Métodos de Aprendizaje en IA. Guía de prácticas.
Bloque 2: Representaciones Simples.

Capítulo 1

Tema 2: Inducción de Conjunciones Lógicas

Este tema se centrará en la inducción de conceptos formados por conjunciones lógicas. Denominaremos a este tipo de conceptos "conceptos CL" y a su representación "representación CL". Estos algoritmos (ver citas al final del tema 2 [EML]) surgen históricamente en la etapa más temprana del aprendizaje automático en su vertiente *simbólica* (en contraposición a la vertiente *conexionista*, cuyos primeros algoritmos veremos en el tema 3).

Se recomienda tener muy presente el documento [SRFI-1], ya que se hará un uso intensivo de las listas Scheme.

Durante el curso, se añadirá una plantilla de ejercicios para *normalizar* (de alguna forma) las entregas de todos vosotros. Para una mejor planificación del tiempo requerido por cada ejercicio, se recomienda que en la entrega incluyáis el tiempo real que os ha llevado realizar cada ejercicio. Así en el futuro podamos mejorar la planificación temporal de la asignatura. Con este objetivo se debe rellenar `(he-tardado <minutos><ejercicio>)` en cada plantilla[1]. **Importante:** el tiempo estimado no se tendrá en cuenta en ningún caso para evaluar el ejercicio.

1.1. Lenguaje de representación de conceptos CL

Requisitos: estudio de la sección 2.1 [EML] completa. Los ejercicios de este tema servirán para aclarar y reforzar los conceptos aprendidos en esta sección del texto base.

La representación de estos *conceptos CL* se realizará mediante *listas* de test, un test por cada atributo. De hecho, la comprobación de pertenencia de un ejemplo concreto a cierto *concepto CL* se determina mediante la aplicación de cada uno de estos test (uno por uno) a cada uno de los atributos de

[1]Para poder cargar vuestro código sin errores, se añade a cada plantilla el código siguiente: `(define (he-tardado minutos ejercicio) ())`.

dicho ejemplo. Note que la posición de cada test en la lista de test determina de manera unívoca el atributo sobre el que se aplica dicho test.

Los test para atributos nominales adoptarán una de las 3 formas siguientes:

- El test representado por `(*)` acepta instancias con cualquier valor en el atributo correspondiente (es el test más general posible).
- El test representado por `(<valor>)` acepta sólo instancias con el valor `<valor>` en el atributo correspondiente. Por ejemplo, el test `(soleado)` sólo acepta instancias con el valor `soleado` en el atributo correspondiente, pero no instancias con el valor `lluvioso` o `nublado`.
- El test representado por `()` no acepta ninguna instancia (es el test más específico posible), independientemente del atributo al que corresponda dicho test.

Los test para atributos numéricos se componen de una *lista de dos límites*: `(<límite inferior><límite superior>)`, los cuales, cuando son representados por dos *números*, definen un rango *abierto* aceptado por dicho test. No obstante, si uno de los límites se representa como una *lista* de un solo número, se considerará que el rango está *cerrado* por ese extremo. Concretando, dicho test aceptará:

1. Los valores *mayores* que `<límite inferior>`, si se cumple `(number? <límite inferior>)`, siempre que se cumpla que dichos valores sean aceptados por `<límite superior>`.
2. Los valores *menores* que `<límite superior>`, si se cumple `(number? <límite superior>)`, siempre que se cumpla que dichos valores sean aceptados por `<límite inferior>`.
3. Los valores *mayores o iguales* que `(car <límite inferior>)`, si se cumple `(list? <límite inferior>)`, siempre que se cumpla que dichos valores sean aceptados por `<límite superior>`.
4. Los valores *menores o iguales* que `(car <límite superior>)`, si se cumple `(list? <límite superior>)`, siempre que se cumpla que dichos valores sean aceptados por `<límite inferior>`.

Nota: el test numérico `(#i-1/0 #i1/0)` acepta todo el rango $(-\infty, +\infty)$ de valores reales. En otras palabras, acepta cualquier valor y por tanto se considerará el test numérico más general posible. Para simplificar y por paralelismo con los atributos nominales, consideraremos el test `(#i-1/0 #i1/0)` equivalente al test `(*)`, pudiendo representarse dentro de un concepto CL de cualquiera de las dos maneras. Complementariamente, consideraremos cualquier test `(a b)` que cumpla `(> a b)` (sea rango abierto o cerrado),

equivalente al test `()`. Finalmente, por completitud, cualquier test `((a) (a))` será equivalente al test `(a)`.

Algunos ejemplos de conceptos "buen día para salir al campo" mediante la representación CL:

```
((soleado) (25 30) (*) (*));soleado entre 25 y 30 grados (no incluidos).
((nublado) (30) (*) (no));nublado sin viento con 30 grados.
((lluvioso) ((30) #i1/0) (*) (no));lluvioso sin viento con 30 ó más grados.
```

Ejercicio 1. *[2h] Construya la función* `(match-CL <concepto CL><ejemplo-sin-clase>)` *que devuelva* `#t` *si se cumple que: (1)* `<concepto CL>` *y* `<ejemplo-sin-clase>` *tienen el mismo tamaño (misma longitud), (2)* `<ejemplo-sin-clase>` *contiene valores numéricos en las posiciones en que* `<concepto CL>` *contiene test numéricos, (3)* `<ejemplo-sin-clase>` *cumple todos los test atributo por atributo de* `<concepto CL>`*. En cualquier otro caso, la función devolverá* `#f`*. Por ejemplo,*

```
> (match-CL '((soleado)(*)(10 40)(si)) '(soleado 30 40 si))
 #f
> (match-CL '((soleado)(*)(10 (40))(si)) '(soleado 30 40 si))
 #t
> (match-CL '((soleado)(*)(10 40)(si)) '(soleado 30 25 no))
 #f
> (match-CL '((soleado)(*)(10 40)(si)) '(30 soleado 25 no))
 #f
```

