Programación Lógica y Funcional

Lisp en la IA

Dr. Alejandro Guerra-Hernández

Instituto de Investigaciones en Inteligencia Artificial
Universidad Veracruzana
Campus Sur, Calle Paseo Lote II, Sección Segunda No 112,
Nuevo Xalapa, Xalapa, Ver., México 91097

mailto:aguerra@uv.mx
https://www.uv.mx/personal/aguerra/plf

Maestría en Inteligencia Artificial 2024

Universidad Veracruzana

Contenido

Universidad Veracruzana

DNA, Codones y Nucleótidos

- El Código Genético Universal suele representarse como un mapa de 64 codones a 20 aminoácidos.
- Un codón es una tripleta de nucleótidos $\langle N_1, N_2, N_3 \rangle$ donde $N_i \in \{A, C, G, T, \}$
- Los nucleótidos incluyen: Adenina (A), Citosina (C), Guanina (G), Timina (T).

Universidad Veracruzana

Código Genético Universal

- El mapa completo es como sigue:

```
BIOINFO> (print-tensor *gc-tensor*)
((T T T) PHE) ((T C T) SER) ((T A T) TYR) ((T G T) CYS)
((T T C) PHE) ((T C C) SER) ((T A C) TYR) ((T G C) CYS)
((T T A) LEU) ((T C A) SER) ((T A A) TER) ((T G A) TER)
((T T G) LEU) ((T C G) SER) ((T A G) TER) ((T G G) TRP)
((C T T) LEU) ((C C T) PRO) ((C A T) HIS) ((C G T) ARG)
((C T C) LEU) ((C C C) PRO) ((C A C) HIS) ((C G C) ARG)
((C T A) LEU) ((C C A) PRO) ((C A A) GLN) ((C G A) ARG)
((C T G) LEU) ((C C G) PRO) ((C A G) GLN) ((C G G) ARG)
((A T T) ILE) ((A C T) THR) ((A A T) ASN) ((A G T) SER)
((A T C) ILE) ((A C C) THR) ((A A C) ASN) ((A G C) SER)
((A T A) ILE) ((A C A) THR) ((A A A) LYS) ((A G A) ARG)
((A T G) MET) ((A C G) THR) ((A A G) LYS) ((A G G) ARG)
((G T T) VAL) ((G C T) ALA) ((G A T) ASP) ((G G T) GLY)
((G T C) VAL) ((G C C) ALA) ((G A C) ASP) ((G G C) GLY)
((G T A) VAL) ((G C A) ALA) ((G A A) GLU) ((G G A) GLY)
((G T G) VAL) ((G C G) ALA) ((G A G) GLU) ((G G G) GLY)
T
```

Universidad Veracruzana

Observaciones

- Observen que el *prompt* de Lisp es diferente. Esto indica que estamos en un paquete llamado `BIOINFO`.
- Los paquetes evitan los conflictos entre nombres de símbolos definidos en diferentes piezas de código independientes.
- La función `print-tensor` no está predefinida, la implementaremos más adelante.
- La variable `*gc-tensor*` denota una lista de listas donde cada elemento es un par codón-aminoácido correspondiente.
- Ejemplo. `((a g c) ser)`
- Es sobre estos elementos que vamos a operar en lo que sigue.

Universidad Veracruzana

Significancia

- La significancia de una base o de sus propiedades físico-químicas indica que tan importantes son estos elementos para la identidad de un aminoácido.
- La significancia de una base N_i, es fuertemente dependiente de su posición en el codón.
- Generalmente $N_2 > N_1 > N_3$.
- Con mayor probabilidad, las mutaciones en N_3 producirán el mismo aminoácido, u otro con propiedades físico-químicas similares; no así en N_2.

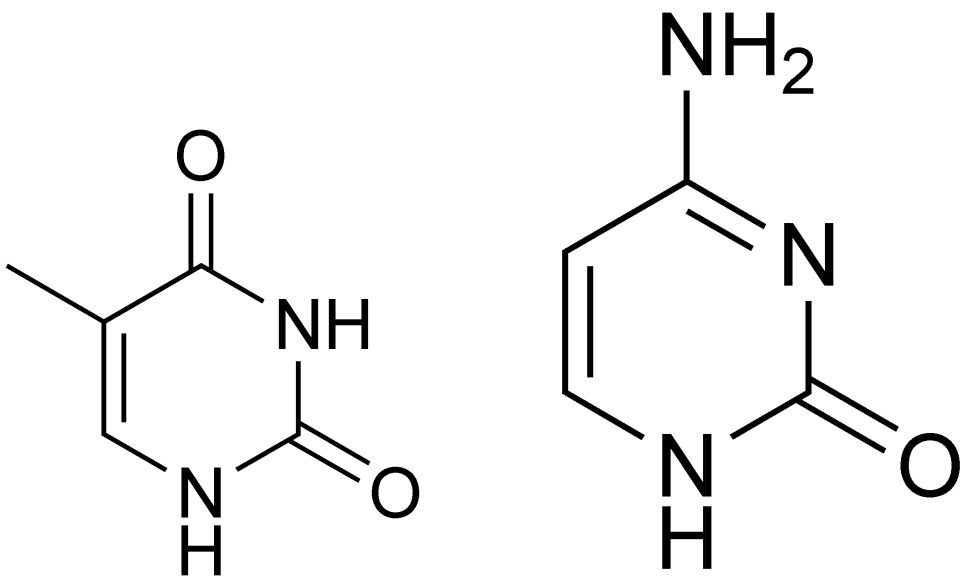
Universidad Veracruzana

Propiedades Físico-Químicas

- Es posible representar el Código Genético como una 6-tupla de la forma $\langle C_1, H_1, C_2, H_2, C_3, H_3 \rangle$.
- C_i representa la naturaleza química del nucleótido N_i, es decir, si es una pirimidina (Y) o una purina (R).
- H_i representa la fuerza del enlace de hidrógeno del nucleótido N_i y su complemento, es decir, enlace fuerte (S) o débil (W).

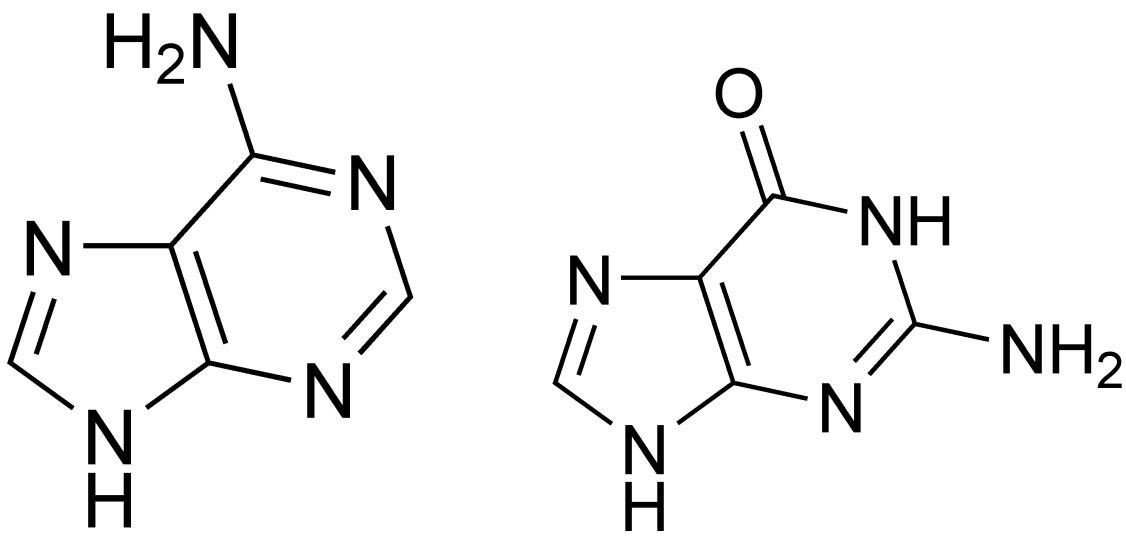
Universidad Veracruzana

Pirimidinas: Timina y Citosina



Universidad Veracruzana

Purinas: Adenina y Guanina



Universidad Veracruzana

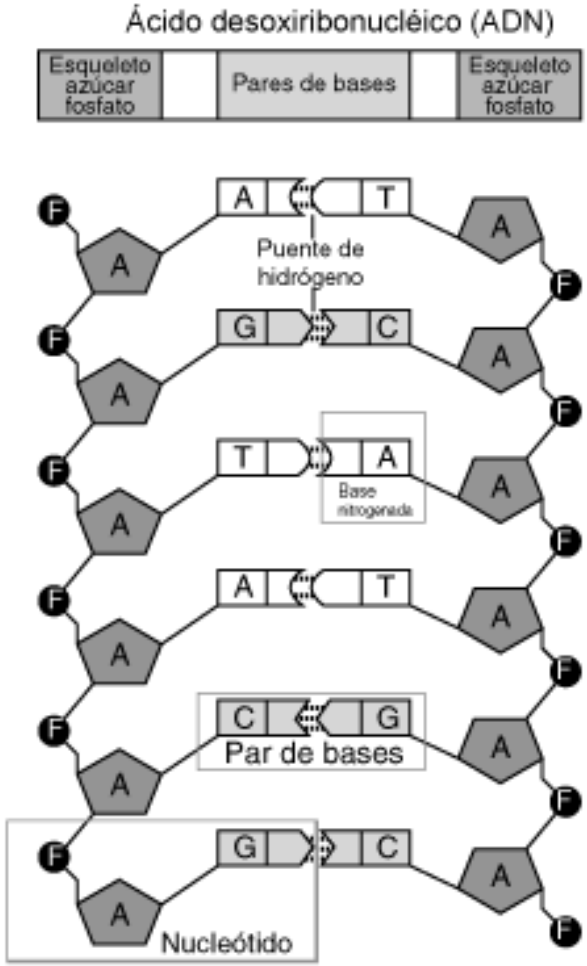
Fuerza de los Enlaces

A=T enlace débil (doble enlace de hidrógeno)

G≡C enlace fuerte (triple enlace de hidrógeno)

Universidad Veracruzana

Gráficamente



Universidad Veracruzana

Significancia de estas Propiedades

- Aquí el orden bajo significancia no es tan claro:
  - M.A. Jiménez-Montaño propone el siguiente orden de significancia:

$$C_2 > H_2 > C_1 > H_1 > C_3 > H_3$$

  - R. Swanson propone un orden alternativo:

$$C_2 > C_1 > H_2 > H_1 > C_3 > H_3$$

- ¿Cual es el bueno?
- La mayoría de estos ordenes fueron establecidos siguiendo criterios cualitativos, pero es posible seguir una aproximación cuantitativa en términos evolutivos [3, 2].

