FUNDAMENTOS DE APRENDIZAJE AUTOMÁTICO

PRÁCTICA 3: ALGORITMOS GENÉTICOS

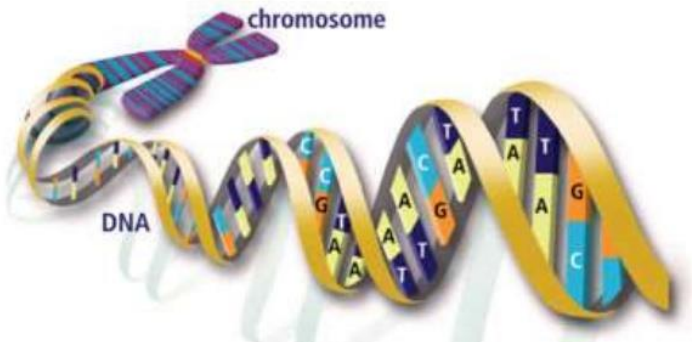
Irene Rodríguez Luján

ALGORITMOS GENÉTICOS

- **Objetivo**: Determinar un conjunto de reglas de clasificación mediante algoritmos genéticos.
  - La forma natural de expresar conceptos es mediante un **conjunto de reglas de clasificación**
- **Algoritmo genético** (Wikipedia): Algoritmos inspirados en la evolución biológica y su base genético-molecular. Se hace evolucionar una población de individuos sometiéndola a acciones aleatorias semejantes a las que actúan en la evolución biológica (mutaciones y recombinaciones genéticas), así como también a una selección de acuerdo con algún criterio, en función del cual se decide cuáles son los individuos más adaptados, que sobreviven, y cuáles los menos aptos, que son descartados.

ALGORITMOS GENÉTICOS

- **Cromosoma**: cadena de información (generalmente binaria) que contiene una posible solución al problema.
  - Los símbolos que forman la cadena se denominan **genes**.
  - Los cromosomas evolucionan a través de iteraciones, llamadas **generaciones**.
  - En cada generación, los cromosomas son evaluados usando alguna **medida de aptitud o *fitness*.**
  - Las siguientes generaciones (nuevos cromosomas), son generadas aplicando los **operadores genéticos** repetidamente, siendo estos los operadores de **selección**, **cruzamiento**, **mutación** y **reemplazo**.



Chromosome 1	1101100100110110
Chromosome 2	1101111000011110

PRÁCTICA 3

Spears, William M., and Kenneth A. De Jong. "Using genetic algorithms for supervised concept learning." *Tools for Artificial Intelligence, 1990., Proceedings of the 2nd International IEEE Conference on*. IEEE, 1990.

OBJETIVO

Implementar los métodos **entrenamiento** y **clasifica** de la clase **ClasificadorGenetico** (hereda de Clasificador) para determinar un conjunto de reglas de clasificación mediante algoritmos genéticos.

- Tenéis bastante **libertad** en el diseño de las estructuras de datos, operadores genéticos y criterio de clasificación.

Tareas.

1. Determinar el esquema de representación más adecuado para codificar cada una de las soluciones.
2. Determinar el conjunto de operadores genéticos y su operativa concreta.
3. Definir la función de adaptación o función de *fitness*.

Esquema de representación

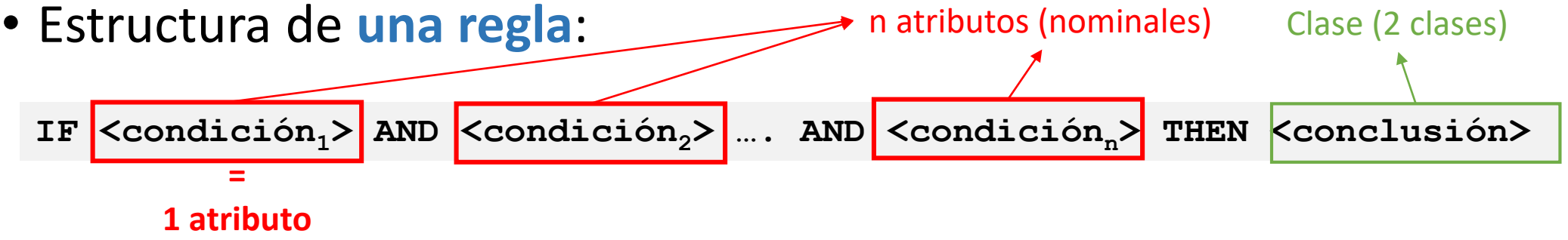
- Dos enfoques respecto a la forma de codificar las reglas dentro de una población de individuos:

