Bloque 1. Tema 1: Introducción

Métodos de Aprendizaje en Inteligencia Artifcial.

UNED

Patricia Mayo Tejedor

# Representación de la experiencia mediante ejemplos de entrenamiento.

Ejercicio 1.

*Siguiendo con el concepto “buen día para salir al campo” ,reconsidere los atributos y valores que se muestran en los ejemplos anterioresy proponga los suyos propios. Tras ello, construya su propio conjunto de 20 ejemplos de entrenamiento diferentes. Finalmente, en el papel de tutor,asigne la clase (+ ó −) para cada uno de estos ejemplos. Debido a que los casos generados van a ser utilizados en sucesivos ejercicios, se recomienda escribir los ejemplos en un archivo de texto “.txt” (p.ej. en un editor como el “block de notas/notepad”). Escriba cada ejemplo de entrenamiento en una línea de texto diferente separando cada atributo/valor por medio de espacios. Utilice las primeras líneas del archivo para describir cada atributo: si es numérico, indicando este hecho; si es nominal, indicandolos posibles valores que éste puede adquirir.*

Para empezar la actividad se ha llevado a cabo una búsqueda por internet para saber cuáles atributos podrían ayudar mejor a predecir el tiempo. Se ha ampliado el atributo “perspectiva” ya dado como ejemplo, añadiendo variable, frio, niebla y calorSeco. También se han añadido las tendencias en temperatura y barómetro (que mide la presión atmosférica), ya que se entiende que la tendencia puede influir más que los valores en sí.

Los atributos y su definición de valores han quedado por tanto de esta manera:

perspectiva {lluvia, variable, bueno, frio, niebla, calorSeco }

temperaturaGrados {0..40}

temperaturaTendencia {bajando, estable, subiendo}

barometroTendencia {bajando, estable, subiendo}

humedadEn% {0..100}

viento {si, no}

clase {-, +}

y la tabla de 20 ejemplos resultante es la siguiente:

**perspectiva temperaturaGrados temperaturaTendencia barometroTendencia humedadEn% viento clase**

lluvia 5 bajando bajando 100 si -

lluvia 17 estable bajando 65 si -

variable 26 subiendo bajando 37 no +

lluvia 12 bajando estable 70 no -

variable 15 estable estable 46 no +

bueno 27 subiendo estable 54 si +

frio 5 bajando subiendo 20 no +

bueno 23 estable subiendo 45 no +

calorSeco 30 subiendo subiendo 10 si +

niebla 8 bajando bajando 95 no -

niebla 6 bajando bajando 88 si -

calorSeco 35 subiendo subiendo 10 si +

calorSeco 32 subiendo bajando 15 no +

buen0 20 subiendo estable 50 si +

bueno 24 estable subiendo 55 no +

frio 6 estable subiendo 40 si -

frio 1 bajando estable 30 no +

bueno 22 estable estable 49 si +

niebla 3 bajando estable 90 si -

lluvia 26 bajando bajando 80 si -

# Introducción a Scheme

## Ejercicio 2.

*Definir la función* (siguiente <lista>), *la cual recibe una lista de números enteros y devuelve una lista con cada uno de sus sucesores. Por ejemplo,* (siguiente '(1 3 5)) => (2 4 6)*.*

(define siguiente (lambda (lista)

(map (lambda (number)

(+ 1 number)) lista )))

Explicación:

Para realizar este ejercicio lo primero ha sido aprender como se define una lista o funciones en Scheme/Racket. (define <nombreDeFuncion> <funcion>)

Por lo que crear la lista nos queda: (define lista (list 1 2 3 4))

La función en sí se declara: (lambda (<parametroEntrada>) (<cuerpo/Salida>))

El <parametroEntrada> será la lista que creemos y para el <cuerpo/Salida> usamos la función map, que aplica un procedimiento determinado a cada uno de los elementos de una lista, devolviendo una lista nueva: (map <funcion> <lista>)

La función recibe un número y lo devuelve sumando uno: (lambda (number) (+ 1 number))

## Ejercicio 3.

*Definir la función (sumas <lista1><lista2>), la cual recibedos listas de números enteros y devuelve una lista con la suma de los elementosde las dos listas sumados uno a uno. Por ejemplo, (sumas '(1 35) '(2 4 6)) => (3 7 11).*

(define sumas (lambda (lista1 lista2)

(map (lambda (numero1 numero2)

(+ numero1 numero2)) lista1 lista2)))

Explicación:

De forma similar al ejercicio anterior, esta vez iterando en dos listas al mismo tiempo.

## Ejercicio 4.

*Construya la función (factorial <número>) de manera recursiva.Por ejemplo, (factorial 3) =>6*

(define factorial

(lambda (n)

(if (= n 0) 1

(\* n (factorial (- n 1))))))

Explicación:

En el capítulo 6 sobre recursión en el libro “Teach Yourself Scheme in Fixnum Days” (Sitaram, 2003), se explica paso a paso el ejemplo de factorial.