Universidad Veracruzana

Cuantificando la Similitud

- Se propone mutar un nucleótido seleccionado, ó una de sus propiedades físico-químicas, para cuantificar el efecto de tal cambio.
- La matriz $PAM250$ es una métrica ampliamente usada en análisis de similitud de secuencias genéticas.
- El elemento $s_{ij} \in PAM250$ es una cuantificación de la probabilidad de que un aminoácido i mute a j sobre un periodo evolutivo. Valores más pequeños (o negativos) reflejan disimilitud.
- Se pueden considerar otras matrices de similitud.

Universidad Veracruzana

Significancia de la posición de los nucleótidos

1. Una posición $i \in \{1, 2, 3\}$ es seleccionada.
2. 12 mapeos son posibles: $T \rightarrow \{C, A, G\}$; $C \rightarrow \{A, G, T\}$; $A \rightarrow \{G, T, C\}$; $G \rightarrow \{T, C, A\}$.
3. Los mapeos se aplicas a los 64 codones en la posición i, resultando en un código genético modificado $Gcode'$.
4. Los elementos s_{ij} de una matriz de similitud dan un valor numérico que mide la similitud de los aminoácidos $i \in GCode$ y $j \in GCode'$.

Universidad Veracruzana

Continuación...

5. La significancia global S de un mapeo es computada a partir de los valores s_{ij}, sumados en los 64 codones::

$$S = \sum_{i,j=1\ldots64} s_{GCode_i Gcode'_j} \tag{1}$$

6. La significancia promedio $\sum_{i=1\ldots3} S_i/12$ se obtiene considerando los 12 posibles mapeos para cada posición de los nucleótidos.

Universidad Veracruzana

Significancia de las propiedades de los nucleótidos

1. Una $i \in \{1, 2, 3\}$ y una propiedad del nucleótido C_i o H_i son seleccionados.
2. Dos mapeos son posibles, dependiendo de la propiedad seleccionada: $f_C : \{R \rightarrow Y, Y \rightarrow R\}$ o $f_H : \{W \rightarrow S, S \rightarrow W\}$. Basicamente, esto se corresponde con un cambio de bit de la propiedad seleccionada..
3. El mapeo correspondiente se aplica a los 64 codones en el Código Genético, resultando en un código modificado. table $GCode'$. Se procede como en los pasos 4 y 5 del experimento anterior.

Universidad Veracruzana

Exp. 1: Similitud para las posiciones

- Como se esperaba $N_2 > N_1 > N_3$ para todas las matrices consideradas:

Matriz	N_1	N_2	N_3
PAM250	235.91	220.08	281.08
BLOSUM62	242.25	220.83	292.17
Prlić	157.48	142.38	185.40
Miyazawa	27.26	21.91	31.96
Sánchez	87.69	90.88	73.14

- Nota. Para Sánchez, los valores mayores son más significativos. Para las otras matrices es el caso opuesto.

Universidad Veracruzana

Exp. 2: Similitud para las propiedades físico-químicas

- Valores y ordenes obtenidos:

Matriz	C_1	H_1	C_2	H_2	C_3	H_3
PAM250	-16.00	16.00	-60.00	-32.00	122.00	276.00
BLOSUM62	-2.00	1.00	-76.00	-66.00	162.00	302.00
Prlić	18.14	29.34	-26.93	-30.83	111.54	189.84
Miyazawa	7.06	7.02	-18.71	-13.70	21.06	30.70
Sánchez	173.79	135.56	196.88	140.20	100.42	70.88

Matriz	Orden
PAM250	$C_2 > H_2 > C_1 > H_1 > C_3 > H_3$
BLOSUM62	$C_2 > H_2 > C_1 > H_1 > C_3 > H_3$
Prlić	$C_2 > H_2 > C_1 > H_1 > C_3 > H_3$
Miyazawa	$C_2 > H_2 > H_1 > C_1 > C_3 > H_3$
Sánchez	$C_2 > C_1 > H_2 > H_1 > C_3 > H_3$

Universidad Veracruzana

Paquetes

- Vamos a crear un paquete para implementar nuestro cálculo de las significancias. El programa empezará así:

```
1  (defpackage :bioinfo
2    (:use :cl :ltk))
3
4  (in-package :bioinfo)
```

- Es necesario declarar el paquete y moverse dentro de él, antes de continuar con el resto de la implementación.
- `ltk` es la librería para el diseño de interfaces gráficas de basada en `tk` (Se debe instalar con quicklisp).
- Un paquete puede verse como un mapeo entre cadenas de texto y símbolos.

Universidad Veracruzana

Símbolos y paquetes

- Los símbolos en Lisp pueden ser:

  No calificados. No contienen : en su nombre. Denotan al símbolo en el paquete actual. Ej. `*gc-tensor*`

  Calificados. Contienen al menos un : en su nombre. Denotan al símbolo en un paquete dado. Pueden ser:

  Externos. Explicitamente exportado por el paquete, *e.g.*, `bioinfo:*gc-tensor*`

  Internos. No exportado por el paquete. *e.g.*, `bioinfo::*gc-tensor*`

Universidad Veracruzana

Otras funciones sobre paquetes

- La variable especial *package* contiene el paquete actual:

```
1  > *package*
2  #<The COMMON-LISP-USER package, 23/64 internal, 0/4 external>
3  > (in-package "bioinfo")
4  #<The bioinfo package, 0/16 internal, 0/16 external>
5  BIOINFO> *package*
6  #<The bioinfo package, 0/16 internal, 0/16 external>
```

- La directiva :export exporta símbolos.
- Los paquete pueden organizarse por capas , usando use-package.
- Vean el capítulo 21 del libro de Seibel [6].

Universidad Veracruzana

Código Genético Universal

- Recuerden que estamos en el paquete `:bioinfo`

```
(defvar *gc-tensor*
  '(((t t t) phe) ((t c t) ser) ((t a t) tyr) ((t g t) cys)
    ((t t c) phe) ((t c c) ser) ((t a c) tyr) ((t g c) cys)
    ((t t a) leu) ((t c a) ser) ((t a a) ter) ((t g a) ter)
    ((t t g) leu) ((t c g) ser) ((t a g) ter) ((t g g) trp)
    ((c t t) leu) ((c c t) pro) ((c a t) his) ((c g t) arg)
    ((c t c) leu) ((c c c) pro) ((c a c) his) ((c g c) arg)
    ((c t a) leu) ((c c a) pro) ((c a a) gln) ((c g a) arg)
    ((c t g) leu) ((c c g) pro) ((c a g) gln) ((c g g) arg)
    ((a t t) ile) ((a c t) thr) ((a a t) asn) ((a g t) ser)
    ((a t c) ile) ((a c c) thr) ((a a c) asn) ((a g c) ser)
    ((a t a) ile) ((a c a) thr) ((a a a) lys) ((a g a) arg)
    ((a t g) met) ((a c g) thr) ((a a g) lys) ((a g g) arg)
    ((g t t) val) ((g c t) ala) ((g a t) asp) ((g g t) gly)
    ((g t c) val) ((g c c) ala) ((g a c) asp) ((g g c) gly)
    ((g t a) val) ((g c a) ala) ((g a a) glu) ((g g a) gly)
    ((g t g) val) ((g c g) ala) ((g a g) glu) ((g g g) gly))
  "universal genetic code - tensor A")
```

Universidad Veracruzana

Probando el paquete

- Si forzamos el asunto con `bioinfo::*gc-tensor*` obtendremos la matriz, pero con todos sus símbolos calificados explícitamente:

```
CL-USER> bioinfo::*gc-tensor*
(((T T T) BIOINFO::PHE) ((T BIOINFO::C T) BIOINFO::SER) ...
```

- Si nos movemos a `:bioinfo`, el acceso a los elementos de la matriz se hace normalmente:

```
CL-USER> (in-package :bioinfo)
#<The BIOINFO package, 25/64 internal, 0/16 external>
BIOINFO> *gc-tensor*
(((T T T) PHE) ((T C T) SER) ...
```

- Observen el curioso caso de `T`. ¿Por qué se comporta así?

Universidad Veracruzana

Matriz PAM250 I

- Las matrices de similitud serán implementadas como arreglos, no como listas.

```
(defvar *pam250*
;;; Dayhoff PAM250 (percent accepted mutations)
;;; as reported by Mac Donaill, Molecular Simulation 30(5) p.269
  (make-array
   '(20 20)
   :initial-contents
   '(( 2 -2  0  0 -2  0  0  1 -1 -1 -2 -1 -1 -4  1  1  1 -6 -3  0)
     (-2  6  0 -1 -4  1 -1 -3  2 -2 -3  3  0 -4  0  0 -1  2 -4 -2)
     ( 0  0  2  2 -4  1  1  0  2 -2 -3  1 -2 -4 -1  1  0 -4 -2 -2)
     ( 0 -1  2  4 -5  2  3  1  1 -2 -4  0 -3 -6 -1  0  0 -7 -4 -2)
     (-2 -4 -4 -5 12 -5 -5 -3 -3 -2 -6 -5 -5 -4 -3  0 -2 -8  0 -2)
     ... ))))
```

- Esto es para hacer más eficiente el acceso a sus elementos.

Universidad Veracruzana

Opciones como variable global

- Se puede definir una variable global para las matrices que podemos usar y la que usamos por defecto:

```
(defvar *options*
  '(*pam250* *codegen* *robersy-grau* *miyazawa* *prlic* *blosum62*)
  "Matrices")

(defvar *option* '*pam250* "Default option")
```

- Y aprovechando, otra para las posibles mutaciones:

```
(defvar *nucleotide-mappings*
  '((T C) (T A) (T G)
    (C A) (C G) (C T)
    (A G) (A T) (A C)
    (G T) (G C) (G A))
  "The 12 possible nucleotide mappings")
```

Universidad Veracruzana

Indexación de los aminoácidos

- La función `index` regresa el índice de un aminoácido en una matriz de similitud (renglón o columna):

```
(defun index (aa)
  "Get the index of aminoacid aa in a matrix"
  (case aa
    ((A ALA) 0)
    ((R ARG) 1)
    ((N ASN) 2)
    ((D ASP) 3)
    ...)
```

- Observen el uso de `case`. De manera que:

```
BIOINFO> (index 'arg)
1
BIOINFO> (index 'r)
1
```

Universidad Veracruzana

Distancias entre dos aminoácidos I

- La función `get-dist` computa la distancia entre dos aminoácidos dada una matriz de similitud (por defecto *PAM*250).
- Ejemplo.

```
1 BIOINFO> (get-dist 'val 'arg)
2 -2
3 BIOINFO> (get-dist 'val 'arg *blosum62*)
4 -3
```

- Su definición hace uso de la palabra clave `&optional` para indicar que la matriz es un parámetro opcional y que su valor por defecto es `*option*`:

```
1 (defun get-dist (aai aaj &optional (matrix *option*))
2   ... )
```

Universidad Veracruzana

Distancias entre dos aminoácidos II

- El cuerpo de la función cubre tres casos: Cuando ambos aminoácidos son terminadores, cuando uno de ellos lo es; y cuando ninguno de los dos lo es.
- Cada matriz regresa un valor específico para los casos especiales que involucran un terminador.
- El caso normal simplemente regresa el elemento (aai, aaj) en la matriz correspondiente, computado con `aref`.

