Obtención de reglas de clasificación: Algoritmo PRISM

Hristo Ivanov Ivanov
Alberto Lorente Sánchez

Índice

Introducción. ¿Qué es PRISM?

PRISM es un algoritmo de clasificación propio de las técnicas de minería de datos propuesto por Jadzia Cendrowska en 1987 [1].

Surge como respuesta a los árboles de clasificación generados por algoritmos como el ID3. Estos árboles a menudo son incomprensibles o difíciles de manejar.

El algoritmo PRISM genera Reglas de conocimiento modulares del tipo:

$$a_2 \text{ ^ } b_1 \text{^ } c_3 \text{^ } d_2 \text{-> } r_2$$

¿Qué es una regla?

- Pequeños fragmentos de conocimiento independientes
- Fáciles de entender. Autoexplicativas.
- Estructura común:

if A then B

La ejecución de las reglas tiene importancia:

- Ordenadas: Las reglas son una lista de decisión
- Desordenadas: Las reglas pueden superponerse y llegar a distintas conclusiones para la misma instancia

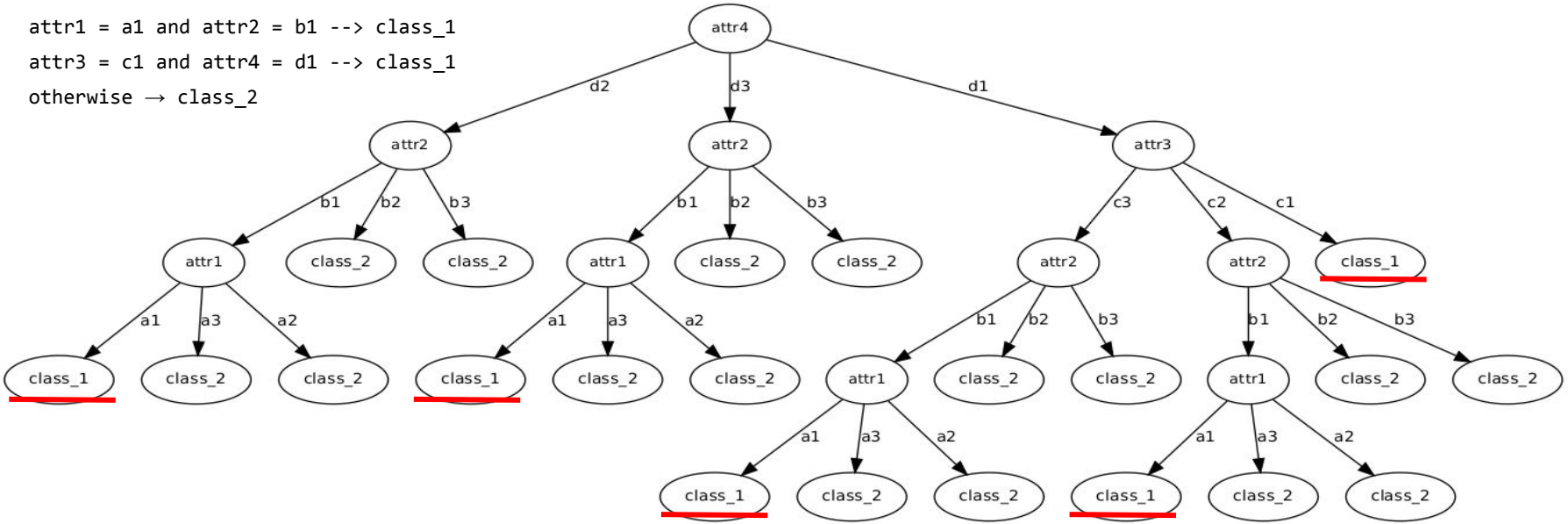
“Correctividad” de una regla

- Un set completo
  - Un set de reglas es correcto si para cada posible instancia hay al menos una regla que la explique.
  - Para que un set de reglas sea completo es necesario que el conjunto de entrenamiento sea completo. Sino las reglas podrán ser incompletas.
- Misclassifies
  - Una regla puede clasificar de forma incorrecta una instancia tanto si es muy general como si es muy específica. Si es muy general puede fallar al clasificar ciertas instancias mientras que la regla específica puede no utilizarse hasta la determinación de un atributo irrelevante.
  - Una regla “correcta” es aquella que referencia a todos los atributos relevantes para la clase.

