**Anotaciones guía 4**

**Computación evolutiva e inteligencia de enjambre**

**Cosas habladas en la evaluación de esta guía:**

-En el ejercicio 1, en el algoritmo de cruza, para el caso de la función f2 que es de dos variables f2(x, y), nosotros tenemos un cromosoma que es una lista donde primero tiene los genes o bits para “x” y luego para “y”, y nosotros hicimos lo mismo que con f1 de elegir un solo punto de cruza que puede caer en la parte de “x” o en la parte de “y”, y ahí hacemos la cruza.

Una mejora podría ser tomar por separado la parte para “x” y la parte para “y”, y luego en la cruza tomar dos puntos, uno en la parte de “x” y otro en la parte de “y”, y cruzar en esos dos puntos.

La diferencia visual es que en el primer caso nosotros vamos a ver que se mueve en “x” o en “y”, porque modifica una de las partes, mientras que con la otra forma los puntos se van a mover en diagonal porque estaremos modificando tanto “x” como “y”.

-Dependiendo del problema, hay que recordar que si el fitness es negativo, puede generar problemas por ejemplo en el algoritmo de selección con ruleta.

-En el ejercicio 2, cuando tenemos la función:

fitness[i] = alpha \* accuracy - beta \* selectedFeature/totalFeature

Tanto accuracy como el término selectedFeature/totalFeature varían entre 0 y 1, entonces si yo uso todas las características voy a tener 7129/7129 = 1, entonces si por ejemplo tengo accuracy 0.5, al aplicar la fórmula obtengo fitness = 0.5 – 1 = -0.5 y me va a quedar negativo, y podría tener problema como mencioné antes.

-En el ejercicio 2, también para comparar si verdaderamente nuestra solución utilizando menos características mejora el resultado, deberíamos tener una solución usando todas las características, por ejemplo, en el archivo de mejorIndv colocar las 7129 características con 1, y usar ese para calcular el accuracy, entonces luego cuando calculemos el accuracy con nuestra solución (nuestro individuo con menos características), vamos a poder comparar si en verdad mantuvo un buen accuracy utilizando menos características.

**EJERCICIO 1**

<https://drive.google.com/drive/folders/1U0tUQK7EB05OsQpdo2_qdjOlkuqAzrUH>

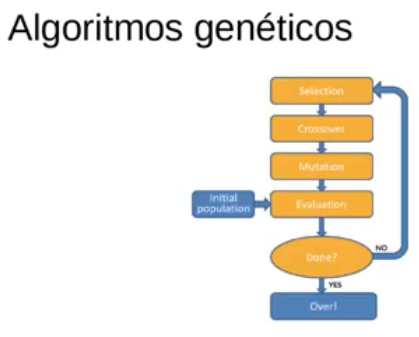
Se pide implementar un algoritmo genético y luego para probar este algoritmo vamos a abordar un método de optimización muy sencillo que consiste en buscar el mínimo global de las funciones que tenemos. En nuestro algoritmo, la función de aptitud o fitness va a estar dada en cada caso por la ecuación de cada función, y las soluciones, es decir, los individuos que componen la solución en nuestro algoritmo genético, van a representar los valores de “x”, es decir que, al decodificar el cromosoma de cada individuo va a representar un valor de “x” en el rango determinado para cada función.

A continuación vamos a hacer un **repaso de algoritmos genéticos**. En los vídeos de teoría esta explicado más en detalle.

Los algoritmos genéticos están inspirados en los principios de selección y evolución natural. Siguiendo la imagen de abajo, tenemos una población de individuos, donde cada individuo representa una solución al problema. Comenzamos con una población inicial (“initial population”, rectángulo azul a la izquierda de la imagen), iniciada al azar, luego esa población se evalúa (Evaluation) mediante la función de aptitud o fitness que depende del problema, y luego vamos iterando mientras el mejor individuo de la población no alcance la aptitud deseada, o podría ser una cantidad de iteraciones máxima, e iteramos aplicando los operadores: primero se aplica el operador de selección (Selection), que selecciona padres entre la población que van a tener hijos a partir de los operadores de cruza (Crossover) y mutación (Mutation).

Recordemos que este operador de selección permite elegir cualquier individuo de la población pero dándoles mayor probabilidad a los individuos más aptos, luego se aplica el operador de cruza que lo que hace es combinar información útil de distintos individuos (padres), para generar nuevos individuos (hijos) más aptos. Y también tenemos el operador de mutación, el cual modifica aleatoriamente el valor de un gen del cromosoma, lo cual le permite al algoritmo explorar distintas zonas del espacio de búsqueda y, a su vez, escapar de mínimos locales.