Universidad Veracruzana

Distancias entre dos aminoácidos III

```
(cond ((and (equal aai 'ter) ;; ter to ter
            (equal aaj 'ter))
       (case matrix
         (*pam250* 1)
         (*codegen* 0)
         (*miyazawa* 0)
         (*prlic* 0)
         (*blosum62* 0)
         (*robersy-grau* (aref (eval matrix)
                               (index aai)
                               (index aaj)))
         (otherwise (error "matrix not defined"))))
      ((or (equal aai 'ter) ;;; aminoacid to ter
           (equal aaj 'ter))
       (case matrix
         (*pam250* -8)
         (*codegen* 6)
         (*miyazawa* -1.01)
         (*prlic* 0)
         (*blosum62* 0)
         (*robersy-grau* (aref (eval matrix)
```

Universidad Veracruzana

Distancias entre dos aminoácidos IV

```
22                                  (index aai)
23                                  (index aaj)))
24              (otherwise (error "matrix not defined"))))
25          (t (aref (eval matrix) (index aai) (index aaj)))))
```

Universidad Veracruzana

Nucleotidos a aminoácidos y viceversa

- Necesitamos dos funciones de conversión entre notaciones:

```
(defun aa-to-nnn (aa)
  "Gets the nucleotides of the aminoacid"
  (remove-if #'(lambda(aminoacid)
                 (not (equal (cadr aminoacid) aa)))
             *gc-tensor*))

(defun nnn-to-aa (nnn)
  "Gets the aminoacid from the nucleotides"
  (car (member-if #'(lambda(aminoacid)
                      (equal (car aminoacid) nnn))
                  *gc-tensor*)))
```

- Ejemplo.

```
BIOINFO> (aa-to-nnn 'val)
(((G T T) VAL) ((G T C) VAL) ((G T A) VAL) ((G T G) VAL))
BIOINFO> (nnn-to-aa '(g t t))
((G T T) VAL)
```

Universidad Veracruzana

Imprimir código genético universal

- Función para imprimir las matrices en 4 columnas:

```
(defun print-tensor (tensor)
  (cond ((null tensor) t)
        (t (progn
             (format t "~a ~a ~a ~a ~%"
               (car tensor) (cadr tensor)
               (caddr tensor) (cadddr tensor))
             (print-tensor (cddddr tensor))))))
```

Universidad Veracruzana

O usando solo format

- Como en la tarea:

```
BIOINFO> (format t "~{~a ~a ~a ~a ~%~}" *gc-tensor*)
((T T T) PHE) ((T C T) SER) ((T A T) TYR) ((T G T) CYS)
((T T C) PHE) ((T C C) SER) ((T A C) TYR) ((T G C) CYS)
((T T A) LEU) ((T C A) SER) ((T A A) TER) ((T G A) TER)
((T T G) LEU) ((T C G) SER) ((T A G) TER) ((T G G) TRP)
((C T T) LEU) ((C C T) PRO) ((C A T) HIS) ((C G T) ARG)
((C T C) LEU) ((C C C) PRO) ((C A C) HIS) ((C G C) ARG)
((C T A) LEU) ((C C A) PRO) ((C A A) GLN) ((C G A) ARG)
...
NIL
```

Universidad Veracruzana

Distancia de aminoácidos mutados I

- Una de las operaciones básicas consiste en mutar una de las bases (`pos`) de acuerdo a un mapeo (`mapping`) en el código genético universal.
- Esto lo hace la función `apply-mapping`:

```
1  (defun apply-mapping (pos mapping)
2    "Applies a mapping in the position pos of *gc-tensor*"s"
3    (mapcar #'(lambda(mmm)(cons mmm (cdr (nnn-to-aa mmm))))
4      (mapcar #'(lambda(nnn)(change pos (car nnn) mapping))
5              *gc-tensor*)))
```

- Observen el uso del doble `mapcar` para mutar el codón y obtener el nombre del aminoácido del codón ya mutado. En ese orden.
- En este caso, `mapcar` resulta más natural que `loop` o una solución recursiva.

Universidad Veracruzana

Distancia de aminoácidos mutados II

- La función auxiliar `change` es la que modifica la base de acuerdo al mapeo:

```
1  (defun change (pos nnn mapping)
2    "Returns the codon nnn modified in position pos accordingly to mapping"
3    (let* ((nnnaux (copy-list nnn))
4           (from-n (first mapping))
5           (to-n (second mapping))
6           (elem (nth (- pos 1) nnnaux)))
7      (when (eql elem from-n)
8        (setf (nth (- pos 1) nnnaux) to-n))
9    nnnaux))
```

- Observen el uso de `copy-list` para evitar que el cambio afecte también al codón original (que `nnn` y `nnnaux` denoten el mismo objeto).

Universidad Veracruzana

Distancia de aminoácidos mutados III

- La función `apply-dist` computa las distancias entre los elementos de un código genético (normalmente mutado) y el código genético universal, de acuerdo a una matriz dada (por efecto *PAM*250):

```
1  (defun apply-dist (tensor &optional (matrix *option*))
2    "Returns the distance between *gc-tensor* and tensor accordingly
3     to *option*"
4    (mapcar #'(lambda(i j)
5                (list (car i) (get-dist (cadr i) (cadr j) matrix)))
6            *gc-tensor* tensor))
```

- Observen el uso de `cadr` para acceder al nombre del aminoácido a comparar.
- ¿Cómo es la salida de esta función?

Universidad Veracruzana

Distancia de aminoácidos mutados IV

- Ejemplo.

```
BIOINFO> (apply-dist (apply-mapping 1 '(C-G)))
(((T T T) 9) ((T C T) 2) ((T A T) 10) ((T G T) 12) ((T T C) 9) ((T C C) 2)
 ((T A C) 10) ((T G C) 12) ((T T A) 6) ((T C A) 2) ((T A A) 1) ((T G A) 1)
 ((T T G) 6) ((T C G) 2) ((T A G) 1) ((T G G) 17) ((C T T) 6) ((C C T) 6)
 ((C A T) 6) ((C G T) 6) ((C T C) 6) ((C C C) 6) ((C A C) 6) ((C G C) 6)
 ((C T A) 6) ((C C A) 6) ((C A A) 4) ((C G A) 6) ((C T G) 6) ((C C G) 6)
  ...
```

- ¿Porqué?

Universidad Veracruzana

S-nnn I

- Con estos elementos podemos computar la significancia de las tres bases con la función `S-nnn`:

```
(defun S-nnn (pos &optional (matrix *option*))
  ;;; computes average S for a given position in the codon
  (/ (float (apply #'+
                   (mapcar #'(lambda(ldists)
                               (S-aux ldists matrix))
                           (mapcar #'(lambda (lmaps)
                                       (apply-mapping pos lmaps))
                                   *nucleotide-mappings*))))
     12.0))
```

- Observen el uso de `mapcar` para aplicar los doce mapeos posibles y calcular las doce distancias.
- Observen el uso de `float` para forzar el tipo de dato resultante.

Universidad Veracruzana

S-nnn II

- La función auxiliar `S-aux` regresa la sumatoria de todas las distancias que computa `apply-dist`:

```
1  (defun S-aux (tensor &optional (matrix *option*))
2    ;;; Computes S = Sum Cijk
3    ;;; The overall significance of a selected mapping
4    (apply #'+ (mapcar #'(lambda(l) (cadr l))
5                       (apply-dist tensor matrix))))
```

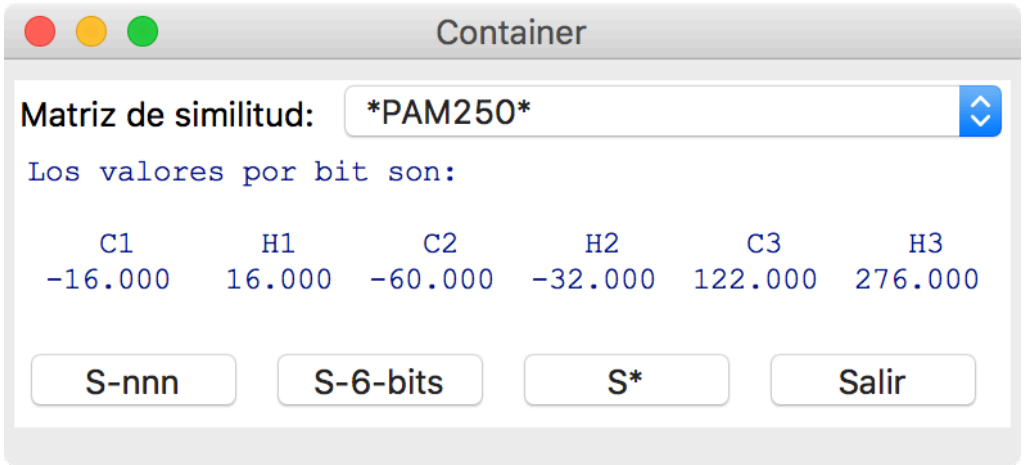
- De manera que:

```
1  BIOINFO> (s-nnn 1)
2  235.91667
3  BIOINFO> (s-nnn 2)
4  220.08333
5  BIOINFO> (s-nnn 3)
6  281.08334
```

Universidad Veracruzana

¿Y el paquete :ltk?

- La función `gui` despliega la siguiente interfaz gráfica:

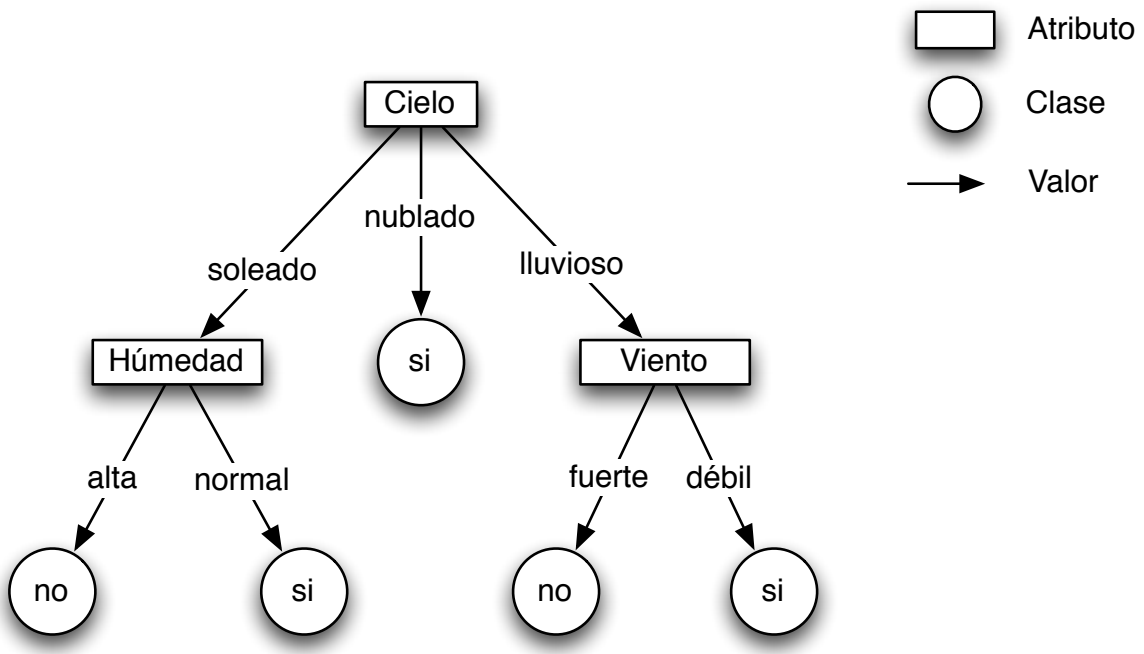


- Nota. Un bug en MacOS 15 puede romper la comunicación con Tk. En los otros sistemas operativos no deben tener problema.
- Los casos para 6 bits y normalizados (S*) son análogos al caso revisado.