Problemas con los árboles de clasificación

En la imagen podemos ver el árbol que el algoritmo ID3 genera para las siguientes reglas.

```
attr1 = a1 and attr2 = b1 --> class_1
attr3 = c1 and attr4 = d1 --> class_1
otherwise → class_2
```




```
attr3 = c1 and attr4 = d1 --> class_1
attr1 = a1 and attr2 = b1 and attr3 = c2 and attr4 = d1 --> class_1
attr1 = a1 and attr2 = b1 and attr3 = c3 and attr4 = d1 --> class_1
attr1 = a1 and attr2 = b1 and attr4 = d3 --> class_1
attr1 = a1 and attr2 = b1 and attr4 = d2 --> class_1
```

Problemas con los árboles de clasificación

El algoritmo ID3 añade atributos no relevantes a las reglas. Esto tiene consecuencias negativas:

- ➔ Demanda de información innecesaria.
- ➔ Difícil de interpretar y explicar.

Este problema surge porque ID3 encuentra el atributo que mayor entropía aporta y divide el conjunto de datos según los valores de dicho atributo. El problema se presenta cuando:

- ➔ Un atributo es muy relevante para solo una clase específica.
- ➔ Un atributo es muy relevante para solo uno de sus valores.

```
attr1 = a1 and attr2 = b1 --> class_1
attr3 = c1 and attr4 = d1 --> class_1
otherwise → class_2
```

```
attr3 = c1 and attr4 = d1 --> class_1
attr1 = a1 and attr2 = b1 and attr3 = c2 and attr4 = d1 --> class_1
attr1 = a1 and attr2 = b1 and attr3 = c3 and attr4 = d1 --> class_1
attr1 = a1 and attr2 = b1 and attr4 = d3 --> class_1
attr1 = a1 and attr2 = b1 and attr4 = d2 --> class_1
```

Solución al problema de ID3

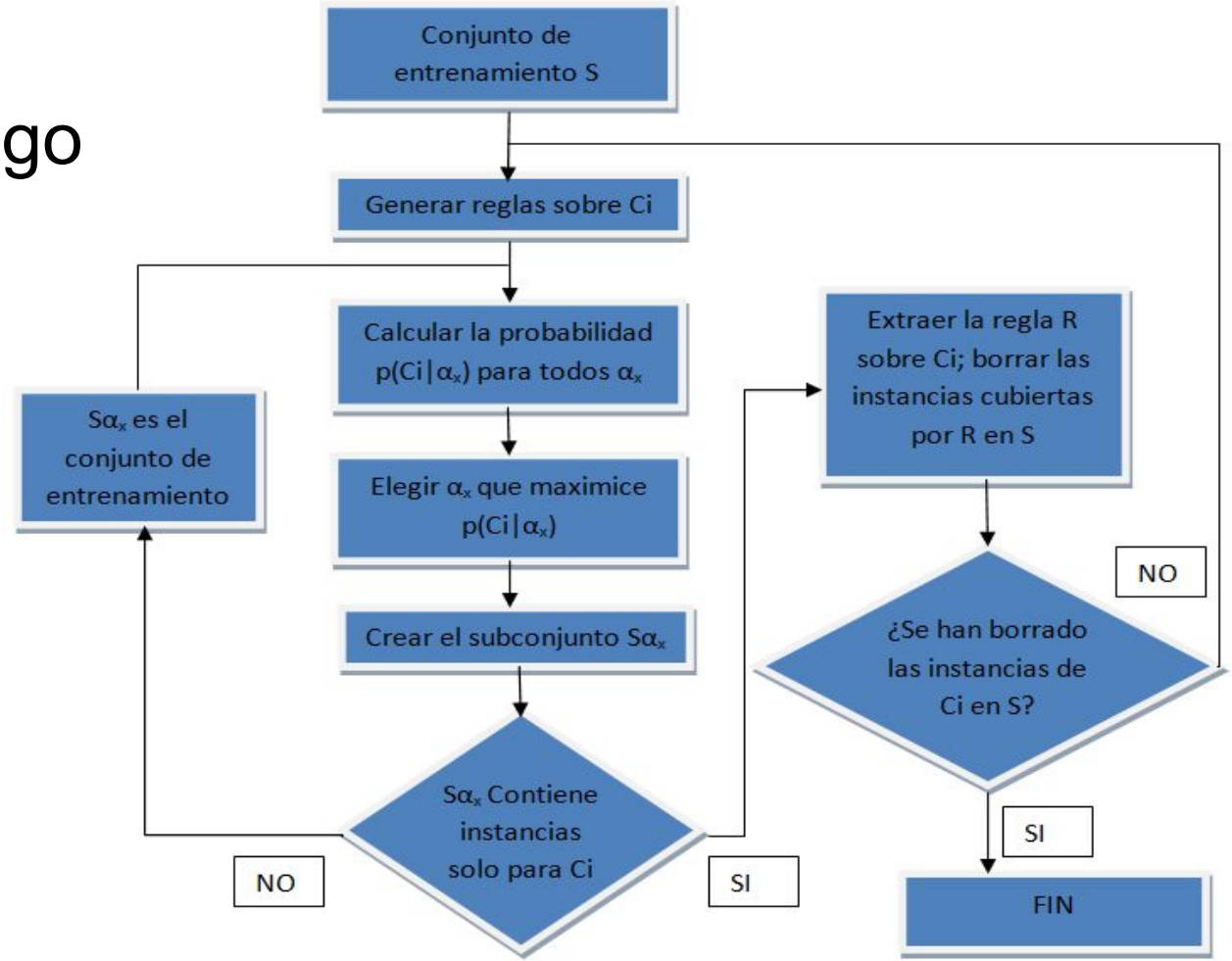
Existen varias modificaciones sobre ID3 cuyo objetivo es resolver los problemas anteriormente descritos. La mayoría de estas se centran en identificar ramas comunes e intentar simplificar el árbol de decisión.