Una vez que tenemos la nueva población, volvemos a evaluar cada individuo (Evaluation) con la función de aptitud o fitness y así vamos iterando a lo largo de las generaciones hasta encontrar un nivel de aptitud deseado o alcanzar un máximo de iteraciones. De esta forma se espera que a lo largo de las iteraciones, las poblaciones tengan individuos cada vez más aptos.



La principal **ventaja** de este algoritmo es que permite optimizar cualquier tipo de función, no necesita que esa función tenga características como la existencia de derivada o continuidad, incluso ni siquiera necesita una expresión matemática de la función, podría ser por ejemplo un algoritmo, lo único que necesita es que la función nos dé una medida de que tan buena es cada posible solución, y eso lo hace muy flexible ya que nos permite resolver infinidad de problemas de optimización.

Diferencia entre cromosoma y gen:

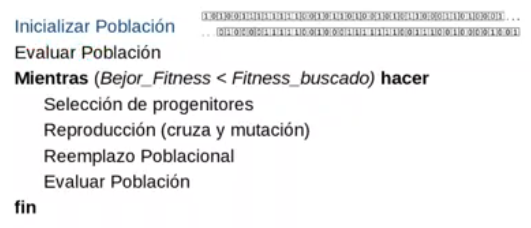
Cromosoma: Toda la cadena de bits.

Gen: Representación de una de las partes de la solución (un bit).

**Pseudocódigo algoritmos genéticos**

Como se ve en la imagen de abajo, comenzamos inicializando la población de forma aleatoria. Cada individuo está representado por un cromosoma que es una cadena binaria, es decir que, para inicializarlo vamos a completar esa cadena binaria con unos y ceros de forma aleatoria.

Como segundo paso vamos a evaluar la población aplicando la función de fitness dada por el problema que queremos optimizar, y previamente tenemos que decodificar el cromosoma, es decir, llevar los valores del genotipo al genotipo en el dominio donde se puede evaluar con la función de fitness y luego, mientras no se cumpla la condición de criterio de corte, vamos a aplicar un método de selección, luego se aplican los operadores de cruza y mutación para la reproducción y generar la descendencia (hijos) y una vez que tenemos la descendencia de eso se pueden aplicar distintas estrategias de reemplazado poblacional, y luego se vuelve a evaluar la población e iterando hasta cumplir el criterio de corte.

****

**Representación del cromosoma**:

En los algoritmos genéticos que vamos a implementar, la representación es binaria, así que *los cromosomas son cadenas binarias.*

Lo que represente cada gen (bit) del cromosoma depende del problema en cuestión, por ejemplo, podría ser que cada gen represente un switch y que indique si esta encendido o apagado. En problemas como los que vamos a resolver nosotros, en el cromosoma representamos valores numéricos, por lo tanto, tenemos que decodificar esa cadena binaria en uno o más valores numéricos (como en la imagen de abajo). Si estamos representando un solo valor, decodificamos todo el cromosoma en un solo valor (real o entero), y si estamos codificando varios valores tendremos una cantidad de bits para cada valor, que no necesariamente debe ser la misma cantidad de bits sino que puede ir variando, y al decodificar iremos teniendo en cuenta los bits que representa cada variable en nuestro problema.



**Operadores de selección: Ruleta**

A cada individuo de la población se le asigna un área en la ruleta, proporcional a su valor de aptitud o fitness, por lo tanto, los individuos más aptos tendrán un área más grande que los menos aptos. Al simular el giro de la ruleta habrá más probabilidad de que caiga en un área grande, es decir, es más probable que se elijan los individuos más aptos.

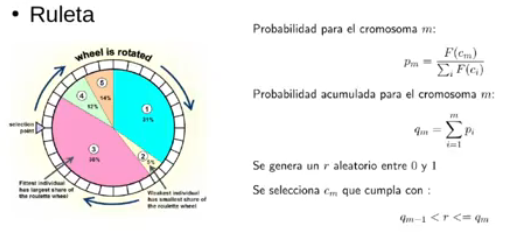
**Implementación de ruleta**:

-Se calcula una probabilidad para cada cromosoma “m” (pm), que se calcula como el valor de fitness para ese cromosoma F(cm) sobre la suma total de los fitness para todos los individuos.

-Luego se calcula una probabilidad acumulada para el cromosoma “m” (qm) que se calcula como la sumatoria de todas las probabilidades pi de los cromosomas desde 1 hasta m, es decir, desde el primer individuo hasta el individuo “m” para el cual estoy calculando su probabilidad acumulada.

-Se genera un valor aleatorio “r” entre 0 y 1.