Universidad Veracruzana

Arboles de Decisión

- Los árboles de decisión representan hipótesis sobre una clase, como una conjunción de disyunciones de atributos.
- Recordando un buen día para jugar tenis [4]:



Universidad Veracruzana

Ejemplos de entrenamiento

- Representación proposicional (atributo-valor)

Día	Cielo	Temperatura	Humedad	Viento	Jugar-tenis?
1	soleado	calor	alta	débil	no
2	soleado	calor	alta	fuerte	no
3	nublado	calor	alta	débil	si
4	lluvia	templado	alta	débil	si
5	lluvia	frío	normal	débil	si
6	lluvia	frío	normal	fuerte	no
7	nublado	frío	normal	fuerte	si
8	soleado	templado	alta	débil	no
9	soleado	frío	normal	débil	si
10	lluvia	templado	normal	débil	si
11	soleado	templado	normal	fuerte	si
12	nublado	templado	alta	fuerte	si
13	nublado	calor	normal	débil	si
14	lluvia	templado	alta	fuerte	no

Universidad Veracruzana

Arboles de Decisión

- Podemos representarlos como una lista de listas:

```
CL-USER> (setq arbol '(cielo
                       (soleado (humedad
                                 (normal si)
                                 (alta no)))
                       (nublado si)
                       (lluvia (viento
                                (fuerte no)
                                (debil si)))))

(CIELO (SOLEADO (HUMEDAD (NORMAL SI) (ALTA NO)))
(NUBLADO SI) (LLUVIA (VIENTO (FUERTE NO) (DEBIL SI))))
```

- La raíz del árbol es el primer elemento de la lista. El resto de la lista son los sub-árboles de la raíz y su rama correspondiente.

Universidad Veracruzana

Funciones de acceso al árbol

- Podemos explotar esa estructura para definir algunas funciones básicas de acceso y validación:

```
1  (defun root (tree)
2    "Returns the root of the decision tree"
3    (car tree))
4
5  (defun sub-tree (tree value)
6    "Returns the sub-tree under the branch labeled as value"
7    (second (assoc value (cdr tree))))
8
9  (defun leaf-p (tree)
10   "Verifies if tree is a leaf"
11   (atom tree))
```

Universidad Veracruzana

Ejemplos

- Dado nuestro árbol inicial tenemos su raíz es:

```
CL-USER> (root arbol)
CIELO
```

- O el subárbol por la rama `soleado`:

```
CL-USER> (sub-tree arbol 'soleado)
(HUMEDAD (NORMAL SI) (ALTA NO))
```

- O podemos verificar si el sub-árbol de `nublado` es una hoja:

```
CL-USER> (leaf-p (sub-tree arbol 'nublado))
T
```

Universidad Veracruzana

¿Cómo funciona assoc?

- Su primer argumento es un elemento a buscar en una lista de listas (el segundo argumento).
- Regresa el primer `cons` del segundo argumento cuyo `car` es igual a su primer argumento.

```
1  CL-USER> (assoc 'uno '((uno 1) (dos 2) (tres 3)))
2  (UNO 1)
3  CL-USER> (assoc 'dos '((uno 1) (dos 2) (tres 3)))
4  (DOS 2)
5  CL-USER> (assoc 'lluvia (cdr arbol))
6  (LLUVIA (VIENTO (FUERTE NO) (DEBIL SI)))
```

- Hay variantes `assoc-if` y `assoc-if-not`.

Universidad Veracruzana

Imprimiendo un árbol de decisión

- Observen el uso de `loop` y `terpri`:

```
1  (defun print-tree (tree &optional (depth 0))
2    (mytab depth)
3    (format t "~A~%" (first tree))
4    (loop for subtree in (cdr tree) do
5          (mytab (+ depth 1))
6          (format t "- ~A" (first subtree))
7          (if (atom (second subtree))
8              (format t " -> ~A~%" (second subtree))
9              (progn
10               (terpri)
11               (print-tree (second subtree) (+ depth 5))))))
12
13 (defun mytab (n)
14   (loop for i from 1 to n do (format t " ")))
```

Universidad Veracruzana

Ejemplo

- Imprimiendo nuestro árbol:

```
CL-USER> (print-tree arbol)
CIELO
 - SOLEADO
     HUMEDAD
      - NORMAL -> SI
      - ALTA -> NO
 - NUBLADO -> SI
 - LLUVIA
     VIENTO
      - FUERTE -> NO
      - DEBIL -> SI
NIL
```

Universidad Veracruzana

Ejemplos de entrenamiento

- Aunque los ejemplos también pueden representarse como una lista de listas, esto no es buena idea.
- Necesitamos una representación que nos permita relacionar el valor de un atributo en un ejemplo dado.
- Ejemplo: Supongan que `*examples*` es mi repositorio de ejemplos. Quiero computar el valor de `cielo` en el primer ejemplo del repositorio:

```
1 CL-USER> (get-value 'cielo (car *examples*))
2 LLUVIA
```

Universidad Veracruzana

Ejemplos, atributos y valores

- Las bases para la indexación son las siguientes:

```
(defun put-value (attr inst val)
  (setf (get inst attr) val))

(defun get-value (attr inst)
  (get inst attr))
```

- Ejemplo:

```
CL-USER> (put-value 'nombre 'ej1 'alejandro)
ALEJANDRO
CL-USER> (get-value 'nombre 'ej1)
ALEJANDRO
CL-USER> (setq *ejemplos* nil)
NIL
CL-USER> (push 'ej1 *ejemplos*)
(EJ1)
CL-USER> (get-value 'nombre (car *ejemplos*))
ALEJANDRO
```

Universidad Veracruzana

Archivos de ejemplos: ARFF y CSV

- Un archivo `tenis.arff`:

```
@RELATION jugar-tenis

@ATTRIBUTE cielo {soleado,nublado,lluvia}
@ATTRIBUTE temperatura {calor,templado,frio}
@ATTRIBUTE humedad {alta,normal}
@ATTRIBUTE viento {debil,fuerte}
@ATTRIBUTE jugar-tenis {si,no}

@DATA

soleado, calor, alta, debil, no
soleado, calor, alta, fuerte, no
nublado, calor, alta, debil, si
lluvia, templado, alta, debil, si
...
```

Universidad Veracruzana

With a little help from your friends

- ASFD2 (*Another System Definition Facility 2*) es una herramienta para describir como están organizados los archivos fuente de un sistema Lisp.
- En particular sus dependencias.
- Quicklisp es una herramienta que nos permite instalar librerías que han sido definidas usando ASDF2:

`http://www.quicklisp.org`

Universidad Veracruzana

Instalando librerías: trivial-shell

- `trivial-shell` permite ejecutar comandos del shell de Unix.
- Su instalación con `quicklisp` es como sigue:

```
CL-USER> (ql:quickload "trivial-shell")
To load "trivial-shell":
  Install 1 Quicklisp release:
    trivial-shell
; Fetching #<QL-HTTP:URL
"http://beta.quicklisp.org/archive/trivial-shell/
2011-05-22/trivial-shell-20110522-http.tgz">
; 13.61KB
==================================================
13,937 bytes in 0.00 seconds (13610.35KB/sec)
; Loading "trivial-shell"
[package com.metabang.trivial-timeout].....
[package trivial-shell]..
("trivial-shell")
```

Universidad Veracruzana

Ejecutando comandos shell: grep

- De forma que podemos usar egrep para extraer líneas del archivo ARFF:

```
CL-USER> (trivial-shell:shell-command
           "egrep \"@ATTR\" tenis.arff")
"@ATTRIBUTE cielo {soleado,nublado,lluvia}
@ATTRIBUTE temperatura {calor,templado,frio}
@ATTRIBUTE humedad {alta,normal}
@ATTRIBUTE viento {debil,fuerte}
@ATTRIBUTE jugar-tenis {si,no}
"
NIL
0
```

- Observen que obtuvimos una sola cadena de texto con la respuesta deseada.
- Habrá que partirla!

Universidad Veracruzana

Partiendo cadenas: split-sequence

- Podemos llamar a `split-sequence` para partir cadenas de texto:

```
CL-USER> (ql:quickload "split-sequence")
To load "split-sequence":
  Load 1 ASDF system:
    split-sequence
; Loading "split-sequence"

("split-sequence")
CL-USER> (split-sequence:split-sequence #\, "hola, que, tal")
("hola" " que" " tal")
14
```

- El símbolo `\#,` es un carácter. Indica que partiremos la cadena en las comas.

Universidad Veracruzana

Ensamblando cajitas

- Podemos usar estas facilidades para leer las líneas del archivo ARFF:

```
CL-USER> (split-sequence:split-sequence #\Newline
                                        (trivial-shell:shell-command
                                         "egrep \"@ATTR\" tenis.arff"))
("@ATTRIBUTE cielo {soleado,nublado,lluvia}"
 "@ATTRIBUTE temperatura {calor,templado,frio}"
 "@ATTRIBUTE humedad {alta,normal}" "@ATTRIBUTE viento {debil,fuerte}"
 "@ATTRIBUTE jugar-tenis {si,no}" "")
184
```

- Ahora tenemos una cadena por cada línea de texto.
- Observen la composición funcional.

Universidad Veracruzana

Variables globales

- Definamos variables globales para nuestro ambiente de trabajo:

```
1 (defvar *attributes* nil "The attributes of the problem")
2 (defvar *domains* nil "The domain of the attributes")
3 (defvar *target* nil "The target concept")
4 (defvar *examples* nil "The training set")
5 (defvar *trace* nil "Trace the computations")
```

- Recuerden adornarlas con orejas.