1. Enfoque **"*Cromosoma = Regla*"**. En esta aproximación cada individuo codifica una sola regla. A este enfoque corresponden dos propuestas:
   a) En la propuesta ***Michigan***, la solución final será la población final o un subconjunto de la misma.
   b) En la propuesta conocida como ***Aprendizaje de Reglas Iterativo***, la solución del algoritmo genético es el mejor individuo, pero la solución global estará formada por los mejores individuos de una serie de ejecuciones sucesivas.

2. Enfoque **"*Cromosoma = Base de Reglas*"**. En esta aproximación, que se conoce con el nombre de ***Pittsburgh***, cada individuo representa un conjunto de reglas.

Esquema de representación

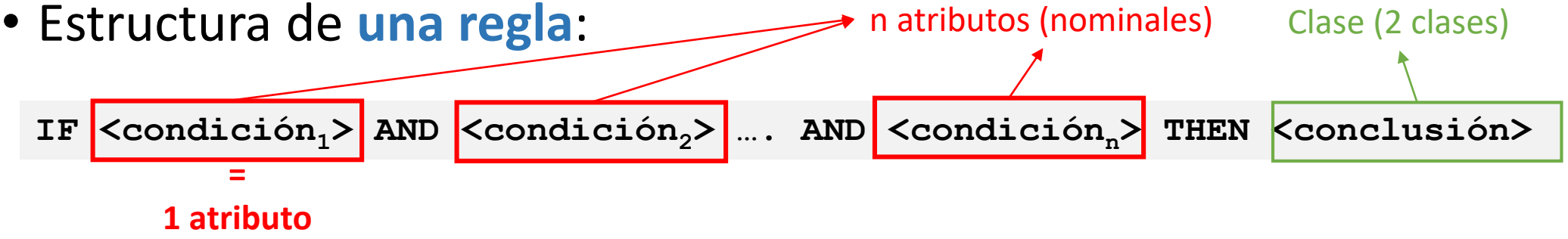
- Estructura de **una regla**:



- Cada condición puede incluir también el operador **OR** para combinar condiciones disyuntivas.
    - Cada condición se puede representar mediante una cadena binaria en la que un 1 representa que el valor del atributo o característica está presente y un 0 representa la ausencia del valor del atributo o característica
- La conclusión representa la clase del patrón y solo se trabajará con problemas binarios.
    - Codificación de la condición con un bit.
- **OJO!** Aunque se podría utilizar implementación a nivel de bit para codificar las reglas, se recomienda utilizar **vectores de enteros 0/1**.

Esquema de representación

- Estructura de **una regla**:



- **Ejemplo**: clasificar supervivencia o no en el titanic (0: no sobrevive / 1: sobrevive) en base a la clase del billete (3 posibles clases: clase 1, clase 2, clase 3) y género del pasajero (hombre/mujer).
  - ¿Cuántas condiciones tiene la regla?
  - ¿Cuántos 0/1s serán necesarios para codificar una regla?
  - ¿Codificación de la regla IF (CLASE=2 || CLASE=3) AND (GENERO=HOMBRE) THEN NO SOBREVIVE?

    011010
  - ¿Codificación de la regla IF (CLASE=1) THEN SOBREVIVE?

    100111

Esquema de representación

- Conjunto de reglas **disyuntivas**:
  - Cada individuo de la población corresponde a una base de reglas completa, por lo que cada individuo se representará mediante una cadena binaria de longitud variable que incluirá un conjunto no ordenado de reglas de longitud fija como las descritas anteriormente.
- El número de reglas de cada individuo no está restringido en principio, siendo éste un parámetro bastante relevante que deberá analizarse con detalle.
- Problema a resolver / decisión a tomar: ¿cómo se clasifica?