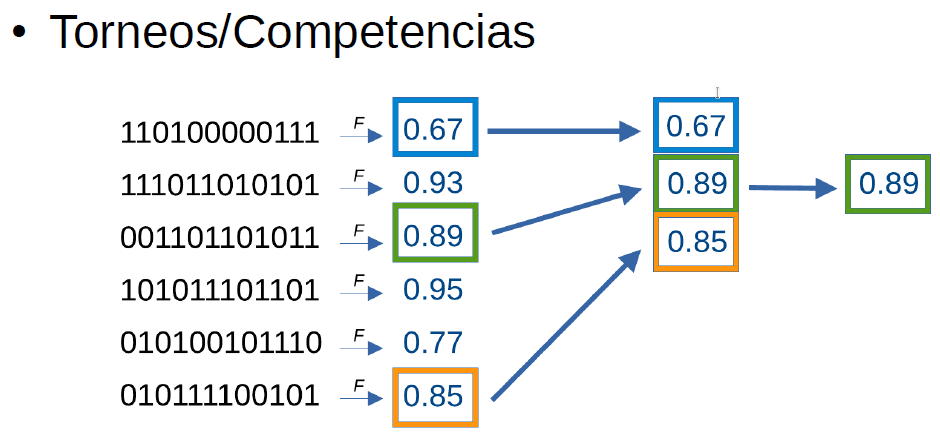
-Se elige el cromosoma cm para el cual se cumpla que “r” esta entre qm-1 y qm, y de esa forma elegimos con el método de la ruleta.



Si bien la ruleta es un método muy usado, tiene **dos problemas**: mar de mediocres y mar de virtuosos, como vimos en teoría. El mar de mediocres significa que si tenemos muchos individuos de baja aptitud y pocos individuos buenos, tenemos mayor chance de elegir un individuo malo que uno bueno, y el mar de virtuosos es que si tenemos todos los individuos con poca diferencia entre ellos, por ejemplo 49.9, 50, 50.1, etc. va a terminar siendo una elección al azar. Por eso existen otros métodos que solucionan estos problemas.

**Operadores de selección: torneos/competencias**.

Consiste en elegir “n” individuos al azar y luego de esos “n” nos quedamos con el de mejor fitness. Ese “n” es un parámetro del método y mientras más grande sea más chances tendremos de elegir los individuos más altos.

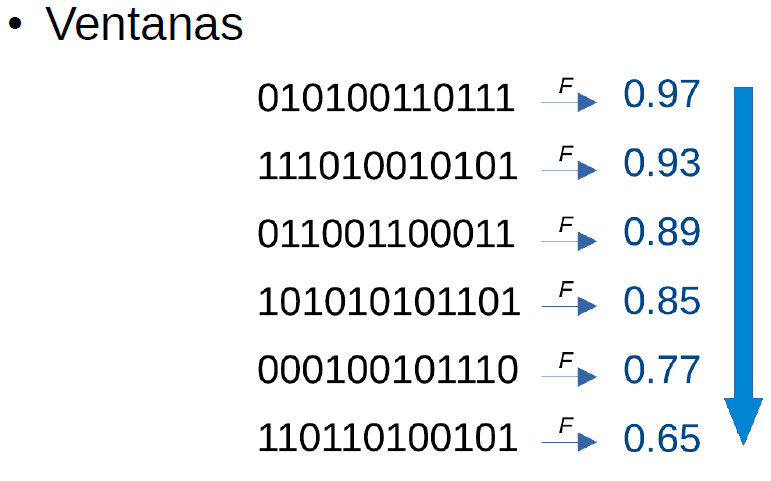
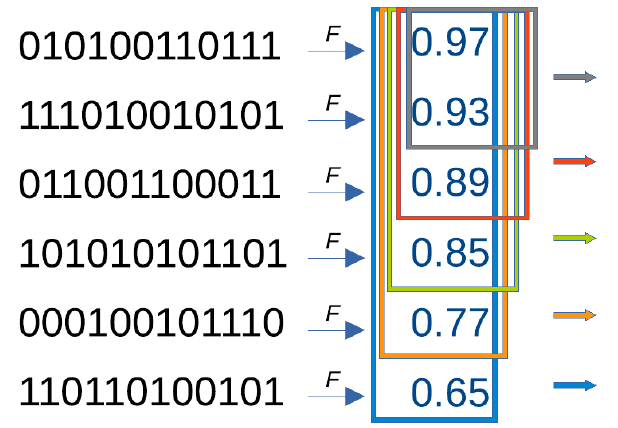


**Operadores de selección: ventanas**.

Consiste en ordenar la población de mayor a menor fitness, luego tomamos una ventana que abarque a todos los individuos y tomamos uno al azar, con la misma probabilidad para todos. Ahí puede salir un individuo con alto fitness o bajo. Luego achicamos esa ventana, por ejemplo, desde el individuo de mejor fitness hasta el anteúltimo (o un 80% o cosas así), y elegimos otro individuo, pero entonces ahora el de peor fitness (el que estaba último) ya no tiene posibilidad de ser elegido porque no entró en la ventana, y así seguimos achicando la ventana y eligiendo nuevos individuos hasta completar la población.