Universidad Veracruzana

Datos crudos

- Esta función lee los datos del conjunto de entrenamiento (líneas que no comienzan con salto de línea o arroba o porciento):

```
1  (defun get-data (arff)
2    (mapcar #'(lambda(x)
3                (mapcar #'read-from-string
4                        (split-sequence:split-sequence #\, x)))
5            (butlast
6             (split-sequence:split-sequence
7              #\Newline
8              (trivial-shell:shell-command
9               (concatenate 'string
10                          "egrep -v \"^$|[@%]\" "
11                          arff))))))
```

- Observen la expresión regular en la línea 10.
- ¿Para qué usamos `butlast` en la línea 5?

Universidad Veracruzana

Ejemplo

- Leyendo los ejemplos de entrenamiento:

```
CL-USER> (get-data "tenis.arff")
((SOLEADO CALOR ALTA DEBIL NO) (SOLEADO CALOR ALTA FUERTE NO) (NUBLADO
CALOR ALTA DEBIL SI) (LLUVIA TEMPLADO ALTA DEBIL SI) (LLUVIA FRIO
NORMAL DEBIL SI) (LLUVIA FRIO NORMAL FUERTE NO) (NUBLADO FRIO NORMAL
FUERTE SI) (SOLEADO TEMPLADO ALTA DEBIL NO) (SOLEADO FRIO NORMAL DEBIL
SI) (LLUVIA TEMPLADO NORMAL DEBIL SI) (SOLEADO TEMPLADO NORMAL FUERTE
SI) (NUBLADO TEMPLADO ALTA FUERTE SI) (NUBLADO CALOR NORMAL DEBIL SI)
(LLUVIA TEMPLADO ALTA FUERTE NO))
```

- Observen que ya no tenemos cadenas de texto, sólo símbolos ¿Cómo conseguimos esto?

Universidad Veracruzana

Clase a predecir

- Recuperando la clase a predecir (*target*):

```
(defun get-target (arff)
  (read-from-string
   (cadr (split-sequence:split-sequence
          #\Space
          (car
           (butlast
            (split-sequence:split-sequence
             #\Newline
             (trivial-shell:shell-command
              (concatenate 'string
                           "egrep \"@rel|@REL\" "
                           arff)))))))))
```

```
CL-USER> (get-target "tenis.arff")
JUGAR-TENIS
11
```

Universidad Veracruzana

Dominios

- Asumiendo que `*attributes*` es la lista de atributos en el conjunto de entrenamiento.
- Y que `*domains*` es la lista de dominios, en el mismo orden.
- Podemos computar el dominio de un atributo con:

```
1  (defun get-domain (attribute)
2    (nth (position attribute *attributes*)
3         *domains*))
```

- Ejemplo.

```
1  CL-USER> (get-domain 'cielo)
2  (SOLEADO NUBLADO LLUVIA)
```

Universidad Veracruzana

Todo junto

- Leyendo un ARFF:

```
1  (defun id3-load (arff)
2    (let ((data (get-data arff))
3          (attribs-doms (get-attribs-doms arff)))
4      (setf *gensym-counter* 1)
5      (setf *attributes* (mapcar #'car attribs-doms))
6      (setf *domains* (mapcar #'cadr attribs-doms))
7      (setf *target* (get-target arff))
8      (loop for d in data do
9           (let ((ej (gensym "ej")))
10            (push ej *examples*)
11            (loop for attrib in *attributes*
12               as v in d do
13                 (put-value attrib ej v))))
14     (format t "Training set initialized")))
```

- Observen el doble `loop`: Sobre ejemplos y sobre atributos y valores de un ejemplo.

Universidad Veracruzana

Ejemplo

- Cargando los ejemplos de tenis:

```
CL-USER> (id3-load "tenis.arff")
Training set initialized
NIL
```

- Mis ejemplos de entrenamientos son:

```
CL-USER> *examples*
(#:|ej14| #:|ej13| #:|ej12| #:|ej11| #:|ej10| #:|ej9| #:|ej8| #:|ej7|
#:|ej6| #:|ej5| #:|ej4| #:|ej3| #:|ej2| #:|ej1|)
```

- Que usan los atributos:

```
CL-USER> *attributes*
(CIELO TEMPERATURA HUMEDAD VIENTO JUGAR-TENIS)
```

- Cuyos dominios son:

```
CL-USER> *domains*
((SOLEADO NUBLADO LLUVIA) (CALOR TEMPLADO FRIO) (ALTA NORMAL)
 (DEBIL FUERTE) (SI NO))
```

Universidad Veracruzana

Y el reseteo

- Limpiando las variables globales:

```
(defun id3-reset ()
  (setf *examples* nil
        *target* nil
        *attributes* nil
        *domains* nil
        *trace* nil
        *root* nil)
  (format t "The ID3 setting has been reset."))
```

- Es necesaria para poder trabajar con un nuevo conjunto de entrenamiento sin salir del sistema.

Universidad Veracruzana

Modularización del código

- Como el sistema se está haciendo más grande y complicado, sería conveniente factorizarlo en componentes, *e.g.*, implementarlo en distintos módulos.
- Para ello es necesario especificar las dependencias entre los componentes del sistema y su orden de compilación.
- Y definir un paquete para evitar conflictos de nombres con los componentes predefinidos en Lisp.

Universidad Veracruzana

Definición de un sistema ASDF: cl-id3.asd

```
1  (asdf:defsystem :cl-id3
2      :depends-on (:split-sequence)
3      :components ((:file "cl-id3-package")
4                   (:file "cl-id3-algorithm"
5                          :depends-on ("cl-id3-package"))
6                   (:file "cl-id3-load"
7                          :depends-on ("cl-id3-package"
8                                       "cl-id3-algorithm"))
9                   (:file "cl-id3-cross-validation"
10                         :depends-on ("cl-id3-package"
11                                      "cl-id3-algorithm"))
12                  (:file "cl-id3-classify"
13                         :depends-on ("cl-id3-package"
14                                      "cl-id3-algorithm"
15                                      "cl-id3-load"))))
```

Universidad Veracruzana

Cargando el sistema ASDF

1. Crear un directorio `common-lisp` en su `home`.
2. Copiar la carpeta del proyecto `cl-id3` a este directorio.
3. En SBCL ejecutar:

```
1 CL-USER> (asdf:load-system :cl-id3)
2 T
```

Universidad Veracruzana

Prueba

- Ahora podemos cargar un conjunto de entrenamiento:

```
CL-USER> (cl-id3:load-file "~/common-lisp/cl-id3/tenis.arff")
The ID3 setting has been reset.
Training set initialized after "~/common-lisp/cl-id3/tenis.arff".
```

- E inducir un árbol de decisión con él:

```
CL-USER> (cl-id3:print-tree (cl-id3:induce))
CIELO
 - SOLEADO
     TEMPERATURA
      - NORMAL -> SI
      - ALTA -> NO
 - NUBLADO -> SI
 - LLUVIA
     VIENTO
      - FUERTE -> NO
      - DEBIL -> SI
```

Universidad Veracruzana

Definiendo un paquete: cl-id3-package

- Para definir el paquete `cl-id3` hacemos uso de `defpackage`:

```
1  (defpackage :cl-id3
2    (:use :cl :split-sequence)
3    (:export :load-file
4             :induce
5             :print-tree
6             :classify
7             :classify-new-instance
8             :cross-validation))
```

- Observen que ahora si estamos exportando varias funciones.
- Y que el sistema necesita que `split-sequence` esté instalado en nuestro Lisp (vía `quicklisp`).

Universidad Veracruzana

Leyendo archivos: cl-id3-load

- Este módulo se encarga de cargar los conjuntos de entrenamiento en el sistema.
- Para no usar `trivial-shell` leeremos los archivos con funciones predefindas en Lisp:

```
1  (in-package :cl-id3)
2
3  ;;; Auxiliar functions
4
5  (defun read-lines-from-file (file)
6    "It reads the FILE into a list of strings"
7    (remove-if (lambda (x) (equal x ""))
8               (with-open-file (in file)
9                 (loop for line = (read-line in nil 'end)
10                   until (eq line 'end) collect line))))
```

Universidad Veracruzana

Portabilidad: Bye, bye grep

```
1  (defun arff-get-data (lines)
2    "It extracts the value for *data* from the lines of a ARFF file"
3    (mapcar #'(lambda(x)
4                (mapcar #'read-from-string
5                        (split-sequence #\, x)))
6            (remove-if
7             (lambda (x) (string-equal "@" (subseq x 0 1)))
8             lines)))
9
10 (defun csv-get-data (lines)
11   "It extracts the value for *data* from the lines of a CSV file"
12   (mapcar #'(lambda(x)
13               (mapcar #'read-from-string
14                       (split-sequence #\, x))) (cdr lines)))
```

Universidad Veracruzana

Ejemplo:

- Ejecutando `arff-get-data` (más cajitas):

```
CL-ID3> (arff-get-data (read-lines-from-file "tenis.arff"))
((SOLEADO CALOR ALTA DEBIL NO) (SOLEADO CALOR ALTA FUERTE NO) (NUBLADO
CALOR ALTA DEBIL SI) (LLUVIA TEMPLADO ALTA DEBIL SI) (LLUVIA FRIO
NORMAL DEBIL SI) (LLUVIA FRIO NORMAL FUERTE NO) (NUBLADO FRIO NORMAL
FUERTE SI) (SOLEADO TEMPLADO ALTA DEBIL NO) (SOLEADO FRIO NORMAL DEBIL
SI) (LLUVIA TEMPLADO NORMAL DEBIL SI) (SOLEADO TEMPLADO NORMAL FUERTE
SI) (NUBLADO TEMPLADO ALTA FUERTE SI) (NUBLADO CALOR NORMAL DEBIL SI)
(LLUVIA TEMPLADO ALTA FUERTE NO))
```

Universidad Veracruzana

Cargando los archivos I

- La función `load-file` carga los conjuntos de entrenamiento en el sistema.
- Recibe como parámetro la ruta al archivo que queremos cargar.
- Incluye como función local `get-examples` que vimos antes:

```
1  (defun load-file (file)
2    "It initializes the learning setting from FILE"
3    (labels ((get-examples (data)
4               (loop for d in data do
5                     (let ((ej (gensym "ej")))
6                       (setf *examples* (cons ej *examples*))
7                       (loop for attrib in *attributes*
8                          as v in d do
9                          (put-value attrib ej v))))))
```

Universidad Veracruzana

Cargando los archivos II

- Primero prueba si el archivo existe y extrae su nombre para poder leer las líneas del archivo. Luego hace un `reset` para comenzar a trabajar en él.

```
1  (if (probe-file file)
2      (let ((file-ext (car (last (split-sequence #\. file))))
3            (file-lines (read-lines-from-file file)))
4            (reset)
```

Universidad Veracruzana

Cargando los archivos III

- Luego se inicializan las variables globales pertinentes dependiendo del tipo de archivo (csv o arff):

```
(cond
  ((equal file-ext "arff")
   (let ((attribs-doms (arff-get-attribs-doms file-lines)))
     (setf *attributes* (mapcar #'car attribs-doms))
     (setf *domains* (mapcar #'cadr attribs-doms))
     (setf *target* (arff-get-target file-lines))
     (setf *data* (arff-get-data file-lines))
     (get-examples *data*)
     (format t "Training setting initialized after ~s.~%" file)))
  ((equal file-ext "csv")
   (let ((attribs-doms (csv-get-attribs-doms file-lines)))
     (setf *attributes* (mapcar #'car attribs-doms))
     ...
```

Universidad Veracruzana

Cargando los archivos IV

- Finalmente, si los casos anteriores fallan, se generan los errores correspondientes:

```
1       (t (error "File's ~s extension can not be determined." file))))
2   (error "File ~s does not exist.~%" file))))
```

Universidad Veracruzana

Ejemplo

- Cargando nuestro conocido caso del tenis:

```
CL-ID3> (load-file "~/common-lisp/cl-id3/tenis.arff")
The ID3 setting has been reset.
Training setting initialized after "tenis.arff".
```

- Nuestras variables globales están inicializadas:

```
CL-ID3> *target*
JUGAR-TENIS
CL-ID3> *attributes*
(CIELO TEMPERATURA HUMEDAD VIENTO JUGAR-TENIS)
CL-ID3> *examples*
(#:|ej14| #:|ej13| #:|ej12| #:|ej11| #:|ej10| #:|ej9| #:|ej8|
#:|ej7| #:|ej6| #:|ej5| #:|ej4| #:|ej3| #:|ej2| #:|ej1|)
```

- Observen que estoy dentro del paquete `cl-id3`.
- Hay que dar la ruta completa al archivo de interés.