## Ejercicio 5.

*Construya la función obtener-al-azar0 que admita una lista cuyos elementos sean pares o conses (ver sección 6.3.2 de [R5RS], página25) que asocian un elemento a un número entero. La función debe devolver uno de estos elementos al azar. El número entero se debe relacionar con una frecuencia relativa: la frecuencia con debe aparecer su elemento asociadorespecto al resto de elementos cuando se escogen al azar.*

*Para añadir azar a Scheme, utilice la función random que devuelve un número aleatorio entre 0 y 1. El contenido teórico para desarrollar la función obtener-al-azar se denomina Método de la Transformada Inversa, y se puede encontrar en cualquier buen libro de probabilidad. En cualquier caso, en la plataforma del master se puede obtener el documento [ITM] que explica dicho método.*

## (define obtener-al-azar0

## (lambda (lista)

## (define frecuencias (map (lambda (par) (cdr par)) lista))

## (define total (apply + frecuencias))

## (define variableAleatoria (random))

## (define frecuenciaAleatoria (\* total variableAleatoria))

## (define encontrarLimite

## (lambda (index acumulada lista frecuencia)

## (if (>= acumulada frecuencia)

## (car (list-ref lista index))

## (encontrarLimite (+ index 1)

## (+ acumulada (cdr (list-ref lista (+ index 1))))

## lista

## frecuencia))))

## (encontrarLimite 0 (cdr (list-ref lista 0)) lista frecuenciaAleatoria)

## ))

Explicación:

Este ejercicio me ha costado bastante, porque al principio no entendía que tenía que hacer ni cuál era el objetivo. Como consejo para futuros años, explicaría mejor las instrucciones y el ejemplo, que debería funcionar sin tener que retocarlo. Lo primero ha sido coger todas las frecuencias y sumarlas, para ver el total de la distribución. Con la variable random cogemos una muestra aleatoria, y al multiplicarla por el total de frecuencias (que hemos sumado antes) convertimos el valor (entre 0 y 1) a uno equivalente en nuestra distribución (en nuestro caso entre 0 y 6). Por último solo queda buscar en que rango cae en la función inversa.

## Ejercicio 6.

*Haga más potente la función obtener-al-azar permitiendo que admita también una lista simple de elementos, comportándose como si cada elemento de esta lista tuviera asociada una misma frecuencia. Para ello, la función debe detectar si los elementos de la lista son pares (conses) elemento-número. Si lo son, ésta debe continuar con el código del ejercicio anterior. Si no, transformar la lista de elementos a una lista de pares elemento-numero, con dicho número constante para todos los pares, y continuar con el código del ejercicio anterior.*

(define obtener-al-azar

(lambda (lista)

(define listaCorrecta

(lambda (lista)

(if (pair? (list-ref lista 0))

(lista)

(map (lambda (elemento) (cons elemento 1)) lista))))

(obtener\_al\_azar0 (listaCorrecta lista))))

Explicación:

Se comprueba si el parámetro lista es correcto cogiendo el primer elemento y viendo si es de tipo par. Sí es tipo par entonces la lista es correcta tal cual. Si no lo es, rehacemos la lista para que contenga pares, asignando a cada elemento la frecuencia 1.

# Scheme como herramienta en métodos de aprendizaje para la IA

## Ejercicio 7.

*Actualice su archivo “ejemplos.txt” procedente del Ejercicio 1 con este nuevo formato. Para ello, genere un nuevo archivo denominado “ejemplos.scm”.*

(;principio de la lista de ejemplos.

(;definicion de los atributos

(perspectiva (lluvia variable bueno frio niebla calorSeco));nominal

(temperaturaGrados numerico);numerico

(temperaturaTendencia (bajando estable subiendo));nominal

(barometroTendencia (bajando estable subiendo));nominal

(humedadEn% numerico);numerico

(viento (si no));nominal

(clase (+ -));nominal

);fin definicion de atributos

;;Ejemplos

(lluvia 5 bajando bajando 100 si -)

(lluvia 17 estable bajando 65 si -)

(variable 26 subiendo bajando 37 no +)

(lluvia 12 bajando estable 70 no -)

(variable 15 estable estable 46 no +)

(bueno 27 subiendo estable 54 si +)

(frio 5 bajando subiendo 20 no +)

(bueno 23 estable subiendo 45 no +)

(calorSeco 30 subiendo subiendo 10 si +)

(niebla 8 bajando bajando 95 no -)

(niebla 6 bajando bajando 88 si -)

(calorSeco 35 subiendo subiendo 10 si +)

(calorSeco 32 subiendo bajando 15 no +)

(bueno 20 subiendo estable 50 si +)

(bueno 24 estable subiendo 55 no +)

(frio 6 estable subiendo 40 si -)

(frio 1 bajando estable 30 no +)

(bueno 22 estable estable 49 si +)

(niebla 3 bajando estable 90 si -)

(lluvia 26 bajando bajando 80 si -)

);fin de la lista de ejemplos

## Ejercicio 8.

Construya una función (leer-ejemplos <archivo>) que lea del archivo “ejemplos.scm” del ejercicio anterior y devuelva la lista de ejemplos. De hecho, se pide que dicha lista pueda quedar almacenada en la variable ejemplos para su uso en ejercicios posteriores. Utilice la función with-input-from-file (repasar la sección 6.6 del texto [R5RS]).