El algoritmo PRISM propone una solución alternativa a los árboles de clasificación. **ID3 busca el atributo que maximiza la ganancia de información media para todas las clases,** en otras palabras, el atributo que produce los subconjuntos cuya variedad de clases es la mínima. La solución que **PRISM** propone es **buscar la pareja atributo-valor que maximiza la ganancia de información para una clase en concreto**.

Siendo α_x una pareja atributo-valor y C_i una clase, PRISM busca maximizar la probabilidad de ocurrencia de C_i sabiendo α_x , **$p(C_i|\alpha_x)$.**

$$p(C_i|\alpha_x) = (\text{ Número de instancias } C_i \text{ }) / S\alpha_x$$

PRISM
Pseudocódigo



PRISM Código Python

```
Parámetros:
 inst:     Lista con las instancias.
 attr_dic: Diccionario con los valores de cada atributo.
 classes:  Lista de las posibles clases.
Comentarios:
 1. Generamos reglas para cada clase, 'cl', en 'classes'.
 2. Empezamos haciendo una copia de las instancias.
 3. Dejamos de generar reglas para la clase 'cl'
    al llegar a un 'conj' sin instancia de la clase.
 4. Invocamos a 'prism_inner' que genera una regla.
 5. Guardamos esa regla
 6. Eliminamos todas las instancias cubiertas por la nueva
regla.
```