Universidad Veracruzana

Algoritmo principal I

- Considera los mismos casos que en Prolog. Si todos los ejemplos son de la misma clase regresa una hoja con la clase:

```
(defun id3 (examples attribs)
  "It induces a decision tree runing ID3 over EXAMPLES and ATTRIBS)"
  (let ((class-by-default (get-value *target* (car examples))))
    (cond
      ;; Stop criteria
      ((same-class-value-p *target*
                           class-by-default
                           examples) class-by-default)
```

- Si no tenemos más atributos regresa el valor más común de la clase de los ejemplos:

```
;; Failure
((null attribs) (target-most-common-value examples))
```

Universidad Veracruzana

Algoritmo principal II

- En cualquier otro caso, el árbol crece:

```
1  ;; Recursive call
2  (t (let* ((partition (best-partition attribs examples))
3            (node (first partition)))
4       (cons node
5             (loop for branch in (cdr partition) collect
6                   (list (first branch)
7                         (id3 (cdr branch)
8                              (remove node attribs)))))))))
```

- Observen la llamada recursiva a `id3` en el `loop` (línea 7).
- Observen el uso de `cons` para construir el árbol (línea 4).

Universidad Veracruzana

¿Todos somos de la misma clase?

- Criterio terminal:

```
1  (defun same-class-value-p (attrib value examples)
2    "Do all EXAMPLES have the same VALUE for a given ATTRIB ?"
3    (every #'(lambda(e)
4               (eq value
5                   (get-value attrib e)))
6           examples))
```

- Ejemplo:

```
1  CL-ID3> (same-class-value-p *target* 'si *examples*)
2  NIL
3  CL-ID3> (same-class-value-p *target* 'no *examples*)
4  NIL
```

Universidad Veracruzana

Valor mayoritario de clase

- La función no es complicada:

```
1  (defun target-most-common-value (examples)
2    "It gets the most common value for *target* in EXAMPLES"
3    (let ((domain (get-domain *target*))
4          (vals (mapcar #'(lambda(x) (get-value *target* x))
5                        examples)))
6      (caar (sort (loop for v in domain collect
7                        (list v (count v vals)))
8                  #'(lambda(x y) (>= (cadr x)
9                                     (cadr y)))))))
```

- Pero observen que `sort` necesita una función *ad-hoc* para comparar los pares valor-frecuencia (línea 8).
- Ejemplo.

```
1  CL-ID3> (target-most-common-value *examples*)
2  SI
```

Universidad Veracruzana

Partición I

- La siguiente función parte los ejemplos con base en los valores de `attrib`:

```
1  (defun get-partition (attrib examples)
2    "It gets the partition induced by ATTRIB in EXAMPLES"
3    (let (result vlist v)
4      (loop for e in examples do
5            (setq v (get-value attrib e))
6            (if (setq vlist (assoc v result))
7                ;;; value v existed, the example e is added
8                ;;; to the cdr of vlist
9                (rplacd vlist (cons e (cdr vlist)))
10               ;;; else a pair (v e) is added to result
11               (setq result (cons (list v e) result))))
12     (cons attrib result)))
```

- Observen esta variante de `let` que inicializa las tres variables como `NIL`. Observe el uso de `rplacd` que es destructiva.

Universidad Veracruzana

Partición II

- Ejemplo.

```
1  CL-ID3> (get-partition 'cielo *examples*)
2  (CIELO (SOLEADO #:|ej1| #:|ej2| #:|ej8| #:|ej9| #:|ej11|)
3         (NUBLADO #:|ej3| #:|ej7| #:|ej12| #:|ej13|)
4         (LLUVIA #:|ej4| #:|ej5| #:|ej6| #:|ej10| #:|ej14|))
```

Universidad Veracruzana

Entropía

```
(defun entropy (examples attrib)
  "It computes the entropy of EXAMPLES with respect to an ATTRIB"
  (let ((partition (get-partition attrib examples))
        (number-of-examples (length examples)))
    (apply #'+
           (mapcar #'(lambda(part)
                       (let* ((size-part (count-if #'atom
                                                   (cdr part)))
                              (proportion
                               (if (eq size-part 0) 0
                                   (/ size-part
                                      number-of-examples))))
                         (* -1.0 proportion (log proportion 2))))
                   (cdr partition)))))
```

```
CL-ID3> (entropy *examples* 'jugar-tenis)
0.9402859
```

Universidad Veracruzana

Ganancia de Información

```
(defun information-gain (examples attribute)
  "It computes information-gain for an ATTRIBUTE in EXAMPLES"
  (let ((parts (get-partition attribute examples))
        (no-examples (count-if #'atom examples)))
    (- (entropy examples *target*)
       (apply #'+
              (mapcar
               #'(lambda(part)
                   (let* ((size-part (count-if #'atom
                                               (cdr part)))
                          (proportion (if (eq size-part 0) 0
                                          (/ size-part
                                             no-examples))))
                     (* proportion (entropy (cdr part) *target*))))
               (cdr parts))))))
```

```
CL-ID3> (information-gain *examples* 'cielo)
0.2467497
```

Universidad Veracruzana

Mejor partición I

- Esta función hace uso de `loop` para recolectar las ganancias de información de los atributos:

```
(defun best-partition (attributes examples)
  "It computes one of the best partitions induced by ATTRIBUTES over EXAMPLES"
  (let* ((info-gains
          (loop for attrib in attributes collect
               (let ((ig (information-gain examples attrib))
                     (p (get-partition attrib examples)))
                 (when *trace*
                   (format t "Partición inducida por el atributo ~s:~%~s~%"
                           attrib p)
                   (format t "Ganancia de información: ~s~%"
                           ig))
                 (list ig p))))
         (best (cadar (sort info-gains #'(lambda(x y) (> (car x) (car y)))))))
    (when *trace* (format t "Best partition: ~s~%------------~%" best))
    best))
```

- Observen el uso de `*trace*` para controlar la cantidad de salida en terminal.

Universidad Veracruzana

Mejor partición II

- Ejemplo.

```
CL-ID3> (best-partition (remove *target* *attributes*)
                        *examples*)
(CIELO (SOLEADO #:|ej1| #:|ej2| #:|ej8| #:|ej9| #:|ej11|)
       (NUBLADO #:|ej3| #:|ej7| #:|ej12| #:|ej13|)
       (LLUVIA #:|ej4| #:|ej5| #:|ej6| #:|ej10| #:|ej14|))
```

Universidad Veracruzana

Induciendo un árbol

```
1  (defun induce (&optional (examples *examples*))
2    "It induces the decision tree using learning setting"
3    (when (not (member *target* *attributes*))
4      (error "The target is defined incorrectly: Maybe Weka modified your ARFF"))
5    (id3 examples (remove *target* *attributes*)))
```

```
1  CL-ID3> (induce)
2  (CIELO (SOLEADO (HUMEDAD (NORMAL SI) (ALTA NO)))
3  (NUBLADO SI) (LLUVIA (VIENTO (FUERTE NO) (DEBIL SI))))
```

Universidad Veracruzana

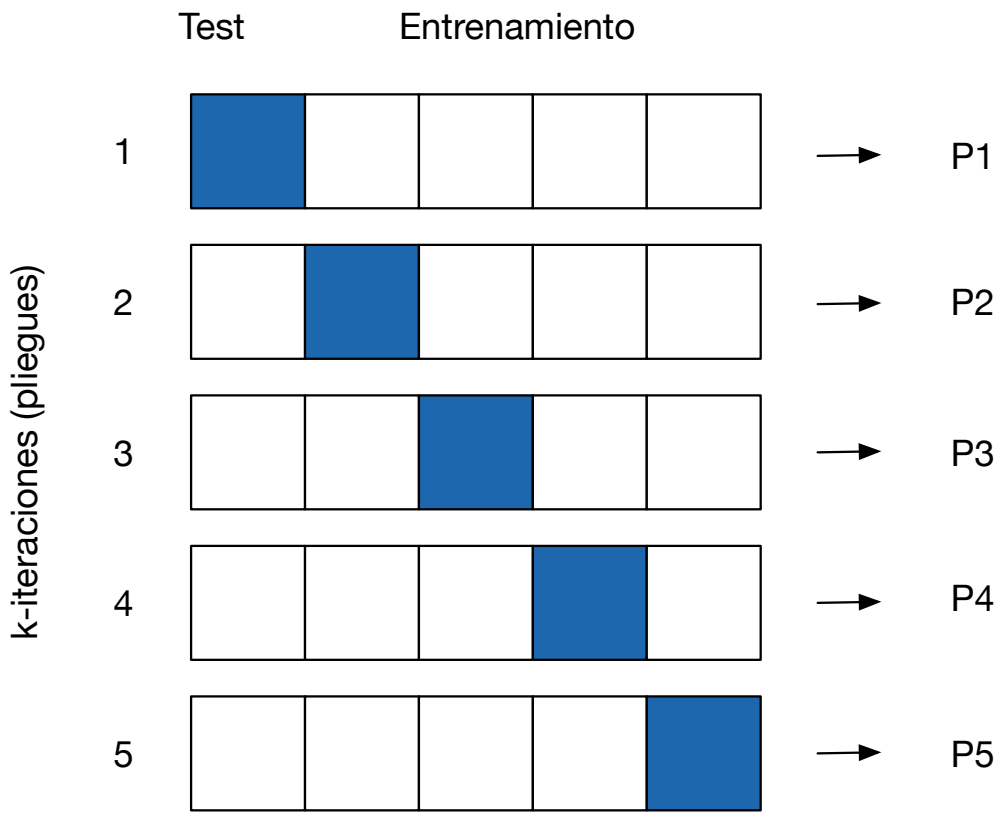
Y con print-tree

```
CL-ID3> (print-tree (induce))
CIELO
 - SOLEADO
     HUMEDAD
      - NORMAL -> SI
      - ALTA -> NO
 - NUBLADO -> SI
 - LLUVIA
     VIENTO
      - FUERTE -> NO
      - DEBIL -> SI
NIL
```