(define leer-ejemplos

(lambda (archivo)

(call-with-input-file archivo read)

))

Explicación:

Se debe especificar el camino completo al archivo, por ejemplo: (define ejemplos (leer-ejemplos "/Users/Patricia/Documents/UNED/Machine Learning/tema1/ejemplos.scm"))

## Ejercicio 9.

Construya la función anadir-ejemplo que admita como parámetros una listas de ejemplos más un nuevo ejemplo y devuelva una lista de ejemplos con el nuevo ejemplo incorporado.

(define anadir-ejemplo

(lambda (lista ejemplo)

(append lista (list ejemplo))))

Explicación:

Para llamar a la función se hace necesario que el ejemplo sea en sí una lista, por lo que la función se usaría de la siguiente manera: (anadir-ejemplo ejemplos **'**(niebla 3 bajando estable 90 si -))

## Ejercicio 10.

Construya la función atributo que admita 2 parámetros: el primero será el nombre del atributo pedido y el segundo la lista de ejemplos. La función debe devolver una lista con el valor que posee cada uno de los ejemplos en dicho atributo.

(define atributo

(lambda (nombre lista)

(define atributos (list-ref lista 0))

(define atributoEncontrado (assoc nombre atributos))

(list-ref atributoEncontrado 1)

))

Explicación:

Para llamar a la función tenemos que pasar el nombre del atributo con comilla, porque buscamos el valor del string en sí, y no el valor que pudiese guardar como variable.

## Ejercicio 11.

*Construya la función mezclar que admita como parámetros dos listas de ejemplos y devuelva otra con la mezcla de las estas dos.*

(define mezclar

(lambda (lista1 lista2)

(define atributos (list-ref lista1 0))

(define ejemplosJuntos

(append (list-tail lista1 1) (list-tail lista2 1)))

(append atributos ejemplosJuntos)

))

Explicación:

Se ha dado por entendido que ambas listas contienen la misma definición de atributos.

## Ejercicio 12.

Construya la función separar que admita como parámetros una lista de ejemplos junto con una proporción y devuelva una lista con dos listas de ejemplos. La primera lista debe tener el porcentaje de ejemplos indicado por el parámetro, la segunda el resto de los ejemplos disponibles. Añada azar a la composición de las sublistas. Para ello, utilice la función obtener-al-azar del ejercicio 6.

(define separar

(lambda (proporcion lista)

(define casos (list-tail lista 1))

(define tamanoSublista1 (exact-round (\* proporcion (length casos))))

(define crearSublista2

(lambda (contador casos)

(if (= contador 0)

casos

(crearSublista2 (- contador 1)

(remove (obtener-al-azar casos) casos))

)

))

(define sublista2 (crearSublista2 tamanoSublista1 casos))

(define sublista1 (set-subtract casos sublista2))

(list sublista1 sublista2)

))

Explicación:

Se ha optado por crear la primera sublista quitando elementos de la lista original. Esto permite asegurarnos de que no haya duplicados (como podría pasarnos si simplemente vamos creamos la lista según obtenemos valores de obtener-al-azar). Después simplemente usamos la función substract que nos devuelve los elementos restantes de la lista original.

## Ejercicio 13.

Construya la función folds que admita como parámetros una lista de ejemplos junto con un número natural y devuelva una lista formada por tantas sublistas (o folds) de ejemplos de tamaño similar (similar signica que varíen en tamaño en 1 elemento como máximo) como el parámetro indique. Añada azar a la composición de las sublistas. Para ello, utilice la función obtener-al-azar del ejercicio 6

(define folds

(lambda (numeroFolds lista)

(define casos (list-tail lista 1))

(define tamanoSublistas (quotient (length casos) numeroFolds))

(define sublistasCon1Mas (modulo (length casos) numeroFolds))

(define crearSublista

(lambda (contador casos sublista)

(if (= contador 0)

sublista

(let ((quitar (obtener-al-azar casos)))

(crearSublista (- contador 1)

(remove quitar casos)

(append sublista (list quitar)))))

))

; define cuantos extra elementos tendra la sublista

(define elementosExtras

(lambda (sublistasCon1Mas)

(if (> sublistasCon1Mas 0) 1

0)))

(define crearFolds

(lambda (numeroFolds tamanoSublistas sublistasCon1Mas foldsHechos casos)

(if (= 0 numeroFolds)

foldsHechos

(let ((tamanoSublistas tamanoSublistas)

(elementosExtras (elementosExtras sublistasCon1Mas)))

(define sublista (crearSublista (+ tamanoSublistas elementosExtras) casos '()))

(crearFolds (- numeroFolds 1)

tamanoSublistas

(- sublistasCon1Mas elementosExtras)

(append foldsHechos (list sublista))

(set-subtract casos sublista))))

))

(crearFolds numeroFolds tamanoSublistas sublistasCon1Mas '() casos)

))

Explicación:

Se ha modificado la función crearSublista del apartado anterior, ya que era ineficiente y necesitaba llevar la cuenta de cuántos elementos quedaban disponibles, en vez del tamaño de la sublista en sí. Usando los procedures de quotient y modulo podemos sacar los tamaños de cada sublista y cuántas listas tendrán que llevar un elemento extra cuando la división no es exacta.