Algoritmo genético básico

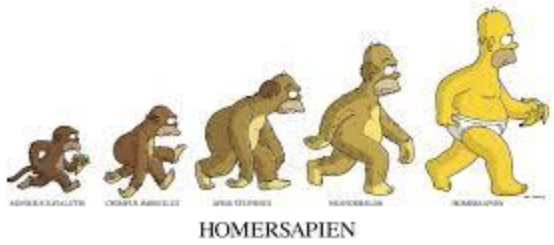
1. Producir una población inicial de individuos — Función inicializacion(…)
2. Evaluar el *fitness* de los individuos — Función fitness(…)
3. Mientras que no se satisfaga la condición de terminación: — Función cruce (…)
   a) Seleccionar individuos para la reproducción de acuerdo a cierto criterio
   b) Cruzar a los individuos
   c) Mutar a los individuos resultantes de 3.b — Función mutacion(…)
   d) Evaluar el *fitness* de los individuos resultantes de 3.c
   e) Generar una nueva población con los individuos de 3.c y (posiblemente) individuos de la generación anterior.
      - P3: Mantener el tamaño de la población a lo largo de las generaciones.

Parámetro número de generaciones: 100, 500, 1000

- Los mejores de la generación anterior (porcentaje de elitismo)
- De los cruces+mutación
- Selección aleatoria de la generación anterior por fitness
- ….

Función inicialización

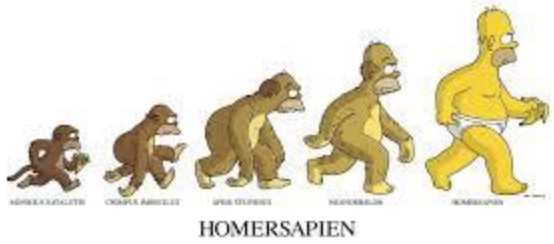
- **Parámetros involucrados**:
  - Tamaño de la población: 10, 100, 200 y 500 individuos
  - Número máximo de reglas
- **Funciones útiles**:
  - numpy.random.randint



Función de *fitness*

Fitness del individuo i = % de aciertos sobre los ***datos de entrenamiento***

- **Sugerencia**: llamada a clasifica con cada individuo de la población
- ¿Qué tiene que hacer clasifica?
    1. Evaluar reglas del individuo
    2. Emitir predicción → tenéis que decidir qué criterio de predicción usaréis
        - Clasifica devolverá un array con las predicciones para cada patrón a testear.



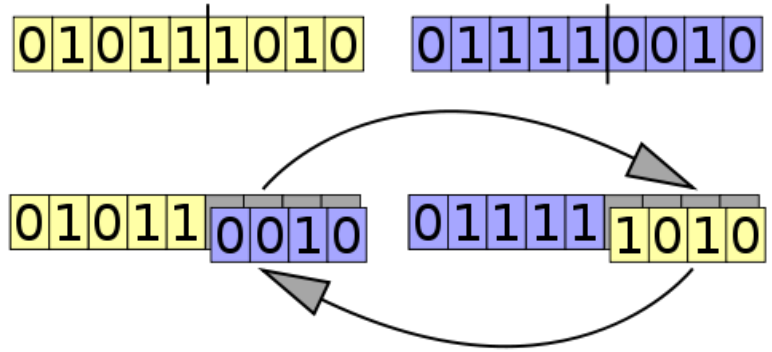
Función de cruce

- **Muchas formas de implementarla!**

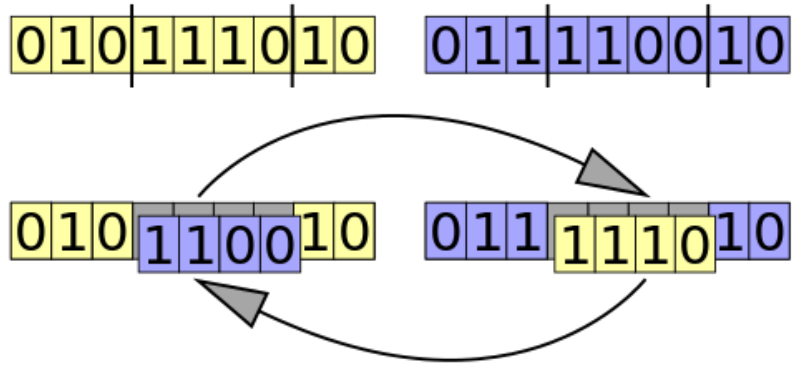
  https://es.wikipedia.org/wiki/Recombinaci%C3%B3n_(computaci%C3%B3n_evolutiva)

