f_r=10 una tasa de coherencia del 80% con 23 reglas, el usuario puede decidir acerca del número de reglas a obtener (si son muchas se aumenta el factor de relajación); o bien, puede incidir en la tasa de coherencia (si es baja, se reduce el factor de relajación). Haciendo algunas pruebas, se puede conseguir un sistema de reglas acorde con las necesidades del usuario.

3. Aplicacion

En primer lugar, se ha aplicado nuestra herramienta a dos ficheros de datos creados ex profeso para mostrar las debilidades del C4.5, incluso para encontrar regiones ortoédricas. Posteriormente se ha aplicado a las bases de datos estándares usadas en aplicaciones de aprendizaje y clasificación del UCI Repository (Murphy y Aha 94). El método utilizado para la comparación de los resultados aportados por C4.5 y nuestra herramienta es la validación cruzada. Para ello, se han obtenido aleatoriamente cinco ficheros de entrenamiento y cinco ficheros de test, en una proporción de 70/30, respectivamente, sobre la base de datos inicial.

A partir de ellos, los resultados incluidos en las siguientes tablas son las medias aritméticas de las tasas de aciertos y número de reglas de la aplicación del conjunto de reglas obtenido mediante el aprendizaje de cada fichero de entrenamiento con respecto de su fichero de test.

3.1 Ficheros ex profeso

En la figura 3 presentamos una base de datos (que denominamos ladrillos) con dos parámetros y tres etiquetas distribuidas de forma que si la primera regla recoge la información del cuadrado central cinco reglas jerarquizadas son suficientes para cubrir toda la base de datos (izq). Por contra, herramientas como el C4.5 necesitarían al menos 9 reglas (dcha).

Otro ejemplo más extremo aún es el que denominamos cajones, pues los tipos se encuentran distribuidos de forma alternativa en cuadrados inscritos. Un sistema de reglas jerarquicas necesita sólo 5 reglas mientras que una distribución en cuadrados daría lugar a no menos de 17 reglas.

BASE DE DATOS	C4.5		COGITO	
	TC	NR	TC	NR
LADRILLOS	92.1	9	93.4	5
CAJONES	90.2	27	90.8	5

3.2 Ficheros del UCI Repository

A continuación se presenta una tabla con los resultados obtenidos para algunas de las bases de datos más extendidas para medir la validez de técnicas de aprendizaje. Se comparan éstos con los que ofrece C4.5. En los dos casos se indica el número de reglas obtenido y la tasa de coherencia utilizando los ficheros de entrenamiento para producir el conjunto de reglas y los ficheros de test para producir la tasa de aciertos. La proporción entre el tamaño de los ficheros de entrenamiento y test sigue siendo 70%/30% respecto de la base de datos original. Para nuestra herramienta se ha incluido el número total de reglas producido, aunque algunas de ellas no tienen efecto sobre el fichero de test y podrían eliminarse.

BASE DE DATOS	C4.5		COGITO	
	TC	NR	TC	NR
IRIS	93.6	4.4	96.1	3.4
VINOS	92.8	5.0	94.2	4.0
PIMA	71.6	77.6	75.6	6.8
CANCER	94.8	13.8	96.4	2.4

La diferencia entre los resultados ofrecidos por C4.5 y nuestra metodología es notoria, sobre todo, en

cuanto al número de reglas. Así hemos podido obtener una tasa de coherencia superior a la de C4.5 con el número de reglas que éste proporciona. Nótese que los casos de las bases de datos Pima y Cáncer de mama son verdaderamente sorprendentes pues, en la primera, el número de reglas que C4.5 produce es muy superior; y, en la segunda, teniendo en cuenta que esta base de datos sólo tiene dos clases, los resultados son excelentes.

En la siguiente tabla se puede apreciar la variación de la tasa de coherencia (tomando la base de datos completa como fichero de entrenamiento) respecto del número de reglas obtenidas para la base de datos PIMA.

PIMA	5 R	6 R	7 R	9 R	11 R	12 R	13 R
TC	81.1	81.5	83.9	85.5	86.9	87.8	88.5

Es interesante anotar que, si adoptamos el método de reescritura como medida de calidad, se puede obtener la tabla anterior para cualquier base de datos, cuya información es de especial interés pues indica a partir de qué momento un aumento en el número de reglas deja de ser significativo respecto del porcentaje de aciertos.

Respecto a los tiempos de computación, las ejecuciones de nuestro método se han realizado en una SUN SPARC 1000E y, lógicamente, el tiempo depende sobre todo del número de datos del fichero de entrenamiento y del número de reglas necesario. Para 200 individuos y 200 generaciones encontrar las trece reglas para PIMA puede ocupar unos doce minutos mientras que las dos reglas para CANCER se obtienen en unos dos minutos.

El principal inconveniente de los señalados en la introducción podría ser la posible diversidad de soluciones, al depender la técnica de una búsqueda probabilística. Los resultados mostrados en las tablas corresponden a la mejor de las ejecuciones posibles, sin embargo, realizadas 20 pruebas para hallar 13 reglas para la base de datos PIMA el intervalo de coherencia (sobre el fichero de entrenamiento) de las soluciones encontradas fue de 87.1 a 88.5, lo que, desde nuestro punto de vista, limita el alcance de este inconveniente.

4. Conclusiones

Presentamos una herramienta de aprendizaje supervisado para la clasificación de un fichero de datos en n-ortoedros. El sistema proporciona un conjunto de reglas jerarquizado donde la premisa de cada regla viene dada por la pertenencia de un dato a un n-ortoedro con dimensiones que el algoritmo encuentra. Dos cualidades diferencian a COGITO de otros sistemas: la reducción importante del número de reglas frente a herramientas como C4.5 y, por otro lado, la flexibilidad para que el usuario construya el clasificador estableciendo límites al número de reglas o cotas del error máximo permitido.

Referencias

Aguilar, J., Riquelme, J. y Toro, M. COGITO: Un Sistema de Autoaprendizaje Basado en Algoritmos Genéticos. Actas de las III Jornadas de Informática. pp. 79-88. El Puerto de Santa María, 1997. ISBN 84-8498-765-5.

De Jong,K.A. Using Genetic Algorithms for Concept Learning. Machine Learning, 93.

Janikow, C.Z. A Knowlege-Intensive Genetic Algorithm for Supervised Learning. Machine Learning, 93.

Michalewicz, Z. Genetic Algorithms + Data Structures = Evolution programs. Second Edition, Springer-Verlag, 1994.

Murphy, P. y Aha, D.W. UCI repository of machine learning databases. Dept. of Information and C.S. University of California, Irvine, 1994

Quinlan, J.R. C4.5: Programs for Machine Learning, Morgan Kaufmann Pub., 1993.