## Ejercicio 14.

Construya la función stratify basada en la anterior función folds, pero restringida a que la proporción de ejemplos de cada clase se mantenga similar en cada fold o sublista de ejemplos y además similar a la existente en el conjunto total de ejemplos. Para ello, utilice la función obtener-al-azar de los ejercicios 5 y 6.

(define stratify

(lambda (numeroFolds ejemplos)

(define casos (list-tail ejemplos 1))

(define tamanoSublistas (quotient (length casos) numeroFolds))

(define sublistasCon1Mas (modulo (length casos) numeroFolds))

(define tiposDeClases (atributo 'clase ejemplos))

; PREPARACION

; Devuelve una lista con tantos elementos como clases hay.

; Cada elemento contiene todos los casos de esa clase

(define separarClases

(lambda (index tiposDeClases casos agrupacion)

(if (= index (length tiposDeClases))

agrupacion

(let ((claseAgrupada

(filter (lambda (caso)

(eqv? (list-ref caso (- (length caso) 1)) (list-ref tiposDeClases index))) casos)))

(separarClases (+ index 1) tiposDeClases casos (append agrupacion (list claseAgrupada)))))))

; Devuelve una lista con tantos elementos como clases hay. Cada elemento es un par, con la clase y la frecuencia en la que aparece en la lista

(define paresClaseFrecuencia

(lambda (tiposDeClases clasesAgrupadas)

(map (lambda (tipo grupo)(cons tipo (length grupo))) tiposDeClases clasesAgrupadas)))

; MANIPULACION DE CASOS

; Elimina el caso dado de las clases agrupadas

(define quitarCaso

(lambda (casoParaEliminar clasesAgrupadas)

(map (lambda (casosDeClase)

(remove casoParaEliminar casosDeClase))

clasesAgrupadas)))

; Encuentra todos los casos de una clase. Si no quedan mas casos de esa clase, escoge de otra al azar

(define encontrarCasosDeClase

(lambda (clase clasesAgrupadas clasesConFrecuencia)

(define casosDeClaseEncontrados

(filter

(lambda (casosDeClase)

(if (= (length casosDeClase) 0)

#f

(let\* ([primerCaso (list-ref casosDeClase 0)]

[claseDelPrimerCaso (list-ref primerCaso (- (length primerCaso) 1))])

(eq? clase claseDelPrimerCaso))))

clasesAgrupadas))

(if (= (length casosDeClaseEncontrados) 0)

(encontrarCasosDeClase (obtener-al-azar0 clasesConFrecuencia) clasesAgrupadas clasesConFrecuencia)

(list-ref casosDeClaseEncontrados 0 ))))

; Devuelve un elemento al azar de la clase dada

(define obtenerAlAzarDeClase

(lambda (clase clasesAgrupadas)

(define encontrarCasosDeClase

(filter (lambda (casosDeClase)

(define primerCaso (list-ref casosDeClase 0))

(define claseDelPrimerCaso (list-ref primerCaso (- (length primerCaso) 1)))

(eq? clase claseDelPrimerCaso))

clasesAgrupadas))

(obtener-al-azar (list-ref encontrarCasosDeClase 0 ))))

; CREACION DE FOLDS

; Define cuantos extra elementos tendra una sublista (cuando no se puedan dividir exactos)

(define elementosExtras

(lambda (sublistasCon1Mas)

(if (> sublistasCon1Mas 0) 1

0)))

; Dado el tamanio deseado, se crea una sublista con proporcion ademas de devolver los casos sobrantes

(define crearSublista

(lambda (numeroElementos casosAgrupadosPorClase paresClaseFrecuencia sublista)

(if (= numeroElementos 0)

(list sublista casosAgrupadosPorClase)

(let\* ([claseElegida (obtener-al-azar0 paresClaseFrecuencia)]

[casoElegido (obtener-al-azar (encontrarCasosDeClase claseElegida casosAgrupadosPorClase paresClaseFrecuencia))])

(crearSublista (- numeroElementos 1)

(quitarCaso casoElegido casosAgrupadosPorClase)

paresClaseFrecuencia

(append sublista (list casoElegido)))))

))

; Se crean los folds de manera recursiva

(define crearFolds

(lambda (numeroFolds tamanoSublistas sublistasCon1Mas foldsHechos casosAgrupadosPorClase paresClaseFrecuencia)

(if (= 0 numeroFolds)

foldsHechos

(let ((tamanoSublistas tamanoSublistas)

(elementosExtras (elementosExtras sublistasCon1Mas)))

(define separarSublistaDeCasos (crearSublista (+ tamanoSublistas elementosExtras) casosAgrupadosPorClase paresClaseFrecuencia '()))