Ejercicio 2. *[10'] A partir del ejercicio anterior, implemente un intérprete para este tipo de conceptos* `(CLi <concepto CL><ejemplo-sin-clase>)`*. Por ejemplo,*

```
> (CLi '((soleado)(*)(10 40)(si)) '(soleado 30 40 si))
(soleado 30 40 si +)
```

Utilizando el lenguaje de representación CL podemos afianzar algunos conceptos introducidos en la sección 1.4 [EML]. Según lo expresado en la sección 1.4.2 [EML], en todo algoritmo de aprendizaje existe un *sesgo (bias) de representación* implícito en el lenguaje utilizado para representar los conceptos asociados a dicho algoritmo. Este lenguaje restringe los posibles conceptos o hipótesis a obtener. De hecho, se suele utilizar la noción de *búsqueda* en un *espacio de hipótesis* para definir la actividad de un algoritmo de aprendizaje (para ampliar, estudiar la sección 2.3 del texto [ML]).

En el caso de conceptos CL, este espacio se puede ver muy claramente. Por ejemplo, imaginemos "buen día para salir al campo" representado únicamente por dos atributos nominales como `perspectiva` y `viento`. En este caso, el *espacio de hipótesis* (de conceptos CL) estaría formado por 20 elementos:

```
((*)(*))
((*)(si))((*)(no))
((soleado)(*))((nublado)(*))((lluvioso)(*))
((soleado)(si))((soleado)(no))
((nublado)(si))((nublado)(no))
((lluvioso)(si))((lluvioso)(no))
((*)())(()(*))
((soleado)())((nublado)())((lluvioso)())
(()(si))(()(no))
(()())
```

En conclusión, la tarea de los algoritmos de aprendizaje de conceptos se puede definir como la *búsqueda* de un concepto que encaje con los ejemplos de entrenamiento de entre aquellos presentes en el *espacio de hipótesis* estructurado (permitido) por *su* representación asociada.

1.1.1. Orden parcial de conceptos CL

Como se indica en la sección 2.1.3 [EML], los conceptos CL poseen una ordenación parcial implícita. Por ejemplo, el concepto `A=((soleado) (25 30) (*) (*))` incluye varias instancias además de todas las que incluye el concepto `B=((soleado) (25 30) (*) (no))`. Pero el concepto `B` no incluye ninguna instancia más que el concepto `A`. Por tanto, se define que `A` es *más general* que `B` o que `B` es *más específico* que `A`.

No obstante, este orden es *parcial* ya que existen parejas de conceptos en los que esta relación no se cumple. En otras palabras, `A` puede no ser ni más general ni más específico que `B`. Por ejemplo, dados los conceptos `A=((soleado) (25 30) (*) (*))` y `B=((*) (25 30) (*) (si))`, cualquiera incluye instancias que el otro no incluye, por lo que ambos conceptos se definen de *orden equivalente*.

Ejercicio 3. ***[10']*** *A partir del ejercicio 1 de [P1] y utilizando la representación anterior, junto con lo explicado en la sección 2.1 de [EML], construya:*

1. *El concepto CL más general posible. En otras palabras, el concepto "buen día para salir al campo" que incluya cualquier posible día.*

2. *El concepto CL más específico posible. En otras palabras, el concepto "buen día para salir al campo" que no incluya ningún día.*

3. *El concepto CL más cercano (representable en este lenguaje) al que para usted signifique "buen día para salir al campo".*

Ejercicio 4. ***[5']*** *Construya la función* `(concepto-CL-mas-general <metadatos>)` *que devuelva el concepto más general dado un conjunto de datos.*

Ejercicio 5. *[5']* *Construya la función* `(concepto-CL-mas-especifico <metadatos>)` *que devuelva el concepto más específico dado un conjunto de datos.*

Para los algoritmos de este tema serán necesarias funciones que comparen el grado de *generalización/especialización* entre dos conceptos CL.

Ejercicio 6. *[2h]* *Construya la función* `(test-CL>= <t1><t2>)` *que devuelva* `#t` *si el test* `<t1>` *es más general o de la misma categoría que el test* `<t2>`*. En otro caso, debe devolver* `#f`*. Por ejemplo,*

```
(test-CL>= '(*) '(*))
#t
(test-CL>= '(lluvioso) '(soleado))
#t
(test-CL>= '(lluvioso) '())
#t
(test-CL>= '() '(lluvioso))
#f
(test-CL>= '(25 30) '(26))
#t
(test-CL>= '(25 30) '(21 25))
#t
(test-CL>= '(26) '(25 30))
#f
```

Ejercicio 7. *[30']* *A partir del ejercicio anterior, construya la función* `(concepto-CL>= <c1><c2>)` *que devuelva* `#t` *si el concepto* `<c1>` *es más general o de la misma categoría que el concepto* `<c2>`*. En otras palabras, si todos y cada uno de los test de* `<c1>` *son más generales o de la misma categoría que los test de* `<c2>`*. En otro caso, debe devolver* `#f`*. Por ejemplo,*

```
(concepto-CL>= '((lluvioso)(2 10)) '((soleado)(10 20)))
#t
(concepto-CL>= '((lluvioso)(2 35)) '((soleado)(23)))
#t
(concepto-CL>= '((soleado)(23)) '((lluvioso)(2 35)))
#f
```

Ejercicio 8. *[20']* *A partir del ejercicio anterior, construya la función* `(cmp-concepto-CL <c1><c2>)` *que devuelva:*

- *1, si el concepto* `<c1>` *es* estrictamente más general *que el concepto* `<c2>` `(and (>= <c1><c2>) (not (<= <c1><c2>)))` .
- *0, si el concepto* `<c1>` *es* estrictamente *de la misma categoría que el concepto* `<c2>` `(and (>= <c1><c2>) (<= <c1><c2>))` .