Lo que logramos es que los individuos más aptos estén en mayor cantidad de ventanas y tengan mayor probabilidad de tener descendencia (hijos).

Como vamos achicando la ventana para quedarnos solo con los de mejor fitness (porque están ordenados), si hacemos un listado de los fitness que vamos obteniendo siempre al final tendremos valores más altos (no necesariamente siempre creciente porque se elige al azar) porque vamos a estar eligiendo entre los mejores.

**Operadores de variación: cruza**

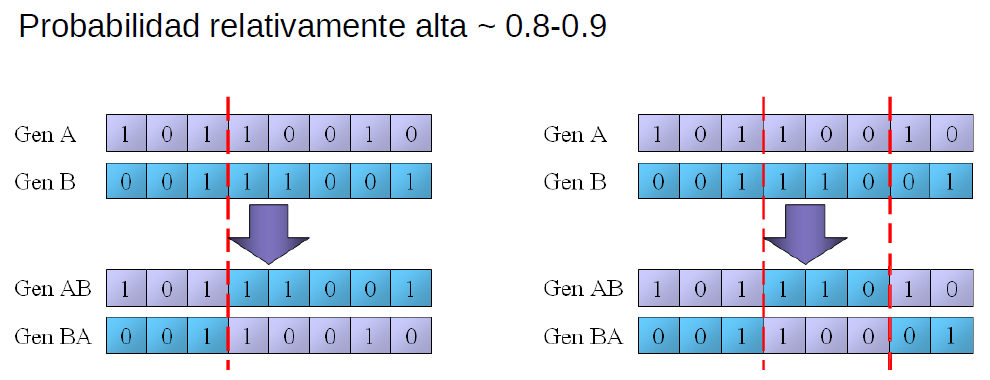
Permite combinar información genética de distintos padres para generar los hijos.

Una vez que tenemos los padres elegidos mediante el método de selección, lo que hacemos es tomarlos de a pares, como muestra la imagen de abajo (Gen A y Gen B), y los vamos a cruzar con cierta probabilidad. En general se usa una probabilidad relativamente alta como 0.8 o 0.9, y entonces lo que hacemos es generar un número aleatorio entre 0 y 1, así si ese número aleatorio es menor que la probabilidad elegida cruzamos los individuos padres, y si es mayor, los individuos hijos son exactamente iguales a los padres.

En el caso de cruzarlos, primero elegimos un punto de cruza al azar (marcado con la línea roja punteada) y a partir de ese punto de cruza intercambiamos la información de los individuos padres, por ejemplo, vemos que el hijo Gen AB tiene la primera parte del padre A (hasta el punto de cruza) y la segunda parte del padre B (desde el punto de cruza hasta el final), mientras que el otro hijo Gen BA, en la primera parte tiene los genes del padre B y en la segunda parte los genes del padre A.

Es decir que, a partir de la cruza de dos padres generamos dos hijos combinando la información genética de esos dos padres.

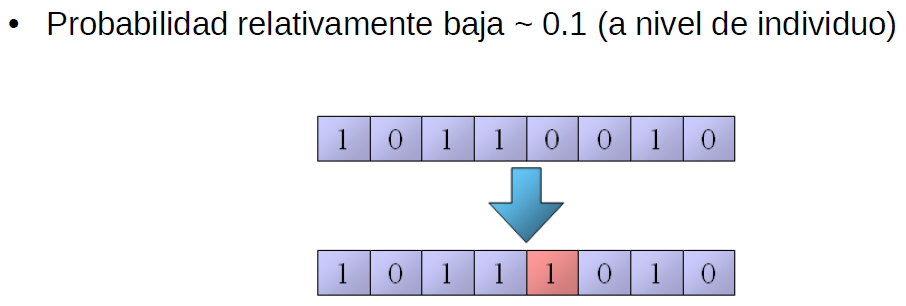
Ese ejemplo sería con un punto de cruza, pero también se puede hacer con varios puntos de cruza, como se muestra a la derecha en la imagen, donde vemos un ejemplo con dos puntos de cruza y el procedimiento es exactamente igual al anterior.



**Operadores de variación: mutación**

Una vez que generamos la descendencia, es decir, los individuos hijos mediante el operador de cruza, vamos a aplicar el operador de mutación a cada hijo.