(define sublista (list-ref separarSublistaDeCasos 0))

(define casosSobrantes (list-ref separarSublistaDeCasos 1))

(crearFolds (- numeroFolds 1)

tamanoSublistas

(- sublistasCon1Mas elementosExtras)

(append foldsHechos (list sublista))

casosSobrantes

paresClaseFrecuencia)))

))

(define casosAgrupadosPorClase (separarClases 0 tiposDeClases casos '()))

(define clasesConFrecuencia (paresClaseFrecuencia tiposDeClases casosAgrupadosPorClase))

(crearFolds numeroFolds tamanoSublistas sublistasCon1Mas '() casosAgrupadosPorClase clasesConFrecuencia)

)

)

# Construcción del primer algoritmo de aprendizaje de conceptos

En esta sección daremos directrices generales para poder codificar algoritmos que inducen conceptos a partir de ejemplos de entrenamiento. No obstante, infinitos conceptos pueden ser inducidos de un mismo conjunto de ejemplos de entrenamiento. Es por esto que dedicaremos la próxima sección(sección 6) a medir la capacidad de un algoritmo para generar conceptos precisos. Antes de pasar a codificar un algoritmo de aprendizaje, el primer paso debe ser establecer la manera de representar el concepto inducido a devolver. En otras palabras, debemos escoger el lenguaje de descripción de conceptos

## Ejercico 15

Tras ejecutar los comandos anteriores, comente las diferencias existentes entre el contenido de la variable ejemplos y el contenido de la variable extension. Existen errores? es natural que existan?

Si hay errores y es natural que hayan, el intérprite por extension simplemente ha entendido que todos los casos son de la clase mayoritaria (en nuestros ejemplos sería la clase “+”), por lo que el resto de clases han salido como errores.

(((perspectiva (lluvia variable bueno frio niebla calorSeco))

(temperaturaGrados numerico)

(temperaturaTendencia (bajando estable subiendo))

(barometroTendencia (bajando estable subiendo))

(humedadEn% numerico)

(viento (si no))

+)

(lluvia 5 bajando bajando 100 si +)

(lluvia 17 estable bajando 65 si +)

(variable 26 subiendo bajando 37 no +)

(lluvia 12 bajando estable 70 no +)

(variable 15 estable estable 46 no +)

(bueno 27 subiendo estable 54 si +)

(frio 5 bajando subiendo 20 no +)

(bueno 23 estable subiendo 45 no +)

(calorSeco 30 subiendo subiendo 10 si +)

(niebla 8 bajando bajando 95 no +)

(niebla 6 bajando bajando 88 si +)

(calorSeco 35 subiendo subiendo 10 si +)

(calorSeco 32 subiendo bajando 15 no +)

(bueno 20 subiendo estable 50 si +)

(bueno 24 estable subiendo 55 no +)

(frio 6 estable subiendo 40 si +)

(frio 1 bajando estable 30 no +)

(bueno 22 estable estable 49 si +)

(niebla 3 bajando estable 90 si +)

(lluvia 26 bajando bajando 80 si +))

## Ejercicio 16.

El número de aciertos o errores depende del tamaño de la muestra de ejemplos existentes. Por tanto, una buena medida de lo ajustado que resulta un determinado algoritmo a los casos que se le presentanes la relación de estos respecto al número de casos presentados. Se denomina precision a la relación de aciertos respecto al número de casos quese le han presentado precision = aciertos/casos . Se denomina error a su opuesto: error= errores/casos . En nuestro ejemplo, si existen 2 aciertos respecto a 3 casos presentados, la precisión será de 2/3 = 0’666=67 %. El error será el contrario, 1/3 = 0’333=33 %. Como ejercicio, calcule la precisión y el error deA0/A0i en su propia muestra de ejemplos de entrenamiento (procedente delos ejercicios 7 y 8).

; Precision es el numero de aciertos dividido por el numero total de casos

(define precision

(lambda (ejemplos extension)

(define casosEjemplos (cdr ejemplos))

(define casosExtension (cdr extension))

(define numeroCasos (length casosEjemplos))

(define numeroAciertos (length

(filter (lambda (caso) caso)

(map (lambda (casoEjemplo casoExtension)

(eq? (last casoEjemplo) (last casoExtension)))

casosEjemplos casosExtension))))

(/ numeroAciertos numeroCasos)))

; Error es el numero de errores dividido por el numero total de casos, o lo que es lo mismo, 1 - precision

(define error

(lambda (ejemplos extension)

(define precisionTotal (precision ejemplos extension))

(- 1 precisionTotal)))

## Ejercicio 17.

Haga responder al intérprete A0i a partir de 5 días que usted imagine y que no estén presentes en el archivo de ejemplos del ejercicio 7. Como tutor que es usted, quien conoce con la máxima precisión el concepto”buen día para salir al campo", reflexione acerca de la respuesta dada por el par A0/A0i al tratar de clasicar estos 5 nuevos días imaginarios. Observe que estos nuevos días no están incorporados al concepto aprendido, por tanto ¿es comparable esta precisión con la que aparece en el ejercicio 16?. Como ayuda, cree el archivo ¨ejemplos2.scm¨ con estos 5 nuevos ejemplos (así podrá utilizarlos en ejercicios posteriores). Tras ello, utilice la función del ejercicio 8 para cargarlos en una nueva variable.