- *-1, si el concepto* `<c1>` *es* estrictamente más específico *que el concepto* `<c2> (and (not (>= <c1><c2>)) (<= <c1><c2>))`

Por ejemplo,

```
(cmp-concepto-CL '((lluvioso)(2 35)) '((soleado)(23)))
1
(cmp-concepto-CL '((lluvioso)(2 10)) '((soleado)(10 20)))
0
(cmp-concepto-CL '((soleado)(23)) '((lluvioso)(2 35)))
-1
```

Como se indica en la sección 2.1.3 de [EML], la mayor parte de los algoritmos de aprendizaje de conceptos CL hacen uso de este orden parcial como ruta en su *búsqueda* dentro del *espacio de hipótesis*. En particular, en cada paso será requerido el conjuntos de conceptos *inmediatamente* más generales o *inmediatamente* más específicos mediante la *inmediata generalización/especialización* de *uno*, y sólo uno, de los test de un determinado concepto.

La *generalización* tiene como objeto el poder *incluir* uno o más ejemplos de entrenamiento positivos, mientras se sigan *rechazando* los mismos ejemplos negativos. La *especialización* tiene como objeto *rechazar* uno o más ejemplos negativos, mientras se sigan *incluyendo* los mismos ejemplos positivos.

El caso de la *inmediata generalización* o la *inmediata especialización* de test en atributos *nominales* es bastante directa, ya que estos siguen un orden muy concreto. Por ejemplo, el siguiente paso en una *inmediata generalización* de un test nominal sería `() =>(<valor>) =>(*)`. Note que la *inmediata* generalización o especialización de cada test nominal genera *como máximo* tantos nuevos conceptos CL generalizados o especializados como valores posea dicho atributo.

Ejercicio 9. *[45']* *Construya la función* `(especializaciones-atributo-nominal <concepto CL><indice-atributo><metadatos>)` *que devuelva una lista de todos los conceptos CL que sean especialización* inmediata *de* `<concepto CL>` *en el atributo referenciado mediante* `<indice-atributo>`. *Por ejemplo,*

```
> (especializaciones-atributo-nominal
                '((*)(*)(20)(si)) 0 (car ejemplos))
(((soleado)(*)(20)(si))((lluvioso)(*)(20)(si))((nublado)(*)(20)(si)))
> (especializaciones-atributo-nominal
                '((soleado)(*)(20)(si)) 0 (car ejemplos))
((()(*)(20)(si)))
> (especializaciones-atributo-nominal
                '(()(*)(20)(si)) 0 (car ejemplos))
((()(*)(20)(si)))
```

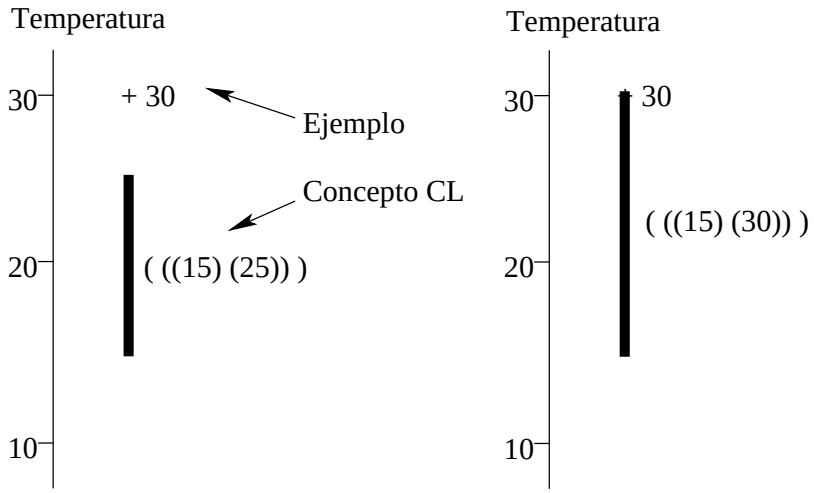

Figura 1.1: Ilustración del proceso de generalización de un atributo numérico. A la izquierda se muestra el concepto CL `( ((15)(25)) )` antes de *cubrir* el nuevo ejemplo *positivo* `30`. A la derecha aparece el concepto CL `( ((15)(30)) )` tras su *generalización inmediata*.

Ejercicio 10. *[20']* *Construya la función* `(generalizaciones-atributo-nominal <concepto CL><indice-atributo><metadatos>)` *que devuelva una lista de todos los conceptos CL que sean generalización* inmediata *de* `<concepto CL>` *en el atributo referenciado mediante* `<indice-atributo>`. *Por ejemplo,*

```
> (generalizaciones-atributo-nominal
               '((soleado)(*)(20)()) 3 (car ejemplos))
(((soleado)(*)(20)(si))((soleado)(*)(20)(no)))
> (generalizaciones-atributo-nominal
               '((soleado)(*)(20)(si)) 3 (car ejemplos))
(((soleado)(*)(20)(*)))
> (generalizaciones-atributo-nominal
               '((soleado)(*)(20)(*)) 3 (car ejemplos))
(((soleado)(*)(20)(*)))
```

Para el caso de atributos numéricos, calcular la *inmediata* generalización o especialización de un test numérico es algo menos evidente, por lo que le dedicaremos más tiempo a este aspecto.

En el caso *generalizar* debemos notar que el objetivo consiste en que el concepto CL pueda *cubrir* el nuevo ejemplo, si este es *positivo*. En los casos en que el ejemplo sea negativo o el ejemplo positivo ya esté cubierto por el concepto CL, dicho ejemplo se ignorará. Como ejemplo, vea la figura 1.1.