Universidad Veracruzana

Gráficamente

- Precisión promedio, para $k = 5$: $\frac{1}{5}\sum_{i=1}^{5} P_i$



Universidad Veracruzana

Funciones auxiliares

```
1  (defun square (n)
2    (* n n))
3
4  (defun mean (data)
5    (/ (reduce #'+ data) (length data)))
6
7  (defun std-dev (data)
8    (let ((data-mean (mean data)))
9      (sqrt (/ (reduce #'+ (mapcar
10                            #'(lambda (x)
11                                (square (- data-mean x)))
12                            data))
13               (length data)))))
```

Universidad Veracruzana

Cross-validation

```
1  (defun cross-validation (k)
2    "Cross-validations with k sets of size l - (/ l k)"
3    (let* ((long (length *examples*))
4           (n-training (- long (floor long k)))
5           (n-testing (- long n-training))
6           (results (loop repeat k
7                          collect
8                          (let* ((trainning-data (folding n-training long))
9                                 (test-data (set-difference *examples* trainnir
10                                (tree (induce trainning-data)))
11                           (/ (report tree test-data) n-testing)))))
12     ;; Estadísticas sobre todos los folds
13     (format t "~% Precision promedio: ~S~% Desviacion std: ~S~%"
14             (* 1.0 (mean results))
15             (std-dev results))))
16
```

Universidad Veracruzana

Folding

```
1  (defun folding (n size)
2    (let ((buffer nil))
3      (loop repeat n
4         collect
5           (nth (let ((r (random size)))
6                  (progn
7                    (while (member r buffer)
8                      (setf r (random size)))
9                    (push r buffer)
10                   r)) *examples*))))
```

Universidad Veracruzana

Prueba I

```
CL-USER> (cl-id3:load-file "~/common-lisp/cl-id3/tenis.csv")
The ID3 setting has been reset.
Training set initialized after "~/common-lisp/cl-id3/tenis.csv".
NIL
CL-USER> (cl-id3:cross-validation 2)
CIELO
 - NUBLADO -> SI
 - SOLEADO -> NO
 - LLUVIA
     VIENTO
      - FUERTE -> NO
      - DEBIL -> SI

Ejemplos bien clasificados: 5
Ejemplos mal clasificados: 2

CIELO
 - SOLEADO -> NO
 - LLUVIA
     VIENTO
```

Universidad Veracruzana

Prueba II

```
21          - FUERTE -> NO
22          - DEBIL -> SI
23      - NUBLADO -> SI
24
25  Ejemplos bien clasificados: 5
26  Ejemplos mal clasificados: 2
27
28
29      Precision promedio: 0.71428573
30      Desviacion std: 0.0
```

Universidad Veracruzana

Amazon, Reddit & Co.

- La idea es usar las preferencias de los usuarios, para dar recomendaciones a otros usuarios.
- Ejemplo: Amazon, Netflix, iTunes Store, Reddit, etc.

- Seguiremos la presentación de Seagaran [5].

Universidad Veracruzana

Filtrado Colaborativo

- Se trata de encontrar usuarios con gustos afines y usar sus preferencias para emitir recomendaciones.
- El término fue acuñado en Xerox PARC, ¿Donde más?, en 1992 por Goldberg et al. [1].

Universidad Veracruzana

Preferencias

- Asumamos 7 críticos para 5 películas.
- La escala de calificación va de 0 a 5.

Película	Rose	Seymour	Phillips	Puig	LaSalle	Mathews	Toby
Lady in Water	2.5	3.0	2.5	-	3.0	3.0	-
Snakes on Plane	3.5	3.5	3.0	3.5	4.0	4.0	4.5
Just my Luck	3.0	1.5	-	3.0	2.0	-	-
Superman returns	3.5	5.0	3.5	4.0	3.0	5.0	4.0
The Night Listener	3.0	3.0	4.0	4.5	3.0	3.0	-
You, Me and Dupree	2.5	3.5	-	2.5	2.0	3.5	3.5

- Observen que no todos los críticos revisan todas la películas.

Universidad Veracruzana

Representando preferencias

- Usaremos listas de propiedades:

```
(defvar *critics*
  '((:critic "Lisa Rose"
     :critics (("Lady in the Water" 2.5)
               ("Snakes on a Plane" 3.5)
               ("Just My Luck" 3.0)
               ("Superman Returns" 3.5)
               ("You, Me and Dupree" 2.5)
               ("The Night Listener" 3.0)))
    (:critic "Gene Seymour"
     :critics (("Lady in the Water" 3.0)
               ("Snakes on a Plane" 3.5)
               ("Just My Luck" 1.5)
               ("Superman Returns" 5.0)
               ("The Night Listener" 3.0)
               ("You, Me and Dupree" 3.5)))
...)
```

Universidad Veracruzana

Funciones de acceso

```
1  (defun get-critic (critic)
2    (find critic *critics*
3          :test #'(lambda(name reg)
4                    (string= name (getf reg :critic)))))
5
6  (defun get-film-rating (critic film)
7    (cadr (find film (getf (get-critic critic) :critics)
8                :test #'(lambda(film reg)
9                          (string= film (car reg))))))
10
11 (defun get-films-rated-by (person)
12   (mapcar #'car (fourth (get-critic person))))
```

Universidad Veracruzana

Ejemplos de uso

```
CL-USER> (get-critic "Toby")
(:CRITIC "Toby" :CRITICS (("Snakes on a Plane" 4.5) ("Superman Returns" 4.0)
("You, Me and Dupree" 1.0)))
CL-USER> (get-film-rating "Toby" "Superman Returns")
4.0
CL-USER> (get-films-rated-by "Toby")
("Snakes on a Plane" "Superman Returns" "You, Me and Dupree")
```

Universidad Veracruzana

Productos en común

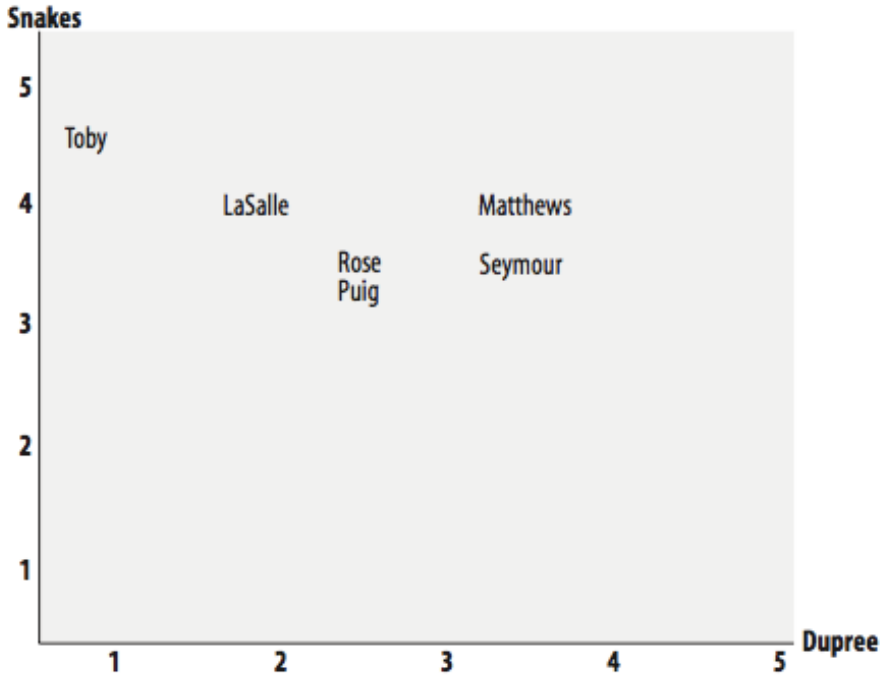
```lisp
(defun shared-items (critic1 critic2)
  (intersection (get-films-rated-by critic1)
                (get-films-rated-by critic2)
                :test #'string= ))
```

```
CL-USER> (shared-items "Toby" "Lisa Rose")
("Snakes on a Plane" "Superman Returns" "You, Me and Dupree")
```

Universidad Veracruzana

Distancia Euclidiana y Espacio de Preferencias

- $dist((x_1, y_1)(x_2, y_2)) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$



- Normalizaremos la distancia en $[0, \ldots, 1]$.

Universidad Veracruzana

Computando

```
(defun sim-euclidean (person1 person2)
  "1 means identical preferences, 0 the opposite"
  (let ((shared (shared-items person1 person2)))
    (if (null shared)
        0
        (/ 1
           (+ 1
              (sqrt (loop for film in shared sum
                          (sqr (- (get-film-rating person1 film)
                                  (get-film-rating person2 film)))))))))

(defun sqr (x) (* x x))
```

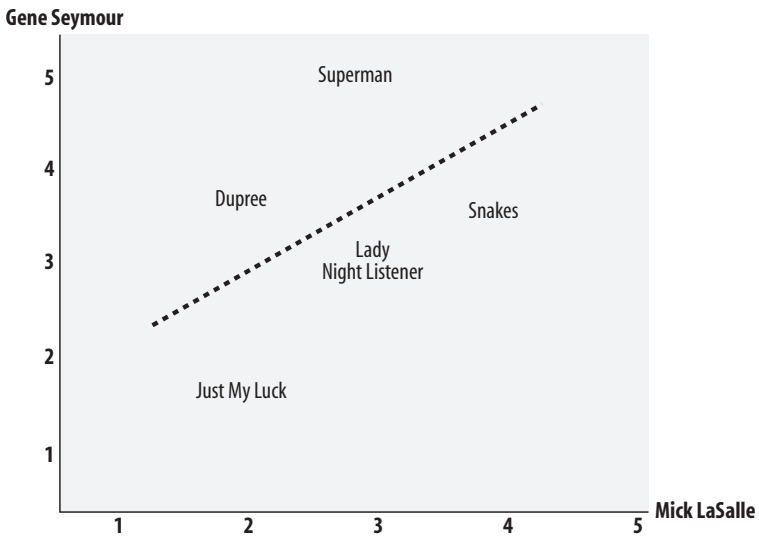
```
CL-USER> (sim-euclidean "Lisa Rose" "Gene Seymour")
0.29429805
CL-USER> (sim-euclidean "Lisa Rose" "Claudia Puig")
0.38742587
```

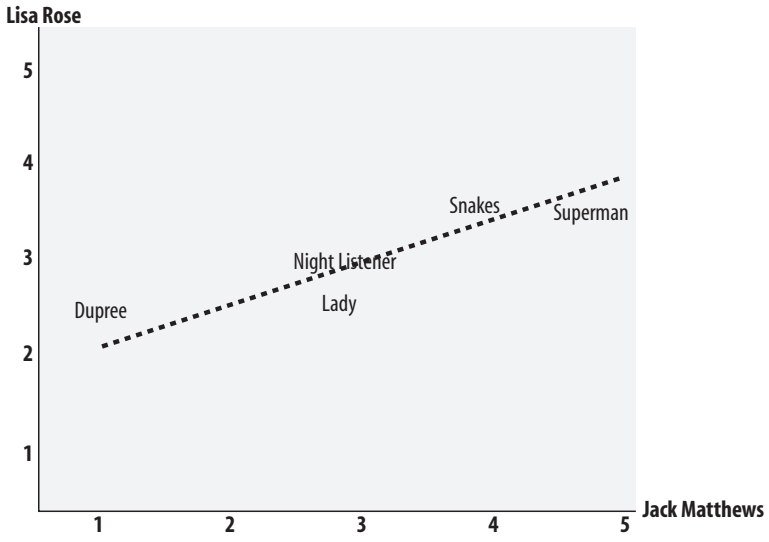
Puig tiene preferencias más cercanas a las de Rose que Seymour.

