# Memoria Práctica 1

## Ejercicio 1:

Apartado 1:

Para este ejercicio hemos creado una función auxiliar que calcula el producto escalar de dos vectores:

*(defun producto-escalar-rec (x y)*

*(if (or (null x) (null y))*

*0*

*(+ (\* (first x) (first y)) (producto-escalar-rec (rest x) (rest y))))*

*)*

Y luego una función que haciendo uso de esa función calcule la distancia coseno entre dos vectores de manera recursiva:

*(defun cosine-distance-rec (x y)*

*(if (or (null x) (null y) (= 0 (\* (producto-escalar-rec x x) (producto-escalar-rec y y))))*

*0*

*(- 1 (/ (producto-escalar-rec x y) (\* (sqrt (producto-escalar-rec x x)) (sqrt (producto-escalar-rec y y))))))*

*)*

Luego haciendo uso de las funciones mapcar hemos reducido la función de la distancia coseno:

*(defun cosine-distance-mapcar (x y)*

*(if (or (null x) (null y) (= 0 (\* (apply #'+ (mapcar #'\* x x)) (apply #'+ (mapcar #'\* y y)))))*

*0*

*(- 1 (/ (apply #'+ (mapcar #'\* x y)) (\* (sqrt (apply #'+ (mapcar #'\* x x))) (sqrt (apply #'+ (mapcar #'\* y y)))))))*

*)*

Los resultados a las pruebas pedidas en el ejercicio 1.1 son:

1. (cosine-distance ’(1 2) ’(1 2 3)) → 0.40238577

2. (cosine-distance nil ’(1 2 3)) → 0

3. (cosine-distance ’() ’()) → 0

4. (cosine-distance ’(0 0) ’(0 0)) → 0

### Apartado 2:

En este apartado se pide crear una función que ordene los vectores de la lista según el valor de confianza con el vector de referencia, siempre y cuando superen el valor de confianza dado como argumento de entrada:

Para ello hemos creado dos funciones auxiliares: la primera para insertar en orden en una lista:

*(defun insert-ordered-cosine-distance (reference vector lst-of-vectors)*

*(cond ((null vector)*

*lst-of-vectors)*

*((null lst-of-vectors)*

*(cons vector lst-of-vectors))*

*((< (cosine-distance-mapcar reference vector) (cosine-distance-mapcar reference (first lst-of-vectors)))*

*(cons vector lst-of-vectors))*

*(t*

*(cons (first lst-of-vectors) (insert-ordered-cosine-distance reference vector (rest lst-of-vectors)))))*

*)*

Y la siguiente como parte recursiva de la función principal:

*(defun order-vectors-cosine-distance-rec (vector lst-of-vectors &optional (confidence-level 0))*

*(cond ((null lst-of-vectors)*

*nil)*

*(t*

*(insert-ordered-cosine-distance vector (first lst-of-vectors) (order-vectors-cosine-distance-rec vector (rest lst-of-vectors) confidence-level))))*

*)*

Por lo que al final la función principal se dedicaría solamente a llamar a la recursiva y esta a la de insertar en orden:

*(defun order-vectors-cosine-distance (vector lst-of-vectors &optional (confidence-level 0))*

*(order-vectors-cosine-distance-rec vector*

*(remove-if (lambda (v) (< (- 1 confidence-level) (cosine-distance-mapcar vector v))) lst-of-vectors)*

*confidence-level)*

*)*

La salida a los ejercicios puestos es:

(order-vectors-cosine-distance ’(1 2 3) ’()) → NIL

(order-vectors-cosine-distance ’() ’((4 3 2) (1 2 3))) → ((1 2 3) (4 3 2))

### Apartado 3:

En este ejercicio se pide que dada una lista de textos y de categorías se le asigne a cada texto la categoría más cercana según la función de distancia asignada, para ello hemos creado dos funciones, la primera que dado un texto y una lista de categorías saque la que le corresponde al texto y la otra que dada una lista de textos y otra de categorías llame repetidamente a la primera para que saque las categorías.

Primera función:

*(defun get-category (categories text distance-measure minimum)*

*(cond ((null categories)*

*(list (first minimum) (funcall distance-measure text minimum)))*

*((null minimum)*

*(get-category (rest categories) text distance-measure (first categories)))*

*(t*

*(if (< (funcall distance-measure text (first categories)) (funcall distance-measure text minimum))*

*(get-category (rest categories) text distance-measure (first categories))*

*(get-category (rest categories) text distance-measure minimum))))*

*)*

Segunda función:

*(defun get-vectors-category (categories texts distance-measure)*

*(cond ((null texts)*

*nil)*

*(t*

*(cons (get-category categories (first texts) distance-measure nil) (get-vectors-category categories (rest texts) distance-measure))))*

*)*

Y los resultados de los ejercicios propuestos es la siguiente.

(get-vectors-category ’(()) ’(()) #’cosine-distance-mapcar) → (NIL 0)

(get-vectors-category ’((1 4 2) (2 1 2)) ’((1 1 2 3)) #’cosine-distance-mapcar) → (2 0.39753592)

(get-vectors-category ’(()) ’((1 1 2 3) (2 4 5 6)) #’cosine-distance-mapcar) → ((NIL 0) (NIL 0))

# Ejercicio 2:

### Apartado 1: