Propuesta de trabajo
Inteligencia Artificial (IS) 23/24

Algoritmos Genéticos para resolver problemas de Regresión

Jose M. Moyano

1. Introducción y objetivos

En el amplio campo de la Inteligencia Artificial (IA) y la optimización, los Algoritmos Genéticos (AGs) se presentan como una herramienta de optimización inspirada en la evolución biológica. A diferencia de otros enfoques tradicionales de optimización como la búsqueda local, los AGs se basan en la idea de evolución y selección natural para encontrar soluciones óptimas en espacios de búsqueda complejos y amplios. Esta metodología se inspira en la forma en que la naturaleza selecciona las mejores soluciones a lo largo del tiempo, mediante la combinación, mutación y selección de individuos en poblaciones.

Los AGs operan en un conjunto de soluciones candidatas, representadas como individuos en una población. Estos individuos están codificados usualmente como cadenas binarias, de números enteros, o estructuras más complejas que representan posibles soluciones al problema de optimización en cuestión. A través de un proceso iterativo que simula la evolución biológica, los algoritmos genéticos crean nuevas generaciones de la población de soluciones candidatas, seleccionando los individuos más aptos y combinándolos para producir descendencia mejorada.

Durante el proceso de evolución, los individuos con características más favorables suelen tener una mayor probabilidad de ser seleccionados para reproducirse, mientras que aquellos menos aptos son eliminados o tienen menos probabilidad de contribuir a la descendencia. Además, la introducción de mutaciones aleatorias garantiza la exploración del espacio de búsqueda, permitiendo a los algoritmos genéticos escapar de óptimos locales y buscar soluciones más diversas y potencialmente mejores.

Los AGs se pueden usar para resolver multitud de problemas de optimización. En este trabajo en concreto, se espera **proponer una solución basada en AGs para resolver un problema de regresión**, mediante el modelado de relaciones complejas entre variables capaces de predecir la salida. Estas expresiones permitirían además obtener modelos comprensibles e interpretables que van más allá de la simple predicción

numérica. Estos modelos simbólicos pueden proporcionar información valiosa sobre cómo interactúan las variables en el problema en cuestión, lo que puede ser crucial para la toma de decisiones informadas en diversas áreas.

Los **objetivos específicos** del trabajo serán:

1. Diseñar un AG (representación de soluciones, función de fitness, operadores de cruce y mutación, métodos de selección, ...) capaz de resolver un problema de regresión (con las características descritas en este documento) dado un conjunto de datos. El diseño de cada una de las partes del AG debe estar debidamente justificado en la memoria.
2. Implementar dicho AG. Se deberá realizar en Python, y el código debe estar debidamente documentado.
3. Ejecutar una batería de experimentos sobre ciertos conjuntos de datos proporcionados. La implementación de los experimentos debe realizarse de tal manera que, únicamente proporcionando el nombre del fichero de datos y ciertos parámetros pueda ejecutarse un nuevo experimento, devolviendo la solución obtenida (ver posteriormente la Figura 3 en la Sección 3). Nótese que los **conjuntos de datos serán distintos** en las diferentes **convocatorias**.
4. Documentar el trabajo usando un formato de artículo científico.
5. Realizar una presentación de los resultados obtenidos y defensa del trabajo.

2. Antecedentes

2.1. Regresión

Existen multitud de formas de resolver problemas de regresión, es decir, de ser capaces de predecir una salida dada a partir de un conjunto de atributos de entrada que describen al ejemplo, como los vistos ya en la asignatura.

Uno de los métodos comúnmente utilizados se basa en buscar una combinación de los valores de los atributos de entrada, coeficientes, y ciertas funciones matemáticas, capaces de modelar la salida esperada para cada ejemplo. Para este trabajo, y para simplificar otras propuestas más complejas para modelar el problema, supondremos que buscaremos modelos de regresión dados por una ecuación que relaciona los atributos de entrada $x_1, ..., x_d$ para modelar la salida y tal y como se indica en la Ecuación 1, donde x_i será el valor del i-ésimo atributo de entrada, c_i un coeficiente multiplicando al valor de dicho atributo, e_i un exponente asociado al valor del atributo, y C el valor de una única constante.

$$\hat{y} = c_1 \cdot x_1^{e_1} + \ldots + c_d \cdot x_d^{e_d} + C \tag{1}$$

Suponiendo el conjunto de datos en la Tabla 2.1, donde cada ejemplo está descrito por dos atributos de entrada: temperatura y humedad, y la variable objetivo es la venta de helados, una expresión como la de la Ecuación 2 trataría de ajustar (aunque no perfectamente) el problema de regresión.

Tabla 1: Ejemplo de conjunto de datos de regresión.

Temperatura (T)	**Humedad (H)**	**Venta de helados (V)**
20	55	100
25	50	150
30	45	200
35	40	250

$$\hat{V} = 8.2T - 3H^{0.5} - 32 \tag{2}$$

Dada dicha Ecuación 2 como ejemplo, los parámetros que representan esa solución serían: $c_1 = 8.2$, $e_1 = 1$, $c_2 = -3$, $e_2 = 0.5$, y $C = -32$.