Vamos a usar una probabilidad baja, alrededor de 0.1, a nivel de individuo (puede ser 0.2 en el caso de que se use elitismo), y consiste en generar un valor al azar entre 0 y 1, si ese valor es menor a la probabilidad de mutación significa que vamos a mutar el cromosoma, para lo cual vamos a elegir de forma aleatoria el gen a mutar, y si ese gen tiene un valor 0 lo cambiamos a 1 y si tiene un valor 1 lo cambiamos a 0.



Esa es la forma más sencilla de implementar la mutación en el caso de cromosomas binarios. Otra forma de implementarla puede ser considerando una probabilidad de mutación individual para cada gen, es decir, consideramos una probabilidad más baja a nivel de cada gen. Entonces en lugar de elegir al azar una vez el gen que se va a mutar, podemos tirar la moneda para cada gen de forma independiente, entonces podría ser que haya más de un gen mutado. De todas formas, para ese caso hay que tener en cuenta que la probabilidad total del cromosoma (toda la cadena de genes o bits) de ser mutado no debería ser superior a 0.1 aproximadamente, es decir, debería ser baja.

Se podría decir que la cruza utiliza información ya existente, mientras que la mutación agrega nueva información.

**Criterios de finalización o corte de los algoritmos genéticos**

-Fitness deseado.

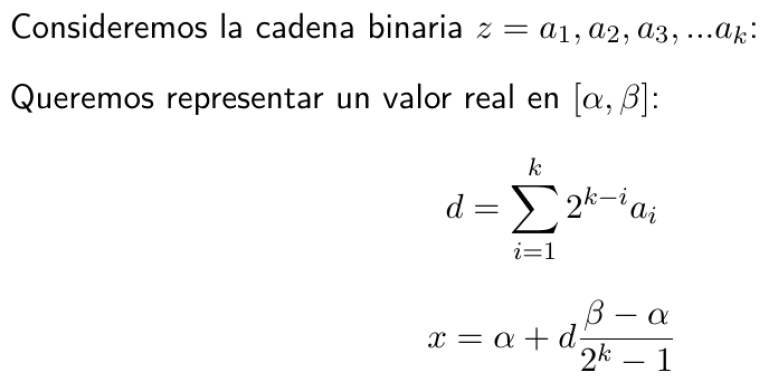
-Cantidad máxima de generaciones.

-Sin mejoras en el fitness por “n” generaciones.

La idea sería que si se alcanza un valor de fitness deseado se detiene la iteración. Pero si no se obtiene el valor de fitness deseado, podemos usar una cantidad máxima de generaciones. El problema es que tampoco sabemos cuánto va a tardar el algoritmo en converger, entonces si seteamos una cantidad máxima de generaciones que sea un poco chica podemos terminar cortando el algoritmo de forma prematura, entonces conviene setear una cantidad máxima alta de generaciones y luego podemos combinarlo con otro criterio donde vamos siguiendo el mejor fitness entre las generaciones y si ese mejor fitness no tiene mejoras a lo largo de “n” generaciones, ahí podemos considerar que el algoritmo convergió y cortamos.

**Decodificación**

Lo que sigue es decodificar nuestros cromosomas en valores enteros o reales. Es decir, tenemos cromosomas que son cadenas binarias, como el cromosoma “z” de la imagen que es una cadena de “k” bits, y los queremos mapear en un intervalo de valores reales entre α y β, para obtener un valor real, entonces primero los mapeamos a un decimal “d”, es decir, convertimos de binario a decimal con la sumatoria indicada, y una vez que tenemos dicho valor decimal usamos la última fórmula para obtener “x” entre α y β, que sería el valor “x” que necesitamos para evaluar el individuo mediante nuestra función de fitness.



***Ver bien*** *cómo explicar esas fórmulas por si las preguntan en el* ***parcial****.*

*(El profe se los explicó en el pizarrón a unos alumnos, pensarlo bien o preguntar).*

*Sofi tenía anotado lo siguiente sobre decodificación*:

: 0 ó 1

d: entero

Cuantos más bits se tienen, mayor representación o “granularidad” tenemos

(bits de 0 a 512, codificados con 00, 01, 10 y 11)

0 01 10 512

00 11

🡪 escala

🡪 traslada

🡪 sería como un delta

**Otras consideraciones importantes**

-Mapeo de fitness en problemas de minimización:

Otra cuestión a tener en cuenta es que en los algoritmos genéticos buscamos maximizar el fitness, es decir, queremos obtener el mayor valor posible de fitness. Mientras que, en algunos problemas de optimización como en el caso del ejercicio de la guía donde tenemos que buscar el mínimo global de ciertas funciones, se tratan de problemas de minimización, entonces en esos casos podemos hacer un mapeo entre la función objetivo y la función de fitness, por ejemplo, podríamos considerar que la función de fitness “F” sea F = 1/f, donde “f” sería el valor de la función y de esa forma lo convertimos a un problema de maximización.