- **Parámetros involucrados**:
  - Probabilidad de cruce: 60%
    - Probabilidad de que una pareja cruce.
  - Selección mediante *fitness*

- **Funciones útiles (I)**:
  - Generación de **parejas**: np.random.choice
    - OJO! Parámetros replace y p.
    - Elegir cuántas parejas generar. Por ejemplo: tamaño de población/2
  - Selección de parejas de acuerdo al parámetro de **probabilidad de cruce**: numpy.random.uniform
    - **OJO!** Pueden salir diferente número en función de la generación de probabilidades aleatoria



Recombinación en un punto



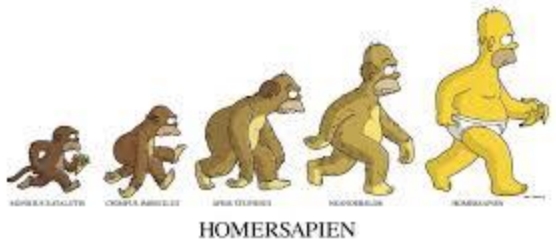
Recombinación en dos puntos

Función de cruce

- **Funciones útiles (II)**:
  - Al **generar los hijos** a partir de los padres, hay que tener cuidado al generarlos puesto que es posible que no queramos modificar los padres.
    - Si en Python hacemos algo del estilo:

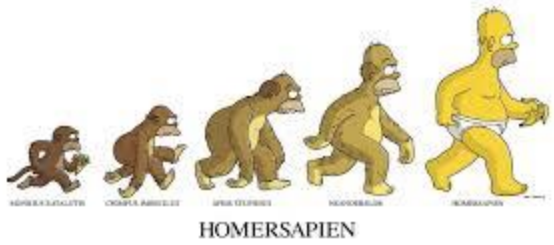
      ```
      Hijo=padre1

      Hijo[n:m]=padre2[n:m]
      ```

    - Padre1 también se modifica puesto que se trata de punteros.
    - Posible solución: función **deepcopy** del módulo copy (ya importada en ClasficadorMulticlase)
  - Al **generar los puntos de cruce** aleatoriamente:
    - Selección aleatoria: puede ser útil utilizar la función **numpy.unravel_index**
    - ¿Rango de selección de los puntos de cruce?



Función de cruce

- **Parámetros involucrados**:
  - Probabilidad de mutación: 0.1%
    - Probabilidad de que un bit mute.
- **Funciones útiles**:
  - **Mutación**: numpy.logical_not
  - Selección de **posiciones a mutar** de acuerdo a la probabilidad de mutación: numpy.random.uniform



Entrega

- Semana del 5 al 9 de diciembre de 2016 antes de la hora de clase.
- Notebook + *.py
- Conjuntos de datos: tic-tac-toe.data (P0), example2.data (P1), ejemplo5.data, ejemplo6.data, titanic.data
- Mostrar mensajes informativos durante la salida del programa.
- **Apartado 2**: mostrar por pantalla el conjunto de reglas asociadas al mejor individuo tras el entrenamiento junto con los diccionarios. Analizar estas reglas posteriormente en el **apartado 3**.

Entrega

- **Apartado 4**: Solo para la fase de entrenamiento, evolución en forma de gráfica:
  - Del fitness del mejor individuo de la población
  - Del fitness medio de la población
- Es suficiente con que representéis la evolución de la última partición de training/test (por ejemplo, mediante validación cruzada con 3 grupos).
- Utilizar el modulo **matplotlib**.

```
import matplotlib.pyplot as plt
%matplotlib inline

print errores
print errores.mean()
print clasificador.best
print dataset.diccionarios

plt.figure(figsize=((15,5)))
plt.hold(True)
plt.plot(clasificador.ev_best,'bo-',label='Mejor')
plt.plot(clasificador.ev_media,'ro-',label='Media')
plt.legend(loc=4)
plt.xlabel('Generacion')
plt.ylabel('Fitness')
plt.ylim(np.min([clasificador.ev_best,clasificador.ev_media])*0.95,\
         np.max([clasificador.ev_best,clasificador.ev_media])*1.05)
```

```
[ 0.  0.  0.]
0.0
[[0 1 1 1 1]]
[{'+': 0, '-': 1}, {'+': 0, '-': 1}, {'+': 0, '-': 1}]
```