En el caso *especializar* debemos notar que el objetivo consiste en que el concepto CL pueda *rechazar* el nuevo ejemplo, si este es *negativo*. En los casos en que el ejemplo sea positivo o el ejemplo negativo ya sea rechazado

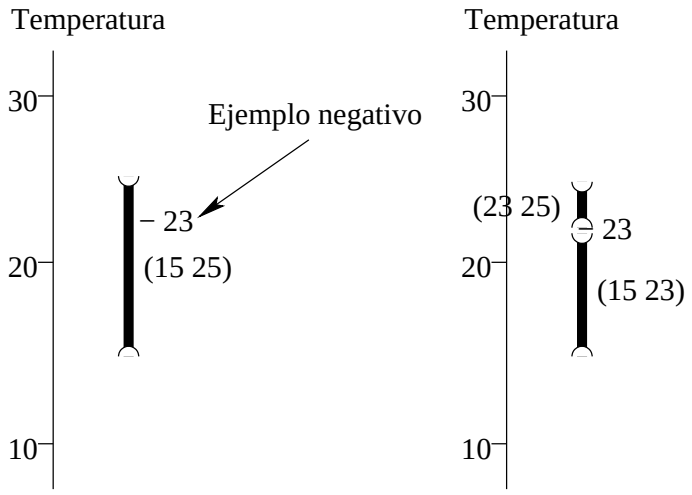

Figura 1.2: Ilustración del proceso de especialización de un atributo numérico. A la izquierda se muestra el concepto CL `( (15 25) )` antes de *rechazar* el nuevo ejemplo *negativo* `23`. A la derecha aparecen dos nuevos conceptos CL `( (15 23) )` y `( (23 25) )` tras su *especialización inmediata*.

por el concepto CL, dicho ejemplo se ignorará. Como ejemplo, vea la figura 1.2.

Ejercicio 11. ***[1h]*** *Construya la función* `(generalizacion-atributo-numerico <concepto CL><indice-atributo><ejemplo>)` *que devuelva el concepto CL que es generalización inmediata de* `<concepto CL>` *en el atributo referenciado mediante* `<indice-atributo>`*. Por ejemplo,*

```
> (generalizacion-atributo-numerico
    '((soleado)()(20)(si)) 1 '(soleado 25 40 si +))
((soleado)(25)(20)(si))

> (generalizacion-atributo-numerico
    '((soleado)()(20)(si)) 1 '(soleado 25 40 si -))
((soleado)()(20)(si))

> (generalizacion-atributo-numerico
    '((soleado)(15 30)(20)(si)) 1 '(soleado 25 40 si +))
((soleado)(15 30)(20)(si))
```

Ejercicio 12. ***[2h]*** *Construya la función* `(especializaciones-atributo-numerico <concepto CL><indice-atributo><ejemplo>)` *que devuelva una lista de todos los conceptos CL que sean especialización* inmediata *de* `<concepto CL>` *en el atributo referenciado mediante* `<indice-atributo>`*. Por ejemplo,*

```
> (especializaciones-atributo-numerico
```

```
          '((*)(*)(20)(si)) 1 '(soleado 25 40 si -))
(((*)(#-1/0 25)(20)(si))((*)(25 #+1/0)(20)(si)))

> (especializaciones-atributo-numerico
          '((*)(*)(20)(si)) 1 '(soleado 25 40 si +))
(((*)(*)(20)(si)))

> (especializaciones-atributo-numerico
          '((*)(10 15)(20)(si)) 1 '(soleado 25 40 si -))
(((*)(10 15)(20)(si)))
```

Ejercicio 13. *[30']* *Construya la función* *(generalizaciones-CL <concepto CL><metadatos><ejemplo>)* *que devuelva el conjunto total de generalizaciones inmediatas (respecto a todos los atributos) de* *<concepto CL>.*

Ejercicio 14. *[30']* *Construya la función* *(especializaciones-CL <concepto CL><metadatos><ejemplo>)* *que devuelva el conjunto total de especializaciones inmediatas (respecto a todos los atributos) de* *<concepto CL>.*

1.2. Inducción Exhaustiva de conceptos CL

Se requiere haber leído detenidamente la sección 2.2 [EML], sobretodo haber entendido el pseudocódigo de la tabla 2-1, el cual habrá que implementar. Este tipo de inducción se denomina exhaustiva porque busca *todos* los conceptos CL (hipótesis) que cubren el total de instancias positivas rechazando a la vez todas las instancias negativas. Como veremos, el algoritmo `EGS` busca una de entre las *hipótesis* más generales que cumplen esto último.

Ejercicio 15. *[2h]* *Implemente la función* *(EGS0 <PSET><NSET><CSET><HSET>)* *a partir del pseudocódigo de la tabla 2-1 [EML] y del ejercicio 14. Después, implemente la función* *(EGS <ejemplos>)* *que llama a* *(EGS0 ejemplos+ ejemplos- () (list (concepto-CL-mas-general (car ejemplos))))* *y que devuelve un solo concepto CL (por ejemplo, al azar) de la lista* *<CSET>* *de* hipotéticos *conceptos o* conjunto de hipótesis *razonables. Para probar el algoritmo utilice el conjunto de ejemplos habitual. Tras ello, compare el concepto CL devuelto por* *EGS* *con el obtenido del ejercicio 3. ¿Tiene relación el concepto devuelto por* *EGS* *con su propio concepto de "buen día para salir al campo"?.*

1.2.1. Desventajas de EGS

Se requiere estudiar la sección 2.2.3 [EML]

Ejercicio 16. ***[20']*** *Ejecute* `EGS` *mediante el conjunto de ejemplos "agaricus-lepiota.scm" e "ionosphere.scm" (ver "repositorio de material" ejemplos UCI). Observe la traza del algoritmo y describa el comportamiento de éste en presencia de los nuevos conjuntos de ejemplos.*

1.3. Inducción Heurística de conceptos CL

Se requiere estudiar la sección 2.3 [EML]