Universidad Veracruzana

Correlación de Pearson

- $$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$





- Resuelve corrimientos en la escala.

Universidad Veracruzana

Computando

```
1  (defun sim-pearson (person1 person2)
2    "1 means identical preferences, -1 the opposite"
3    (let ((si (shared-items person1 person2)))
4      (if (null si)
5          0
6          (let* ((n (length si))
7                 (prefs1 (loop for film in si collect
8                               (get-film-rating person1 film)))
9                 (prefs2 (loop for film in si collect
10                              (get-film-rating person2 film)))
11                (sum1 (sum prefs1))
12                (sum2 (sum prefs2))
13                (sum1sq (sum (mapcar #'sqr prefs1)))
14                (sum2sq (sum (mapcar #'sqr prefs2)))
15                (psum (sum (mapcar #'(lambda (x y) (* x y)) prefs1 prefs2)))
16                (num (- psum (/ (* sum1 sum2) n)))
17                (den (sqrt (* (- sum1sq (/ (sqr sum1) n))
18                              (- sum2sq (/ (sqr sum2) n))))))
19            (if (zerop den)
20                1
21                (/ num den))))))
```

Universidad Veracruzana

Ejemplos

```
CL-USER> (sim-pearson "Lisa Rose" "Gene Seymour")
0.39605904
CL-USER> (sim-pearson "Mick LaSalle" "Gene Seymour")
0.4117648
CL-USER> (sim-pearson "Lisa Rose" "Jack Matthews")
0.74701745
```

Universidad Veracruzana

Y el top de los críticos es...

```
(defun top-matches (person &optional (n 5) (sim #'sim-pearson))
  (subseq (sort (mapcar #'(lambda(c)
                            (list c (funcall sim person c)))
                        (get-other-critics person))
                #'(lambda(x y) (> (cadr x) (cadr y))))
          0 n))

(defun get-other-critics (critic)
  (remove critic
          (mapcar #'(lambda(x) (getf x :critic))
                  *critics*)
          :test #'string=))
```

```
CL-USER> (top-matches "Toby" 3)
(("Lisa Rose" 0.9912409) ("Mick LaSalle" 0.92447347)
 ("Claudia Puig" 0.8934049))
```

Universidad Veracruzana

Emitiendo recomendaciones

```
(defun get-recommendations (person &optional (similarity #'sim-pearson))
  (let* ((other-critics (get-other-critics person))
         (films-not-seen (set-difference
                          (reduce #'(lambda (x y)
                                      (union x y :test #'string=))
                                  (mapcar #'get-films-rated-by
                                          other-critics))
                          (get-films-rated-by person)
                          :test #'string=)))
    (sort
     (loop for film in films-not-seen collect
          (let* ((sim-sums 0)
                 (total (apply #'+
                               (mapcar #'(lambda(critic)
                                           (let ((rating
                                                  (get-film-rating critic film))
                                                 (sim
                                                  (funcall similarity person critic)))
                                             (if rating
                                                 (progn
                                                   (setf sim-sums (+ sim-sums sim))
                                                   (* rating sim))
                                                 0)))
                                       other-critics))))
            (list (/ total sim-sums) film)))
     #'(lambda(x y) (> (car x ) (car y))))))
```

Universidad Veracruzana

Ejemplos

```
CL-USER> (get-recommendations "Toby")
((3.1192015 "The Night Listener") (3.0022347 "Lady in the Water")
 (2.5309806 "Just My Luck"))
```

Universidad Veracruzana

Invirtiendo la lista de preferencias

- Se trata de indexar ahora por película:

```
1   (defun transform-prefs ()
2     "Inverts the dictionary *critics*, indexing it by film"
3     (let* ((critics (loop for critic in *critics* collect
4                           (getf critic :critic)))
5            (films (reduce #'(lambda (x y) (union x y :test #'string=))
6                           (loop for critic in critics collect
7                                 (get-films-rated-by critic)))))
8       (loop for film in films collect
9             (list :critic film
10                  :critics (remove-if #'(lambda(x) (null (cadr x)))
11                                      (mapcar #'(lambda (c)
12                                                  (list c (get-film-rating c film)))
13                                              critics))))))
```

Universidad Veracruzana

Transformación

- El diccionario luce ahora así:

```
CL-USER> (transform-prefs)
((:CRITIC "Lady in the Water" :CRITICS (("Lisa Rose" 2.5) ("Gene
Seymour" 3.0) ("Michael Phillips" 2.5) ("Mick LaSalle" 3.0) ("Jack
Matthews" 3.0))) (:CRITIC "Snakes on a Plane" :CRITICS (("Lisa Rose"
3.5) ("Gene Seymour" 3.5) ("Michael Phillips" 3.0) ("Claudia Puig"
3.5) ("Mick LaSalle" 4.0) ("Jack Matthews" 4.0) ("Toby" 4.5)))
(:CRITIC "Just My Luck" :CRITICS (("Lisa Rose" 3.0) ("Gene Seymour"
1.5) ("Claudia Puig" 3.0) ("Mick LaSalle" 2.0))) (:CRITIC "Superman
Returns" :CRITICS (("Lisa Rose" 3.5) ("Gene Seymour" 5.0) ("Michael
Phillips" 3.5) ("Claudia Puig" 4.0) ("Mick LaSalle" 3.0) ("Jack
Matthews" 5.0) ("Toby" 4.0))) (:CRITIC "You, Me and Dupree" :CRITICS
(("Lisa Rose" 2.5) ("Gene Seymour" 3.5) ("Claudia Puig" 2.5) ("Mick
LaSalle" 2.0) ("Jack Matthews" 3.5) ("Toby" 1.0))) (:CRITIC "The Night
Listener" :CRITICS (("Lisa Rose" 3.0) ("Gene Seymour" 3.0) ("Michael
Phillips" 4.0) ("Claudia Puig" 4.5) ("Mick LaSalle" 3.0) ("Jack
Matthews" 3.0))))
```

- Así podemos usar las funciones ya programadas.

Universidad Veracruzana

Corrida

- ¿Qué películas se parecen a Superman Returns?

```
CL-USER> (setf *critics* (transform-prefs))
((:CRITIC "Lady in the Water" :CRITICS (("Lisa Rose" 2.5) ("Gene
Seymour" 3.0) ("Michael Phillips" 2.5) ("Mick LaSalle" 3.0) ("Jack
Matthews" 3.0))) (:CRITIC "Snakes on a Plane" :CRITICS (("Lisa Rose"
3.5) ("Gene Seymour" 3.5) ("Michael Phillips" 3.0) ("Claudia Puig"
3.5) ("Mick LaSalle" 4.0) ("Jack Matthews" 4.0) ("Toby" 4.5)))
(:CRITIC "Just My Luck" :CRITICS (("Lisa Rose" 3.0) ("Gene Seymour"
1.5) ("Claudia Puig" 3.0) ("Mick LaSalle" 2.0))) (:CRITIC "Superman
Returns" :CRITICS (("Lisa Rose" 3.5) ("Gene Seymour" 5.0) ("Michael
Phillips" 3.5) ("Claudia Puig" 4.0) ("Mick LaSalle" 3.0) ("Jack
Matthews" 5.0) ("Toby" 4.0))) (:CRITIC "You, Me and Dupree" :CRITICS
(("Lisa Rose" 2.5) ("Gene Seymour" 3.5) ("Claudia Puig" 2.5) ("Mick
LaSalle" 2.0) ("Jack Matthews" 3.5) ("Toby" 1.0))) (:CRITIC "The Night
Listener" :CRITICS (("Lisa Rose" 3.0) ("Gene Seymour" 3.0) ("Michael
Phillips" 4.0) ("Claudia Puig" 4.5) ("Mick LaSalle" 3.0) ("Jack
Matthews" 3.0))))
CL-USER> (top-matches "Superman Returns" 3)
(("You, Me and Dupree" 0.6579514) ("Lady in the Water" 0.48795068)
("Snakes on a Plane" 0.111803316))
```

Universidad Veracruzana

Otra corrida

- ¿Quién recomienda Just My Luck?

```
CL-USER> (get-recommendations "Just My Luck")
((3.8716946 "Jack Matthews") (2.9610002 "Toby")
 (2.2872024 "Michael Phillips"))
```

- ¿Y The Night Listener?

```
CL-USER> (get-recommendations "The Night Listener")
((3.5313943 "Toby"))
```

Universidad Veracruzana

Referencias I

[1] D Goldberg et al. "Using collaborative filtering to weave an information tapestry". En: *Communications of the ACM* 35.12 (1992), págs. 61-71.

[2] A Guerra-Hernández, CR de-la Mora-Basáñez y MA Jiménez-Montaño. "The significance of nucleotides within DNA codons: a quantitative approach.". En: *Avances en la Ciencia de la Computación: VI Encuentro Internacional de Computación ENC'05*. Ed. por L Villaseñor-Pineda y A Martínez-Garcí'a. Sociedad Mexicana de Ciencias de la Computación (SMCC). Puebla, Pue., México: Benemérita Universidad Autónoma de Puebla, 2005, págs. 167-169.

[3] DA Mac Dónaill y M Manktelow. "Molecular informatics: quantifying information patterns in the genetic code". En: *Molecular Simulation* 30.5 (2004), págs. 267-272.

[4] T Mitchell. *Machine Learning*. Computer Science Series. Singapore: McGraw-Hill International Editions, 1997.

[5] T Seagaran. *Programming Collective Intelligence*. Sebastopol, CA, USA: O'Reilly Media, 2008.

[6] P Seibel. *Practical Common Lisp*. New York, NY, USA: Apress, 2005.

Universidad Veracruzana