Otra opción sería hacer el fitness F = 1 – (f/max(f)), es decir, 1 menos la función sobre el valor máximo de la función en caso de que sepamos cual es, y de esa forma podemos mapear el valor de la función objetivo en un valor de fitness y pasamos de un problema de minimización a uno de maximización.

Por último, hay dos *estrategias para tener en cuenta al momento de reemplazar la población por su descendencia*, que son el elitismo y la brecha generacional.

-**Elitismo**: consiste en copiar de la población actual el mejor individuo que tenemos, y pasarlo a la población siguiente, sin ninguna modificación, es decir, sin aplicarle operadores de variación.

Eso tiene la ventaja de que no vamos a perder la mejor solución encontrada hasta el momento, que de otra forma con los operadores de variación se podría perder. Además, nos permite aumentar la probabilidad de mutación, permitiendo una mejor exploración del espacio de búsqueda sin riesgo de complicar la convergencia del algoritmo.

(Sofi tenía anotado que si no se hace elitismo, el fitness puede oscilar).

-**Brecha generacional**: consiste en seleccionar una cantidad determinada de individuos de la población, empleando el algoritmo de selección, es decir, dándole mayor probabilidad a los individuos más aptos, y también copiar esos individuos sin modificación para pasarlos a la población de la generación siguiente. De esta forma se favorece la estabilidad de convergencia del algoritmo.

**Volviendo al ejercicio 1**

Una vez implementado el algoritmo genético, lo debemos probar buscando el mínimo global de esas funciones “f(x)”, donde la primera es una función de una variable y que tiene varios mínimos locales, y la segunda función f(x, y) es de dos variables entre -100 y 100, que es la graficada en el pdf donde también se ve que tiene muchos mínimos locales y el algoritmo genético debería ser capaz de encontrar el mínimo global de la función.

Por último, para comparar el desempeño del algoritmo genético, se pide implementar el método del gradiente descendente para buscar el mínimo global de las funciones, para lo cual vamos a partir de un punto elegido al azar y hacer la búsqueda en base al gradiente, repitiendo eso “n” veces para que sea comparable al algoritmo genético.

Esa comparación es simplemente hacer una búsqueda del mínimo por el método del gradiente descendente. Tendremos una función f(x) (o f(x, y) en el inciso ii) que queremos minimizar. Partiendo de un punto x\_0 cualquiera (elegido al azar), sabemos que la dirección del gradiente en x\_0 es la dirección donde la función crece más rápido, por lo tanto la dirección opuesta (-gradiente) será la que decrece más rápido. De esa forma, haces una iteración donde x\_t+1 = x\_t - \alfa gradiente con un \alfa pequeño. Eso debe converger al mínimo local más próximo al punto inicial donde hayas empezado.

**EJERCICIO 2**

Sobre este ejercicio no hay video, pero dejo la explicación que el profe nos dio por correo.

Este problema es fácil si ya tenés implementado el algoritmo genético. Tenés un conjunto de datos donde tenés pocos patrones (38), pero en una dimensión muy grande (7129, es decir, la cantidad de entradas para hacer un clasificador es enorme). Una forma de encarar este problema es reducir el número de entradas, "eligiendo" un subconjunto para el cual puedas tener un buen desempeño del clasificador. Para eso es que vas a usar un algoritmo genético.

Para usar lo que ya implementamos debemos cambiar 2 cosas simplemente: la codificación de los individuos y la función de fitness:

1) individuos: cada individuo es una potencial solución, o sea una selección de qué entradas debemos usar en el clasificador. Vas a usar una cadena binaria, de igual longitud que la cantidad de características (entradas) del dataset, donde cada bit indica si esa característica la vas a usar o no en el entrenamiento. Por ejemplo, suponiendo que hubiera 5 entradas solamente, una cadena 1 0 0 1 0 te diría que tenés que usar sólo las características en la primera y cuarta posición, y el resto descartarlas.

En realidad lo que se cambia es **lo que representan esos cromosomas o individuos**, ya que en el ejercicio 1 las cadenas con ceros y unos representan una característica, mientras que en el ejercicio 2 son una cadena de características, cada bit es si esa características esta activada y se usa (1) o si no (0).

2) lo otro que tenés que modificar es la función de fitness. Y ¿cómo se hace eso? primero tenés que tener claro el objetivo, que en realidad son dos objetivos: por un lado querés que el clasificador funcione muy bien (maximizar el accuracy), y por otro queres que use la menor cantidad de características posibles (minimizar la cantidad de características).