Ejemplos:

(lluvia 2 bajando bajando 75 si -)

(lluvia 28 estable bajando 65 no -)

(variable 22 subiendo bajando 30 no +)

(niebla 8 bajando estable 60 no -)

(bueno 26 estable subiendo 43 si +)

Ejercicio:

;; Preparar ejemplos

(define ejemplos2 (leer-ejemplos direccionEjemplos2))

(define ejemplos-sin-clase2 (map (lambda(x) (drop-right x 1)) ejemplos2))

;; Pasar ejemplos por el interprete A0i

(define extension2 (map (lambda(x) (A0i esencia x)) ejemplos-sin-clase2))

;; Definir precision y error

(define precisionEjemplos2 (precision ejemplos2 extension2))

(define errorEjemplo2 (error ejemplos2 extension2))

Resultados:

> (precision ejemplos extension)

3/5

> precisionEjemplos2

2/5

Estos nuevos ejemplos no estaban entre los conceptos aprendidos, he puesto aposta más ejemplos con días malos (al contrario que en los primeros ejemplos donde predominaban los días buenos), por lo que la precisión no ha sido la misma. Aún así no ha estado mal del todo, y en un set más grande seguramente la precisiones se equilibrarían.

## Ejercicio 18.

Codique el par de funciones A1/A1i, donde A1 es similar a A0 exceptuando que el concepto devuelto debe ser el contenido de la variable clases-contabilizadas en vez del contenido de la variable concepto. Por ejemplo, teniendo en cuenta los ejemplos de este documento, A1 devolverá ((+ . 2)(- . 1)) en vez de devolver únicamente + como hace A0. Además, A1i deberá clasificar cada nuevo ejemplo de entrenamiento en una de las dosclases (+ o -) probabilidad proporcional a la frecuencia de cada clase. Para ello, utilice la función obtener-al-azar del ejercicio 5.

; A1

(define (A1 ejemplos)

(let\*

(;;Asignacion de VARIABLES locales.

;;==================================

(casos (list-tail ejemplos 1))

(tiposDeClases (atributo 'clase ejemplos))

; Devuelve una lista con tantos elementos como clases hay. Cada elemento es un par, con la clase y la frecuencia en la que aparece en la lista

(paresClaseFrecuencia

(lambda (tiposDeClases clasesAgrupadas)

(map (lambda (tipo grupo)(cons tipo (length grupo))) tiposDeClases clasesAgrupadas)))

)

; Devuelve una lista con tantos elementos como clases hay. Cada elemento contiene todos los casos de esa clase

(define separarClases

(lambda (index tiposDeClases casos agrupacion)

(if (= index (length tiposDeClases))

agrupacion

(let ((claseAgrupada (filter (lambda (caso)

(eqv? (list-ref caso (- (length caso) 1)) (list-ref tiposDeClases index)))

casos)))

(separarClases (+ index 1) tiposDeClases casos (append agrupacion (list claseAgrupada)))))))

(define casosAgrupadosPorClase (separarClases 0 tiposDeClases casos '()))

(paresClaseFrecuencia tiposDeClases casosAgrupadosPorClase))

)

; A1i

(define (A1i paresClaseFrecuencia ejemplo-sin-clase)

(append ejemplo-sin-clase (list (obtener-al-azar0 paresClaseFrecuencia))))

Resultados:

(define esenciaA1 (A1 ejemplos))

(define extensionA1i (map (lambda(x) (A1i esenciaA1 x)) ejemplos-sin-clase))

> extensionA1i

'(((perspectiva (lluvia variable bueno frio niebla calorSeco))

(temperaturaGrados numerico)

(temperaturaTendencia (bajando estable subiendo))

(barometroTendencia (bajando estable subiendo))

(humedadEn% numerico)

(viento (si no))

+)

(lluvia 5 bajando bajando 100 si +)

(lluvia 17 estable bajando 65 si +)

(variable 26 subiendo bajando 37 no -)

(lluvia 12 bajando estable 70 no -)

(variable 15 estable estable 46 no +)

(bueno 27 subiendo estable 54 si +)

(frio 5 bajando subiendo 20 no -)

(bueno 23 estable subiendo 45 no +)

(calorSeco 30 subiendo subiendo 10 si +)

(niebla 8 bajando bajando 95 no -)

(niebla 6 bajando bajando 88 si +)

(calorSeco 35 subiendo subiendo 10 si +)

(calorSeco 32 subiendo bajando 15 no +)

(bueno 20 subiendo estable 50 si +)

(bueno 24 estable subiendo 55 no -)

(frio 6 estable subiendo 40 si -)

(frio 1 bajando estable 30 no +)

(bueno 22 estable estable 49 si +)

(niebla 3 bajando estable 90 si +)

(lluvia 26 bajando bajando 80 si +))

(define precisionEjemplosA1 (precision ejemplos extensionA1i))

> precisionEjemplosA1

3/5

## Ejercicio 19.

Mediante los 5 ejemplos (días) anteriores compare la precisión de A0/A0i respecto a la precisión de A1/A1i. ¿Qué algoritmo consideraque induce mejores conceptos? ¿Considera que ha sido rentable aumentar el lenguaje de descripción de conceptos de A0/A0i a A1/A1i (haciendo el algoritmo A1/A1i más complejo)?.