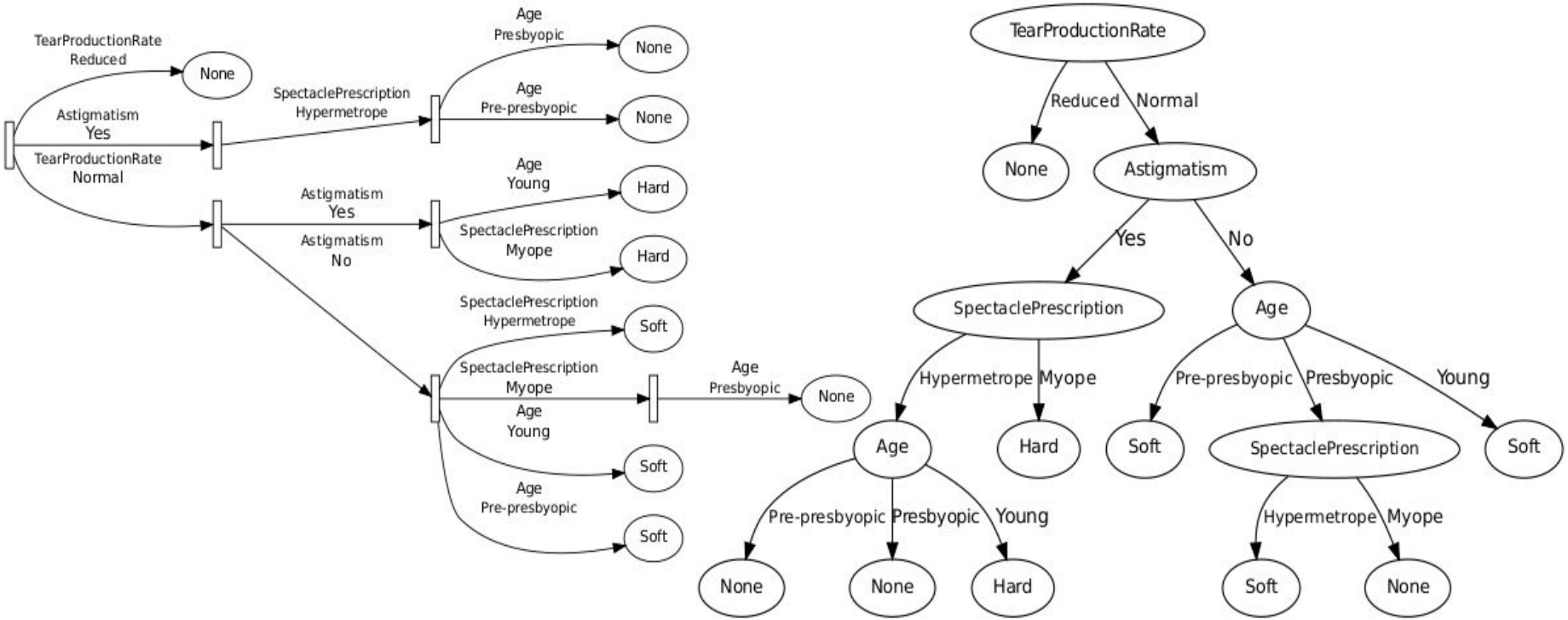
```
def prism_outer(inst, attr_dic, classes):
 rules = []
 for cl in classes:                                  #1
   conj = [x for x in inst]                          #2
   while cl in [x[-1] for x in conj]:                #3
     rule = prism_inner(conj, attr_dic, cl)          #4
     rules.append([rule, cl])                        #5
     conj = removeCoveredIns(rule, conj)             #6
 return rules
```

PRISM Código Python

```
Parámetros:
 inst:     Lista con las instancias.
 attr_dic: Diccionario con los valores de cada atributo.
 cl:       Clase sobre la que generar la regla.
Comentarios:
 1. Generamos una lista con todas las clases en 'inst'.
 2. Si todas las instancias son de la clase 'cl',
    la regla está completa. No es necesario generar más
    condiciones.
 3. Seleccionamos el par (atributo, valor) con mayor ganancia.
 4. Obtenemos las instancias que cumplen 'attr_val'.
 5. Invocamos de forma recursiva este método. De esta manera
    este método se repetirá hasta que el #2 sea cierto.
 6. Finalmente devolvemos la regla generada.
```

```python
def prism_inner(inst, attr_dic, cl):
 cls = [x[-1] for x in inst]                          #1
 if len(set(cls)) == 1:                               #2
   return []
 pairList = []
 attr_val = select_pairAV(attr_dic, inst, cl)         #3
 pairList.append(attr_val)
 subSet = getSubSet(attr_val[0], attr_val[2], inst)   #4
 attr_dic = [x for x in attr_dic if x[0]!=attr_val[0]]
 pairList += prism_inner(subSet, attr_dic, cl)        #5
 return pairList                                      #6
```

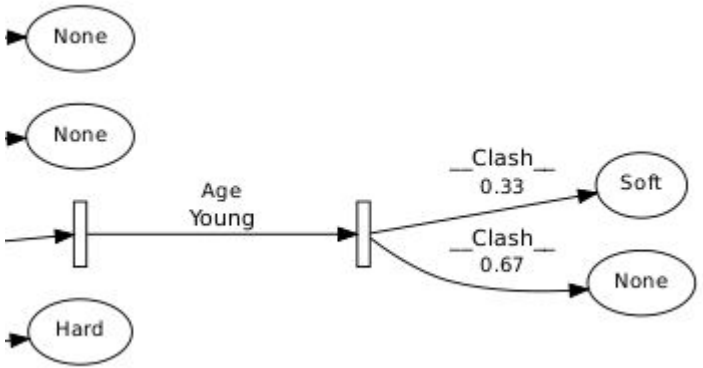
PRISM vs ID3. Eliminación de atributos no relevantes.