Entonces vamos a usar una función de fitness con dos términos, uno es directamente el accuracy de un clasificador, y el otro tiene que restar fitness si el número de características usadas es muy alto. Entonces fitness = \alfa\* accuracy - \beta (número de características elegidas del individuo/número de características totales), donde \alfa y \beta son parámetros que controlan qué tanta importancia se le da a estos dos objetivos (si alfa es más grande, el algoritmo tratará de maximizar el accuracy sin importarle tanto usar más cantidad de características, si beta es más alto tratará de reducir la cantidad de características sacrificando un poco de accuracy, y así)

*(Esa fórmula de fitness está más abajo en una imagen del PDF).*

Lo único que falta es definir de dónde sacar el accuracy. Y eso se hace ENTRENANDO UN CLASIFICADOR (el que queramos de los que usamos en la guía 3 sobre de scikit learn).

Resumiendo:

- Codificas en binario los individuos con una posición para cada característica que querés ver si retenes o no.

- Para cada individuo, usas su codificación para elegir qué características vas a usar.

- Recortas el dataset para usar sólo esas características.

- Entrenas un clasificador (el que quieras) con los datos de entrenamiento.

- Obtenés el accuracy del clasificador.

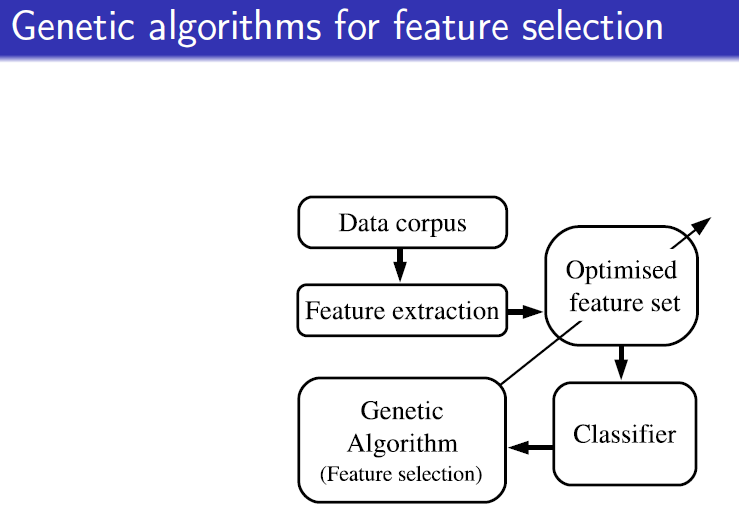
- Con eso calculas el fitness de ese individuo

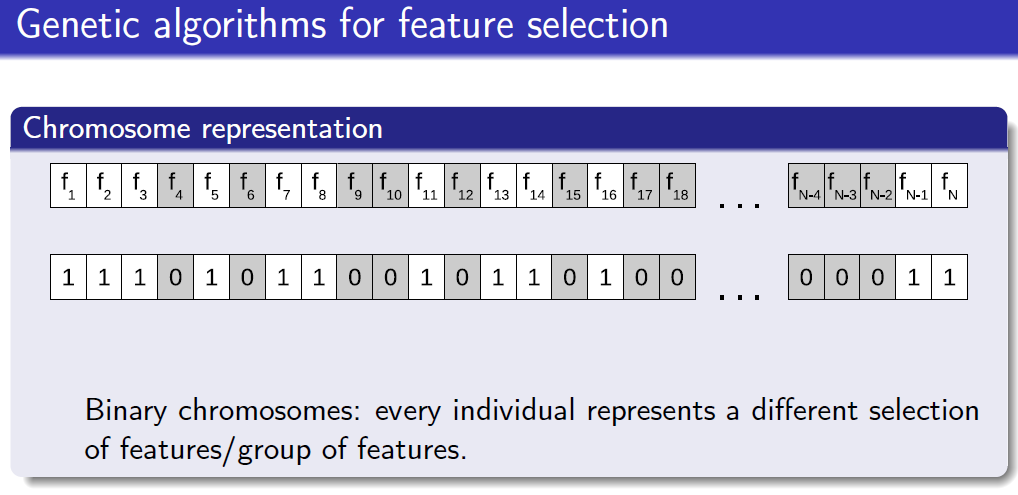
- Repetís esto para todos los individuos

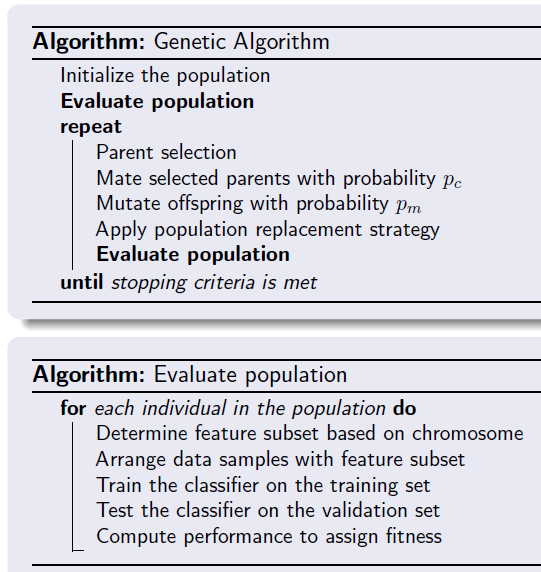
- Y ahí tenés todo lo que necesitas para hacer el algoritmo genético, con el fitness de cada individuo haces selección, cruza, mutación, generas una nueva población, y repetís todo lo anterior, hasta que converja.