Ejercicio 17. ***[30']*** *Implemente la función* `(score-CL <conceptoCL><PSET><NSET>)` *mediante* `match-CL` *(ejercicio 1). Utilice la medida que aparece en la sección 2.3.2 [EML].*

Ejercicio 18. ***[2h]*** *Implemente la función* `(HGS0 <PSET><NSET><CSET><HSET>)` *a partir del pseudocódigo de la tabla 2-2 [EML] y del ejercicio anterior. Después, implemente la función* `(HGS <ejemplos>)` *que llama a* `(HGS0 ejemplos+ ejemplos- () (list (concepto-CL-mas-general (car ejemplos))))` *y que devuelve un solo concepto CL (por ejemplo, al azar) de la lista* `<CSET>` *de* hipotéticos *conceptos o* conjunto de hipótesis*. Para probar el algoritmo utilice el conjunto de ejemplos habitual. Tras ello, compare el concepto CL devuelto por* `HGS` *con el obtenido por* `EGS` *y el del ejercicio 3. ¿Tiene relación el concepto devuelto por* `HGS` *con su propio concepto de "buen día para salir al campo"?.*

Ejercicio 19. ***[20']*** *Ejecute* `HGS` *mediante el conjunto de ejemplos "agaricus-lepiota.scm" e "ionosphere.scm" (ver "repositorio de material" ejemplos UCI). Observe la traza del algoritmo y describa el comportamiento de este algoritmo en presencia de estos ejemplos. Compare el comportamiento de* `HGS` *con el de* `EGS`*.*

Capítulo 2

Tema 3: Inducción de Conceptos Umbral

Este tema comenzará introduciendo estructuras y conceptos (conceptos TC) relacionados con el tema anterior. No obstante, muchas de las ideas que caracterizan a los conceptos umbral provienen de la vertiente *conexionista* de la inteligencia artificial (ver citas al final del tema 3 [EML]). Por ejemplo, en este tema introduciremos el *concepto LUU*, el cual en otros contextos (y para cierto algoritmo concreto) se denomina *perceptrón* o neurona simple. Note que este tema no trata las *redes de neuronas* debido a que, siguiendo la ruta marcada por el texto base [EML], esta estructura es un tipo de *concepto compuesto* que veremos en el bloque 3 (representaciones compuestas).

2.1. Lenguaje de representación de Conceptos Umbral

Se requiere el estudio de la sección 3.1 [EML] completa.

2.1.1. Representación de conceptos tabla de criterios (conceptos TC)

Como se indica en la sección 3.1.1 [EML] un concepto *tabla de criterios*, de ahora en adelante denominado *concepto TC*, es aquel que, a diferencia de los *conceptos CL* estudiados en el tema anterior, no requiere una *correspondencia* (o *match*) de todos los test del concepto, sino únicamente `m-de-n`.

Debido a su similitud con los *conceptos CL*, reutilizaremos algunas estructuras del tema anterior para trabajar con estos nuevos *conceptos TC*. Para ello, la representación de un *concepto TC* o *representación TC* será una ampliación de la *representación CL*. Concretando, un *concepto TC* se representa como un *concepto CL*, pero en el cual su primer elemento es siempre un número *entero* y no un test. Dicho número entero será el *umbral* `m` de

los `n` posibles atributos `(= n (length <concepto CL>))` que una instancia debe cumplir para ser aceptada (match) por el correspondiente *concepto TC*. Ejemplos de *conceptos TC*:

```
(3 (soleado) (25 30) (*) (*))
(2 (nublado) (30) (*) (no))
(4 (lluvioso) ((30) #i1/0) (*) (no))
```

Observe que el último concepto TC es equivalente a su correspondiente concepto CL (sin el número delante).

Ejercicio 20. ***[1h]*** *Construya la función* `(match-TC <concepto TC><ejemplo-sin-clase>)` *que devuelva* `#t` *si se cumple que: (1)* `(cdr <concepto TC>)` *y* `<ejemplo-sin-clase>` *tienen el mismo tamaño, (2)* `<ejemplo-sin-clase>` *contiene atributos numéricos en las posiciones en que* `<concepto TC>` *contiene test numéricos, (3)* `<ejemplo-sin-clase>` *cumple* al menos `m` *de los* `n` *test de* `<concepto TC>`*. En cualquier otro caso, la función devolverá* `#f`*. Por ejemplo,*

```
#kawa:X# (match-TC '(4 (soleado)(*)(10 40)(si)) '(soleado 30 40 si))
 #f
#kawa:X# (match-TC '(3 (soleado)(*)(10 40)(si)) '(soleado 30 40 si))
 #t
#kawa:X# (match-TC '(2 (soleado)(*)(10 (40))(si)) '(soleado 30 40 si))
 #t
#kawa:X# (match-TC '(4 (soleado)(*)(10 40)(si)) '(soleado 30 25 no))
 #f
#kawa:X# (match-TC '(3 (soleado)(*)(10 40)(si)) '(30 soleado 25 no))
 #f
```

Ejercicio 21. ***[15']*** *A partir del ejercicio anterior, implemente un intérprete para este tipo de conceptos* `(TCi <concepto CL><ejemplo-sin-clase>)`*. Por ejemplo,*

```
#kawa:X# (TCi '(4 (soleado)(*)(10 40)(si)) '(soleado 30 40 si))
(soleado 30 40 si +)
```

Orden parcial de los conceptos TC

Aunque los conceptos TC siguen un orden parcial como en el caso de los conceptos CL, no implementaremos aquí las correspondientes funciones de comparación. No obstante, sí serán necesarias funciones de especialización inmediata. Tampoco implementaremos funciones de generalización.

Para ser más efectivos, utilizaremos todas las funciones previas para especializar atributos procedentes de los ejercicios 9 y 12. Sin embargo, deberá tener en cuenta que al especializar también tendrá que incluir los conceptos que proceden de aumentar en 1 el umbral `m` hasta llegar al valor `n`.