Datos incoherentes (clashes)

Una de las mejoras que proponen los autores de '*Principles of Data Mining*' [3] es la extensión del algoritmo, a fin de soportar conjuntos de datos incoherentes.

```
Comentarios:
 1. Si la lista de atributos está vacía, pero tenemos
    instancias de clases diferentes, tenemos un clash.
def prism_inner(inst, attr_dic, cl):
 cls = [x[-1] for x in inst]
 if len(set(cls)) == 1:
   return []
 if len(attr_dic) == 0:                          #1
   aux = cls.count(cl) / float(len(cls))
   return [(-1, '__Clash__', str(round(aux,2)))]
 pairList = []
 # .. .. .. .. .. .. .. .. ..
```

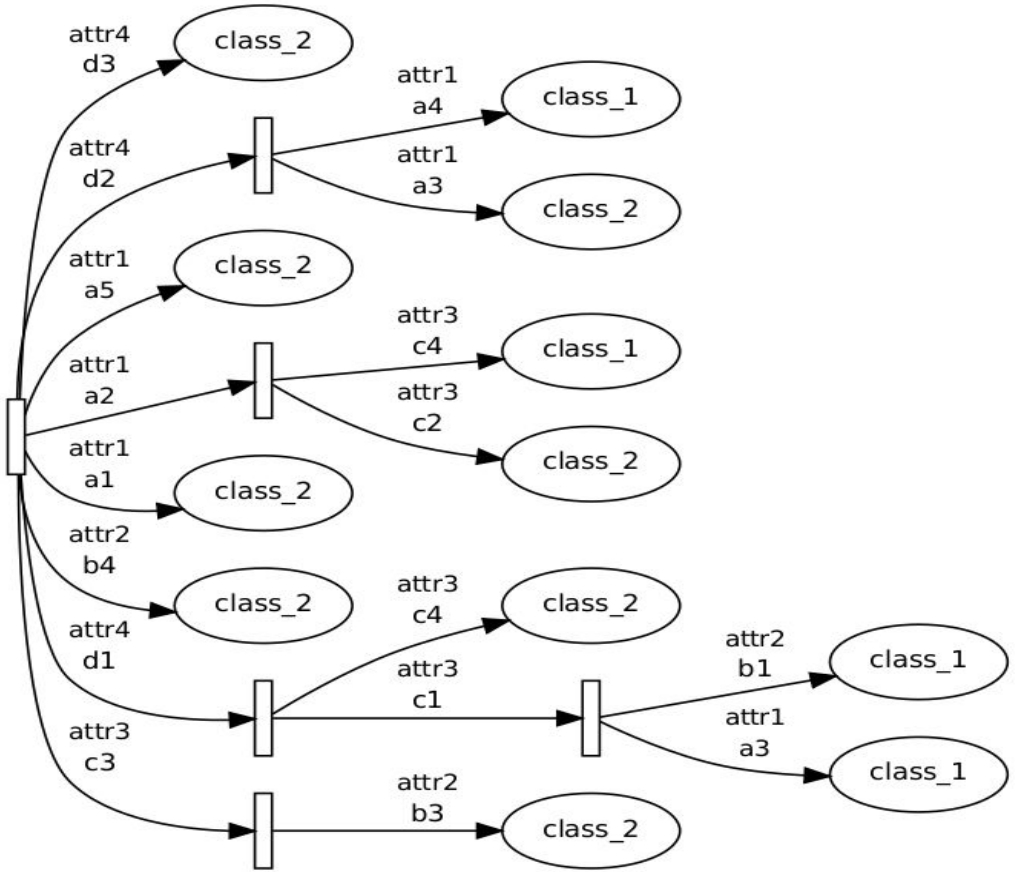


Prism y un conjunto de entrenamiento incompleto.

Consideremos un conjunto de entrenamiento definido por las siguientes reglas. Vamos a utilizar 40/240 instancias.

```
attr1 = a4 and attr4 = d2 --> class_1
attr3 = c1 and attr4 = d1 --> class_1
attr1 = a2 and attr3 = c4 and attr4 = d2 --> class_1
attr1 = a5 and attr3 = c4 and attr4 = d2 --> class_1
otherwise → class_2
```

```
attr1 = a4 and attr4 = d2 --> class_1
attr1 = a2 and attr3 = c4 --> class_1
attr2 = b1 and attr3 = c1 and attr4 = d1 --> class_1
attr1 = a3 and attr3 = c1 and attr4 = d1 --> class_1
otherwise → class_2
```



Prism y un conjunto de entrenamiento incompleto.