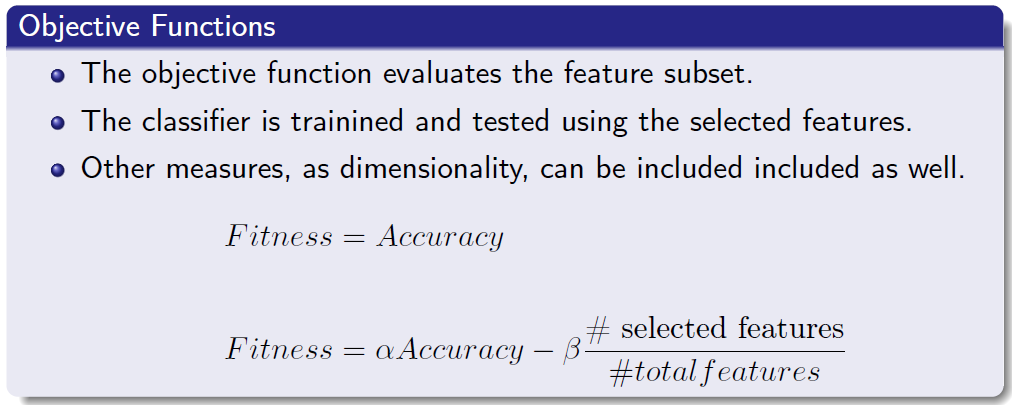
En el **PDF** que subieron los profes (“repasoAG\_introFS” lo tengo en la carpeta) también se explican cosas de este ejercicio a partir de 12 en adelante, y se muestra un algoritmo y ecuaciones que se usan.

Abajo dejo imágenes de pág. 19 en adelante.





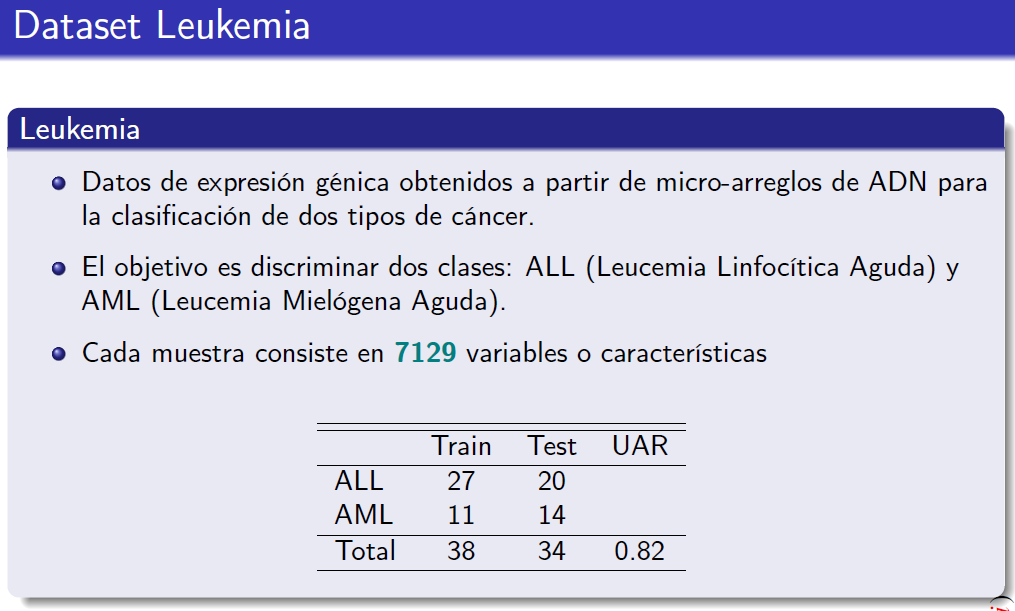




-Mientras más “grande” sea el valor de ‘Accuracy’, mejor.

-Mientras más “chico” sea el valor de ‘total features’, mejor.

Porque así nos dará un fitness más alto.



Sobre lo de “UAR” en la tabla Sofi tenía anotado:

UAR: Referencia. Calcular el accuracy para cada clase, y después calcular el promedio.