También habrá que incluir aquellos conceptos en que se especializa más de un atributo a la vez (recuerde que hablamos de *especializaciones inmediatas*, sección 1.1.1). Para ello, habrá que disminuir el umbral `m` (como mucho hasta llegar a 0) una vez por cada atributo especializado extra (p.ej. ver el ejemplo del ejercicio siguiente `(0 (soleado)())`).

Ejercicio 22. ***[1h][30']*** *Construya la función* *`(especializaciones-TC <concepto TC><metadatos><ejemplo>)`* *que devuelva el conjunto total de especializaciones* inmediatas *(respecto a todos los atributos) de* *`<concepto TC>`*. *Por ejemplo (en caso de un concepto TC con 2 atributos nominales):*

```
> (especializaciones-TC
    '(1 (*)(si))
        (car ejemplos)(car (cdr ejemplos)))

((1 (*)())(1 (soleado)(si))(2 (soleado)(si))(0 (soleado)()))

> (especializaciones-TC
    '(2 (soleado) (masdecero) () (*))
        (car ejemplos) (car (cdr ejemplos)))

(
(2 ()(*))
(2 (soleado)(*))
(2 (soleado)(si))
(2 (soleado)(si))
(2 (soleado)(no))
(3 (soleado)(*))
(4 (soleado)(*))
(1 ()()()(*))
(1 ()(masdecero)()(*))
(1 ()(masdecero)()(si))
(1 ()(masdecero)()(no))
(1 (soleado)()()(*))
(1 (soleado)()()(si))
(1 (soleado)()()(no))
(1 (soleado)(masdecero)()(si))
(1 (soleado)(masdecero)()(no))
(0 ()()()(*))
(0 ()()()(si))
(0 ()()()(no))
(0 ()(masdecero)()(si))
(0 ()(masdecero)()(no))
(0 (soleado)()()(si))
```

```
(0 (soleado)()()(no))
)
```

Nota: utilice las funciones de los ejercicios 9 y 12 pasando como parámetros `<indice-atributo>+1` *y un elemento cualquiera al comienzo de* `<metadatos>` *para que dichas funciones "sigan suponiendo" que se está introduciendo un* `<concepto CL>` *y no un* `<concepto TC>`.

2.1.2. Representación de conceptos Unidad Umbral (conceptos UU)

Se requiere el estudio de las secciones 3.1.2 y 3.1.3 [EML]. Como se indica en dichas secciones, a diferencia de los conceptos TC, estos conceptos permiten asignar un *peso* distinto a cada atributo. Para ello, los conceptos de esta sección se representan mediante un *vector* de números reales. Los denominaremos *concepto UU*, aunque especificaremos mediante *concepto LUU* cuando hablemos de unidades umbral de tipo lineal, dejando abierta la posibilidad de que un concepto UU pueda representar también conceptos de tipo esférico *SUU*.

En los conceptos UU la asignación a una clase u otra se computa mediante operaciones numéricas entre el vector del concepto UU y los atributos del ejemplo. Para ello, se requiere una *traducción* previa de los atributos *nominales* a números. Para unificar criterios, realizaremos esta traducción asociando el valor de cada atributo *nominal* con su *índice* (`list-ref`) en la lista asociada a *su* atributo en los metadatos.

Ejercicio 23. *[30']* *Construya la función* `(traducir <meta-atributo><valor>)` *que devuelve la traducción del* `<valor>` *nominal. En caso de que* `<meta-atributo>` *resulte un atributo numérico, la función devolverá el valor numérico tal cual sin traducir. Por ejemplo,*

```
> (traducir '(perspectiva (soleado nublado lluvioso)) lluvioso)
 2
> (traducir '(temperatura numerico) 15)
 15
```

Debido a la necesidad de mantener los metadatos para realizar la *traducción* de los atributos nominales, los conceptos UU deberán tener un subapartado dedicado a dichos metadatos. Por tanto, definiremos los conceptos UU como *listas* de 2 elementos: (1) los metadatos y (2) un vector representado por una *lista* de números.

Ejercicio 24. *[30']* *Construya la función* `(nuevo-conceptoUU <metadatos><init>)` *que devuelva un conceptoUU en el cual el vector (la lista de números) esté inicializado a distintos valores al azar entre* `-<init>` *y* `+<init>`[1]. *El vector debe*

[1] Utilice por ejemplo: `(* <init>(- (* 2 (random)) 1))`

tener el mismo tamaño que `(length <metadatos>)`*, ya que consideraremos el último valor del vector como el* `-umbral` *a superar. Por ejemplo,*

```
> (nuevo-conceptoUU (car ejemplos) 1)
(((perspectiva (soleado ...) ...)...) (-0.3 0.6 0.2 -0.1 0.3))
                                       ---------v------- -v-
                                              vector    umbral=-0.3
```

Desde este momento nos centraremos únicamente en los conceptos LUU. En estos, la clase se obtiene a partir del "producto escalar" entre el vector del concepto LUU y el ejemplo de entrenamiento "traducido" (ejercicio 23).

Ejercicio 25. ***[1h]*** *Construir la función* `(match-LUU <conceptoUU><ejemplo-sin-clase>)` *que devuelva* `#t` *en caso de que* `<ejemplo-sin-clase>` *supere o iguale el umbral presente en el* vector *de* `<concepto UU>`*. Para ello, debe basarse en los ejemplos de la sección 3.1.2. La función debe devolver* `#f` *en cualquier otro caso. Por ejemplo,*

```
(match-LUU (list(car ejemplos)'(0 1 1 0 -30)) '(lluvioso 10 20 si))
#t
(match-LUU (list(car ejemplos)'(0 1 1 0 -31)) '(lluvioso 10 20 si))
#f
```

Sugerencia: calcule el producto escalar del vector de concepto UU `(cdar <conceptoUU>)` *con* `(append <ejemplo-sin-clase-traducido> 1)`*. Devolver* `#t` *ó* `#f` *dependiendo de que el producto escalar resulte positivo o negativo.*

Ejercicio 26. ***[15']*** *A partir del ejercicio anterior, implemente un intérprete para este tipo de conceptos* `(LUUi <conceptoUU><ejemplo-sin-clase>)`*. Por ejemplo,*

```
#kawa:X# (LUUi (list(car ejemplos)'(0 1 1 0 -30)) '(soleado 30 40 si))
(soleado 30 40 si +)
```

2.2. Inducción de conceptos TC

Esta sección se basa en la sección 3.2 [EML], aunque no se sigue el texto al pie de la letra. Note que los atributos que utilizamos no son booleanos, por lo que lo explicado en dicha sección nos sirve únicamente de inspiración para lo que viene a continuación.