2.2. Algoritmos Genéticos

Los AGs permiten resolver problemas de optimización donde el espacio de búsqueda se estima amplio y complejo [1, 2]. No garantizan la solución óptima, pero sí que suelen ser útiles para encontrar una solución aceptablemente buena en un tiempo asumible. Comparado con otros métodos de optimización, como la búsqueda local, permite evitar facilmente óptimos locales, gracias al característico manejo de poblaciones (conjuntos de soluciones) en lugar de soluciones individuales.

Los primeros ejes principales a la hora de definir un AG son: representación de los individuos, y función de evaluación o *fitness*.

- **Representación.** La representación de los individuos se suele hacer de la forma más simple posible, siempre que permita representar una solución candidata al problema, y permita trabajar posteriormente con los operadores genéticos de la forma más directa y sencilla posible (se detallarán más adelante).

  Supongamos el problema de la mochila, donde tenemos un conjunto de n objetos $o_1, \ldots, o_n$ distintos, disponiendo originalmente de una cantidad $q_1, \ldots, q_n$ para cada uno de ellos, y donde cada objeto tiene asignado un valor $v_i, \ldots, v_n$. El objetivo será buscar una solución donde se indique el número de objetos de cada tipo que se incluyen en la mochila, sin superar un peso máximo k.

Una posible representación para ese problema sería mediante un array de tamaño n, donde cada celda del array representa el número de objetos de cada tipo a incluir en la mochila. Por ejemplo, la solución $[0, 3, 0, 1, 2]$ en un problema con 5 objetos, indicaría que en la mochila incluiríamos 3 unidades del objeto o_2, 1 unidad de o_4, y 2 unidades de o_5.

Hay multitud de formas de representar cada problema, dependiendo de la naturaleza del mismo, siendo por lo general representados mediante un array de: valores binarios, enteros, flotantes, tuplas de valores, etc.

Además de la representación como tal, la implementación que se lleve a cabo del individuo debería tener capacidad de almacenar su valor de aptitud o evaluación de acuerdo a la función de *fitness*, que se detalla a continuación.

- **Función de fitness.** La función de *fitness* será la que indique cuán bueno es un individuo dado, o una solución al problema. Para el ejemplo anterior del problema de la mochila, la función de *fitness* sería el sumatorio de los valores de cada objeto que hay en la mochila. En la solución anterior, el valor de *fitness* del individuo *ind* sería: $f_{ind} = 3 * v_2 + 1 * v_4 + 2 * v_5$, es decir, la suma de los valores obtenidos por cada objeto en la mochila.

  Al tratarse de un problema de maximización, teniendo dos individuos ind_1 e ind_2 cuyos valores de *fitness* sean $f_1 = 40$ y $f2 = 20$, podremos considerar que el primer individuo es mejor que el segundo, ya que tiene un mayor valor de *fitness* o aptitud. Al calcular la función de fitness habría que tener en cuenta también posibles restricciones del problema; es decir, si la mochila superase el peso máximo, habría que asignarle un valor de *fitness* inválido (negativo, 0, ...).

El Algoritmo 1 presenta el esquema general básico de un AG. A continuación se describe en mayor nivel de detalle y con varios ejemplos cada uno de esos pasos:

- **Crear población inicial.** La inicialización de la población (recordemos, la población es un conjunto de individuos, donde cada individuo representa una solución al problema) se suele hacer, por lo general, creando un conjunto de individuos de manera aleatoria (siempre y cuando esos individuos sean válidos para el problema). Para el problema de la mochila, un individuo inicial cualquiera podría ser, por ejemplo $[1, 1, 0, 4, 0]$, suponiendo que existen al menos esa cantidad para cada uno de los objetos indicados.

- **Evaluación de la población.** En este paso, habrá que calcular la función de *fitness* para cada uno de los individuos, y almacenarlo. A partir de este punto, si fuese necesario podriamos ordenar la población por *fitness*, o comparar dos soluciones entre sí.

- **Selección de padres.** La selección de padres permite seleccionar un conjunto de individuos (con o sin repetición) de la población actual para el posterior cruce y obtención de nuevos individuos. Hay diversas formas de realizar la selección de padres para el cruce. Una forma bastante generalizada es seleccionar *nInd*

Algoritmo 1: Esquema general de Algoritmo Genético

Entrada:

- *datos*: Conjunto de datos, en caso de ser necesario
- *nInd*: Número de individuos en la población
- *maxIter*: Número máximo de iteraciones
- *pc*: Probabilidad de cruce
- *pm*: Probabilidad de mutación

Salida: Mejor solución encontrada al problema

```
1  pob ← crearPoblacionInicial(nInd, datos)
2  evaluar(pob)
3  para un número máximo de iteraciones hacer
4  |  padres ← seleccionarPadres(pob)
5  |  hijos ← cruzar(padres, pc)
6  |  hijos ← mutar(hijos, pm)
7  |  evaluar(hijos)
8  |  pob ← seleccionarSiguientePob(pob, hijos)
9  fin
10 devolver mejor(pob)
```

individuos por torneo (aunque pueda ser un número distinto, dependiendo del caso). La selección por torneo se basa en escoger un número k (siendo valores típicos $k = 2$ o $k = 3$, aunque depende del problema) de individuos de la población aleatoriamente, y escoger el mejor de entre esos k para formar parte de la población de padres. Si repetimos ese proceso de torneo *nInd* veces, tendremos una población de *nInd* padres, donde los mejores individuos habrán tenido más probabilidad de seleccionarse (y además pueden aparecer varias veces).