;; Pasar ejemplos por el interprete A1i

(define extensionA1iEjemplos2 (map (lambda(x) (A1i esenciaA1 x)) ejemplos-sin-clase2))

;; Definir precision y error

(define precisionA1Ejemplos2 (precision ejemplos2 extensionA1iEjemplos2))

(define errorA1Ejemplo2 (error ejemplos2 extensionA1iEjemplos2))

)

Resultados

> extensionA1iEjemplos2

'(((perspectiva (lluvia variable bueno frio niebla calorSeco))

(temperaturaGrados numerico)

(temperaturaTendencia (bajando estable subiendo))

(barometroTendencia (bajando estable subiendo))

(humedadEn% numerico)

(viento (si no))

-)

(lluvia 2 bajando bajando 75 si +)

(lluvia 28 estable bajando 65 no -)

(variable 22 subiendo bajando 30 no +)

(niebla 8 bajando estable 60 no -)

(bueno 26 estable subiendo 43 si -))

> precisionA1Ejemplos2

3/5

Aunque no se han apreciado diferencias en la precisión cuando se ha comparado con los ejemplos de entrenamiento, se puede ver que sí que ha aumentado la precisión (de 2/5 a 3/5) cuando se han introducido casos nuevos. El algoritmo A1/A1i es más general y aplicable incluso con casos nuevos no entrenados por lo que si se considera rentable.

# Evaluación del aprendizaje

## Ejercicio 22.

*Construya las funciones resustitution y leave-one-out que admitan 3 parámetros: la función de entrenamiento, la función intérprete y un conjunto de ejemplos de entrenamiento. Dichas funciones deben devolverla precisión total del algoritmo de aprendizaje sobre dichos ejemplos. Compruebe mediante estas funciones la precisión de los algoritmos A0/A0i, A1/A1i e IA1/A1i mediante los 25 ejemplos creados entre los ejercicios 7 y17. Para ello, recomendamos utilizar las funciones creadas en los ejercicios 8 y 11. Por ejemplo, una llamada correcta sería la siguiente: # (resustitution A0 A0i ejemplos)*

; Se usan todos los ejemplos para entrenar y para evaluar

(define ejemplosJuntos (mezclar ejemplos ejemplos2))

(define resustitution

(lambda (esencia interprete ejemplos)

(define ejemplos-sin-clase (map (lambda(x) (drop-right x 1)) ejemplos))

(define cogerEsencia (esencia ejemplos))

(define cogerExtension (map (lambda(x) (interprete cogerEsencia x)) ejemplos-sin-clase))

(precision ejemplos cogerExtension)

))

; Se usan cada vez solo un ejemplo para evaluar (con todos los ejemplos)

(define leave-one-out

(lambda (entrenamiento interprete ejemplos)

(define casos (list-tail ejemplos 1))

(define evaluarCaso

(lambda (casoParaEvaluar)

(define casosDeEntrenamiento (remq casoParaEvaluar ejemplos))

(define casoSinClase (map (lambda(x) (drop-right x 1)) (list casoParaEvaluar)))

(define entrenar (entrenamiento casosDeEntrenamiento))

(define interpretar (map (lambda(x) (interprete entrenar x)) casoSinClase))

(eq? (last casoParaEvaluar) (last (first interpretar)))

))

(define numeroAciertos (length (filter (lambda (caso)

(evaluarCaso caso))

casos)))

(exact->inexact (/ numeroAciertos (length casos)))))

Resultados: Como las funciones A1 y A1i se basan en la clasificación cogida al azar los valores cambian, pero se puede ver que se consigue entre similar o mejor precisión.

|  |  |
| --- | --- |
| Resustitution | Leave one out |
| (resustitution A0 A0i ejemplosJuntos)  0.56  > (resustitution A1 A1i ejemplosJuntos)  0.56  > (resustitution A1 A1i ejemplosJuntos)  0.52  > (resustitution A1 A1i ejemplosJuntos)  0.76 | > (leave-one-out A0 A0i ejemplosJuntos)  0.56  > (leave-one-out A1 A1i ejemplosJuntos)  0.68  > (leave-one-out A1 A1i ejemplosJuntos)  0.56  > (leave-one-out A1 A1i ejemplosJuntos)  0.48 |