Ejercicio 27. ***[30']*** *Basándose en el código del ejercicio 17, implemente la función* `(score-TC <concepto-TC><PSET><NSET>)` *mediante* `match-TC` *(ejercicio 20). Utilice la medida que aparece en la sección 2.3.2 [EML].*

Ejercicio 28. *[2h] Implemente la función* `(HTCO <PSET><NSET><CSET><HSET>)` *a partir de los pseudocódigos de la tabla 2-2 y la tabla 3-1 de [EML]. En efecto, considere una "intersección" entre los dos pseudocódigos reutilizando en lo posible el código del ejercicio 18. Después, implemente la función* `(HTC <ejemplos>)` *que llama a* `(HTCO ejemplos+ ejemplos- () (list (length (cadr ejemplos)) (concepto-CL-mas-general (car ejemplos))))` *y que devuelve un solo concepto TC (por ejemplo, al azar) de la lista* `<CSET>` *de* hipotéticos *conceptos o* conjunto de hipótesis*. Para probar el algoritmo, utilice el conjunto de ejemplos habitual. Tras ello, compare el concepto TC devuelto por* `HTC` *con el concepto CL obtenido por* `HGS` *y el concepto CL del ejercicio 3.*

Ejercicio 29. *[20'] Ejecute* `HTC` *mediante el conjunto de ejemplos "agaricus-lepiota.scm" e "ionosphere.scm" (ver "repositorio de material" ejemplos UCI). Observe la traza del algoritmo y describa su comportamiento en presencia de estos nuevos ejemplos. También compare el comportamiento de* `HTC` *con el de* `HGS`*.*

2.3. Inducción de conceptos LUU

Requisitos: estudio de la sección 3.3 [EML], en particular secciones 3.3.1 y 3.3.2 [EML]. En esta sección construiremos los algoritmos correspondientes a la actualización de un perceptrón y una unidad *least mean square*.

Ejercicio 30. *[2h] Implemente la función* `(PRM <concepto UU><ejemplos>)` *que devuelve un* `<concepto LUU>` *revisado para ajustarse a* `<ejemplos>` *un paso más. Compruebe su funcionamiento con sus ejemplos de entrenamiento "buen día para salir al campo".*

Ejercicio 31. *[1h] Implemente la función* `(PCP <ejemplos>)` *que devuelve un* `<concepto LUU>`*. Utilice* `COUNT` *a 1000 épocas. Compruebe la función con sus ejemplos de entrenamiento "buen día para salir al campo".*

Ejercicio 32. *[1h][30'] Implemente la función* `(LMS <ejemplos>)` *que devuelve un* `<concepto LUU>`*. Utilice* `COUNT` *a 1000 épocas. Compruebe la función con sus ejemplos de entrenamiento "buen día para salir al campo".*

Ejercicio 33. *[30'] Compruebe la diferencia de precisión entre* `PCP` *y* `LMS` *(utilice la función* `cross-validation` *con* `folds` *a 10) en los conjuntos de ejemplos "buen día para salir al campo", "agaricus-lepiota.scm" e "ionosphere.scm". ¿Encuentra diferencias significativas?*

Capítulo 3

Tema 4: Inducción de conceptos competitivos

3.1. Representación y obtención de conceptos basados en instancias (conceptos IB)

Requisitos: estudio de la sección 4.1 [EML].

La representación de este tipo de conceptos es directa. Simplemente consiste en almacenar los ejemplos de entrenamiento tal cual. Por tanto, representaremos los *conceptos IB* mediante el propio conjunto de ejemplos de entrenamiento.

El algoritmo de aprendizaje es tan sencillo que lo incluimos directamente:

```
(define (IB ejemplos) ejemplos)
```

Como podemos ver, realmente todo el trabajo de inducción se realiza en el intérprete:

Ejercicio 34. *[30']* *Construya la función* `(distancia <ejemplo-x><ejemplo-y>)` *que calcule la distancia Euclídea entre dos ejemplos con o sin clase. Para ello, note que debe hacer colapsar los 2 ejemplos al de menor longitud eliminando la* clase *del de mayor longitud. Después se calcula la distancia (en una dimensión) de cada atributo:* `(- valor-x valor-y)` *en el caso de los valores numéricos; en el caso de valores nominales, considere que dos valores nominales iguales se encuentran a distancia 0 y dos valores nominales distintos a distancia 1. Finalmente, con las distancia de cada dimensión (de cada atributo) se calcula la distancia Euclídea entre los dos ejemplos. Por ejemplo,*

```
> (distancia '(soleado 10 20 si) '(soleado 15 21 no -))
5.196152422706632;= (sqrt (+ 0 (exp 5 2) (exp 1 2) 1))
```

Ejercicio 35. *[40']* *Construir* `(IBi <concepto IB><ejemplo-sin-clase>)` *que devuelve el ejemplo con la clase del vecino más cercano de entre los pertenecientes a* `<concepto IB>`.

Ejercicio 36. *[15']* *A partir del ejercicio anterior construir* `(match-IB <concepto IB><ejemplo-sin-clase>)`.