Ante un conjunto completo de entrenamiento, PRISM produce un conjunto de reglas completo y correcto. Sin embargo, al utilizar un conjunto de entrenamiento incompleto la fiabilidad del algoritmo reduce. Pueden presentarse los siguientes problemas.

Fallo al producir una regla. Como podemos ver el algoritmo no genera la regla número **4#**. Esto es normal y es debido a la no presencia del ejemplo correspondiente en el conjunto de entrenamiento. En esto PRISM se diferencia a ID3, ya que ID3 asigna la moda ante una falta de ejemplo.

Generalización excesiva. Vemos que la regla B# es una generalización de la regla **3#**. La razón es la falta de un contraejemplo en el conjunto de entrenamiento.

Especialización excesiva. En un conjunto de entrenamiento la $\mathbf{p(C_i|\alpha_x)}$ es una aproximación del valor real. Esta aproximación introduce un *error*, que puede resultar en la selección de atributos no relevantes. En nuestro ejemplo las reglas C# y D# son especializaciones excesivas de la regla **2#**.

```
1#   attr1 = a4 and attr4 = d2 --> class_1
2#   attr3 = c1 and attr4 = d1 --> class_1
3#   attr1 = a2 and attr3 = c4 and attr4 = d2 --> class_1
4#   attr1 = a5 and attr3 = c4 and attr4 = d2 --> class_1
5#   otherwise → class_2
```

```
A#   attr1 = a4 and attr4 = d2 --> class_1
B#   attr1 = a2 and attr3 = c4 --> class_1
C#   attr2 = b1 and attr3 = c1 and attr4 = d1 --> class_1
D#   attr1 = a3 and attr3 = c1 and attr4 = d1 --> class_1
E#   otherwise → class_2
```

Conclusiones

PRISM produce un conjunto de reglas. Estas reglas a diferencia de los árboles de clasificación producidos por algoritmos como el ID3 no incluyen atributos irrelevantes. Esta mejora es posible dado que PRISM busca las parejas **atributo-valor** que mayor ganancia aportan para una clase específica.

PRISM no es un algoritmo muy complicado de implementar. También podemos utilizar la implementación disponible en herramientas como **Weka**.

Ante un conjunto completo de entrenamiento, PRISM produce un conjunto de reglas completo y correcto. Este no es el caso ante un conjunto de entrenamiento incompleto. Cuanto menor es el conjunto, menor es la fiabilidad de las reglas. Aún así comparado al ID3 el comportamiento sigue siendo mejor.

Bibliografía

1. **PRISM: An algorithm for inducing modular rules.** Jadzia Cendrowska (1987). Artículo donde se propuso por primera vez PRISM. http://sci2s.ugr.es/keel/pdf/algorithm/articulo/1987-Cendrowska-IJMMS.pdf
2. **Why learn rules?.** Página web de West Virginia University. Explicación de reglas, conversión árbol-reglas, PRISM, C4.5. http://www.csee.wvu.edu/~timm/cs591o/old/Rules.html
3. **Principles of Data Mining.** Libro, descripción de PRISM en profundidad y paso a paso con ejemplos además de comparaciones entre PRISM y TDIDT. E-Book de la biblioteca (pág. 162): http://0-link.springer.com.cisne.sim.ucm.es/book/10.1007%2F978-1-84628-766-4
4. **Automatic Induction of Classification Rules from Examples Using N-Prism.** Max Bramer. Artículo que explica el funcionamiento de PRISM con conjunto de entrenamientos incompletos o ruido. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.556.7902&rep=rep1&type=pdf
5. **Integración de los algoritmos de minería de datos 1R, PRISM e ID3 a PostgreSQL.** Artículo con diferentes técnicas de minería de datos, descripción y pseudo-codigos. http://www.scielo.br/scielo.php?pid=S1807-17752013000200389&script=sci_arttext&tlng=pt
6. **A Covering-based Algorithm for Classification: PRISM.** Presentación del departamento de ciencias de la computación de la Universidad de Regina, Saskatchewan, Canadá. Pasos básicos de PRISM, ejemplos discusión de los mismos y diferencias entre ID3 y PRISM. http://www2.cs.uregina.ca/~deng200x/PRISM_PPT.pdf