## Ejercicio 23.

Construya la función holdout que admita 4 parámetros: la función de entrenamiento, la función intérprete, un conjunto de ejemplos de entrenamiento y un conjunto de ejemplos de evaluación. La función debe devolver la precisión total del algoritmo de aprendizaje sobre los ejemplos de evaluación. Compruebe mediante esta función la precisión de los algoritmos A0/A0i, A1/A1i e IA1/A1i utilizando la relación dos tercios/un tercio para entrenar y evaluar mediante los 25 ejemplos creados entre los ejercicios 7 y17. Para ello, recomendamos utilizar las funciones creadas en los ejercicios 8, 11 y 12. Por ejemplo, una llamada correcta sería la siguiente:

# (holdout A0 A0i ejemplos '((soleado 30 50 si +)(lluvioso 4 90 -)))

(define ejemplosSeparados (separar 0.67 ejemplosJuntos))

(define atributos (car ejemplosJuntos))

(define ejemplosEntrenamiento (append (list atributos) (list-ref ejemplosSeparados 0)))

(define ejemplosEvaluacion (append (list atributos) (list-ref ejemplosSeparados 1)))

(define holdout

(lambda (entrenamiento interprete ejemplosEntrenamiento ejemplosEvaluacion)

(define ejemplosEvaluacionSinClase (map (lambda(x) (drop-right x 1)) ejemplosEvaluacion))

(define entrenar (entrenamiento ejemplosEntrenamiento))

(define interpretar (map (lambda(x) (interprete entrenar x)) ejemplosEvaluacionSinClase))

(precision ejemplosEvaluacion interpretar)))

|  |
| --- |
| Holdout 2/3 – 1/3 |
| > (holdout A0 A0i ejemplosEntrenamiento ejemplosEvaluacion)  0.625  > (holdout A1 A1i ejemplosEntrenamiento ejemplosEvaluacion)  0.5  > (holdout A1 A1i ejemplosEntrenamiento ejemplosEvaluacion)  0.625  > (holdout A1 A1i ejemplosEntrenamiento ejemplosEvaluacion)  0.375 |

## Ejercicio 24.

*A partir de los ejercicios 13 y 14, construya las funciones cross-validation y stratified-cross-validation que admitan 4 parámetros: la función de entrenamiento, la función intérprete, un conjunto de ejemplos de entrenamiento y el número de folds. La función debe devolver la precisión total del algoritmo de aprendizaje sobre dichos ejemplos. Compruebe mediante estas funciones, con el parámetro folds a 10, la precisión de los algoritmos A0/A0i, A1/A1i e IA1/A1i mediante los 25 ejemplos creados entre los ejercicios 7 y 17. Por ejemplo, una llamada correcta (respecto a los ejemplos de este documento) sería la siguiente:*

*# (= (cross-validation A0 A0i ejemplos 3)(leave-one-out A0 A0i ejemplos))*

*#t*

(define cross-validation

(lambda (entrenamiento interprete ejemplos numeroFolds)

(define casos (list-tail ejemplos 1))

(define atributos (car ejemplos))

(define ejemplosEnFolds (folds numeroFolds ejemplos))

(define evaluarFold

(lambda (foldParaEvaluar)

(define foldsDeEntrenamiento (remq foldParaEvaluar ejemplosEnFolds))

(define ejemplosEntrenamiento (append (list atributos) (append\* foldsDeEntrenamiento)))

(define ejemplosEvaluacion (append (list atributos) foldParaEvaluar))

(holdout entrenamiento interprete ejemplosEntrenamiento ejemplosEvaluacion)

))

(define sumaPrecisiones (apply + (map (lambda (fold)

(evaluarFold fold))

ejemplosEnFolds)))

(exact->inexact (/ sumaPrecisiones numeroFolds))))

(define stratified-cross-validation

(lambda (entrenamiento interprete ejemplos numeroFolds)

(define atributos (car ejemplos))

(define ejemplosEnFolds (stratify numeroFolds ejemplos))

(define evaluarFold

(lambda (foldParaEvaluar)

(define foldsDeEntrenamiento (remq foldParaEvaluar ejemplosEnFolds))

(define ejemplosEntrenamiento (append (list atributos) (append\* foldsDeEntrenamiento)))

(define ejemplosEvaluacion (append (list atributos) foldParaEvaluar))

(holdout entrenamiento interprete ejemplosEntrenamiento ejemplosEvaluacion)))

(define sumaPrecisiones (apply + (map (lambda (fold)

(evaluarFold fold)) ejemplosEnFolds)))

(exact->inexact (/ sumaPrecisiones numeroFolds))))

|  |  |
| --- | --- |
| cross-validation | stratified-cross-validation |
| > (cross-validation A1 A1i ejemplosJuntos 3)  0.6388888888888888  > (cross-validation A0 A0i ejemplosJuntos 25)  0.56  > (leave-one-out A0 A0i ejemplosJuntos)  0.56 | > (stratified-cross-validation A0 A0i ejemplosJuntos 3)  0.5555555555555555  > (stratified-cross-validation A1 A1i ejemplosJuntos 3)  0.4351851851851852 |

# Bibliography

Sitaram, D. (2003). *Teach Yourself Scheme in Fixnum Days.* Obtenido de http://download.plt-scheme.org/doc/205/pdf/t-y-scheme.pdf