3.2. Representación y obtención de conceptos probabilísticos (conceptos NB)

Requisitos: estudio de las sección 4.2 [EML].

Para representar conceptos probabilísticos Naive Bayes o *conceptos NB* utilizaremos una lista de listas. Cada una de estas listas contiene los valores necesarios para calcular la probabilidad condicionada de la clase correspondiente. Por simplicidad, asumiremos que los valores numéricos siguen una distribución normal. Para unificar criterios, se explicita a continuación el código necesario para su creación.

```
(define (nuevo-conceptoNB metadatos)
  (do ((cuentas '(0))
       (valores ())
       (i 0 (+ i 1))
      )
     ((= i (- (length metadatos) 1))
      (let ((c (reverse cuentas)))
       (list (cons '+ c)(cons '- c)));valor devuelto
     )
    (set! valores (cadr (list-ref metadatos i)))
    (cond
      ((eq? valores 'numerico)
        (set! cuentas (cons '(numerico 0 0) cuentas)))

       (else ;nominales
        (set! cuentas
             (cons (map (lambda(x) (cons x 0))
                    valores)
                   cuentas)))
)))
```

Ilustraremos la representación de estos conceptos mediante un ejemplo:

```
> (nuevo-conceptoNB (car ejemplos))
((+ 0
  ((soleado . 0) (nublado . 0) (lluvioso . 0))
```

```
   (numerico 0 0)
   (numerico 0 0)
  ((si . 0) (no . 0)))
 (- 0
  ((soleado . 0) (nublado . 0) (lluvioso . 0))
   (numerico 0 0)
   (numerico 0 0)
  ((si . 0) (no . 0))))
```

El concepto en sí contiene una lista por cada clase `+` ó `-`, cada una de tamaño `atributos+2`. El primer y segundo valor de cada una de estas listas contienen el nombre de la clase y el número de veces que dicha clase ha aparecido en el conjunto de datos utilizado para obtener el concepto NB. Cada *atributo nominal* debe mantener la cuenta de la cantidad de veces que ha aparecido cada valor en cada clase. Cada *atributo numérico* debe llevar la cuenta de: (1) *la suma* de todos los valores aparecidos en cada clase junto con (2) *la suma del cuadrado* de cada uno de estos valores. Observe que esto último tiene relación con la asunción de que los valores numéricos siguen una distribución normal (sección 4.2.1 [EML]).

Ejercicio 37. ***[1h]*** *A partir de lo explicado más arriba, construya la función* `(INB <conceptoNB><ejemplo>)` *que devuelva el* `<conceptoNB>` *actualizado por el nuevo* `<ejemplo>`*. Por ejemplo,*

```
> (INB (nuevo-conceptoNB (car ejemplos)) '(soleado 25 30 no +))
((+ 1
  ((soleado . 1) (nublado . 0) (lluvioso . 0))
   (numerico 25 625)
   (numerico 30 900)
  ((si . 0) (no . 1)))
 (- 0
  ((soleado . 0) (nublado . 0) (lluvioso . 0))
   (numerico 0 0)
   (numerico 0 0)
  ((si . 0) (no . 0))))
```

Ejercicio 38. ***[5']*** *A partir del ejercicio anterior, construya la función* `(NB <ejemplos>)` *que devuelva el concepto NB correspondiente.*

Ejercicio 39. ***[2h]*** *Como ya hemos dicho, asumimos que los valores numéricos siguen una distribución normal $N(\mu, \sigma)$. Pero además asumiremos μ igual a la media muestral $\mu = \overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i$ y σ^2 igual a la cuasi-varianza muestral $S^2 = \frac{1}{n-1}\sum_{i=1}^{n}(X_i - \overline{X})^2$ de entre los valores aparecidos para dicho atributo numérico. Con esta información, construya* `(media <x><n>)` *que devuelve μ, donde* `<x>` *es la suma de los valores aparecidos hasta el momento en un atributo numérico (primer valor de las listas que comienzan*

por `numérico` *en los conceptos NB) y* `<n>` *cuenta el número de veces que ha aparecido la clase (segundo número en cada lista tras el nombre de la clase* `+` *ó* `-`*). También construya la función* `(varianza <x2><m><n>)` *que devuelve* $S^2 = \sigma^2$*, donde* `<x2>` *es la suma de los cuadrados de los valores aparecidos hasta el momento en un atributo numérico (segundo valor de las listas que comienzan por* `numérico` *en los conceptos NB),* `<m>` *es* μ *y* `<n>` *es el número de veces que ha aparecido la clase (segundo número en cada lista tras el nombre de la clase* `+` *ó* `-`*).*

Ejercicio 40. ***[1h]*** *Construya la función* `(probabilidades <clase><concepto NB><ejemplo-sin-clase>)` *que devuelva* como una lista *el conjunto de probabilidades de la clase* `<clase>` *dado el ejemplo* `<ejemplo-sin-clase>`*. En otras palabras, el conjunto de probabilidades* $p(< clase > |v_i)$ *donde* v_i *corresponde a cada uno de los valores del ejemplo* `<ejemplo-sin-clase>`*.*

Ejercicio 41. ***[30']*** *A partir de la función anterior, construya el intérprete* `(NBi <concepto NB><ejemplo-sin-clase>)` *y la función* `(match-NB <concepto NB><ejemplo-sin-clase>)`*.*

Ejercicio 42. ***[1h]*** *Para finalizar el bloque 2, mediante la función* `stratified-cross-validation` *con el parámetro* `folds` *a 10, compare la eficiencia de los algoritmos de* conceptos simples*: HGS, HCT, PCP, LMS, IB y NB sobre los 3 dominios: "buen día para salir al campo", "agaricus-lepiota" y "ionosphere". ¿Existe algún algoritmo que destaque en alguno de los 3 dominios? Trate de explicar los resultados obtenidos.*