- **Cruzar individuos.** Una vez tenemos la población de padres, y considerando que es un número par, el operador de cruce se encarga de generar nuevos pares de individuos mediante la combinación de pares de padres. La idea que subyace a este operador es el de mezclar o combinar dos individuos, buscando que esa descendencia que contiene material de ambas soluciones pueda ser mejor que los padres (siguiendo la idea de evolución natural mediante la obtención de nueva descendencia). En este proceso entra en juego el valor de probabilidad de cruce: para cada par de individuos, se determina en base a esa probabilidad si se cruzan o no. Si no se cruzan, los individuos pasan tal cual a la población de hijos; si se cruzan, los individuos obtenidos mediante el cruce pasan a la población de hijos. La probabilidad de cruce suele tener valores altos, como 0.7 o 0.8. Por tanto, en torno a un 70-$80\,\%$ de los padres se cruzarán, y el resto se mantendrán tal cual.

  Como se comenta, es importante que los hijos contengan información de ambos padres, siendo una "mezcla" de ambos. Un operador de cruce usado comúnmente es el cruce en un punto: se selecciona un punto de corte aleatorio dentro del

array que representa el individuo, y los nuevos individuos tendrán la información anterior a ese punto de corte de un padre, y la posterior al punto de corte del otro padre. En la Figura 1 se ve un ejemplo del operador de cruce en un punto, donde se crean dos nuevos individuos. Otra opción de operador de cruce (de entre tantas) sería el llamado cruce uniforme, que decide, para cada gen (cada celda del array) si se cruza o no, es decir, para cada posición se decide si el valor del padre 1 va al hijo 1 (y el del padre 2 al hijo 2), o si el valor del padre 1 va al hijo 2 (y el del padre 2 al hijo 1).

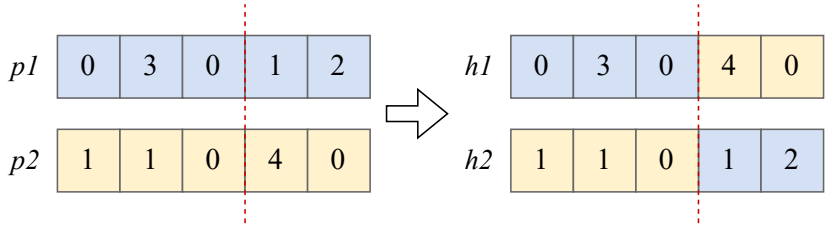

Figura 1: Ejemplo de operador de cruce en un punto.

- **Mutar individuos.** Para cada uno de los individuos en la población de hijos se decide, en base a la probabilidad de mutación (que suele ser pequeña, con valores de 0.1, por ejemplo), si dicho individuo se muta o no; resultando por tanto en que en torno a un $10\,\%$ de los individuos sufren un cambio o mutación en sus valores. La idea de la mutación es la de obtener nuevo material genético en las soluciones, que permitan explorar nuevas zonas del espacio de búsqueda. Dicha variación no suele ser muy disruptiva, aunque depende del problema. Para el ejemplo de la mochila, un posible operador de mutación sería el que se muestra en la Figura 2, donde un gen se selecciona al azar, y se modifica su valor por otro valor factible para dicha posición. Por lo general, el operador de mutación depende en gran medida del problema a tratar y su representación.

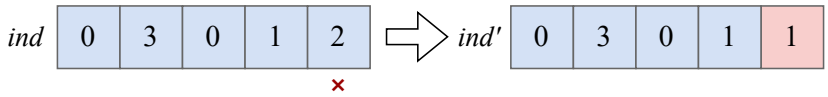

Figura 2: Ejemplo de operador de mutación.

- **Selección de la siguiente generación.** Una vez se tiene la nueva población de hijos (tras cruzar y mutar los padres seleccionados de la población actual), llega el momento de escoger qué individuos se seleccionan para la siguiente generación. Existen también diferentes formas de seleccionar los individuos para la siguiente generación, dependiendo de si se aplica mayor presión selectiva (se indica por presión selectiva la selección fuertemente guiada por los mejores individuos) o se da más peso a una mayor diversidad de la población. A continuación se mencionan dos posibles metodologías para seleccionar dicha nueva población:

- Reemplazo generacional con elitismo: en este caso, se escoge un porcentaje (que será un parámetro a fijar por el equipo de trabajo, pudiendo ser 5, 10, $20\,\%$, ...) de los mejores individuos de la población actual, mientras que el resto de la población vendrá dada directamente por la nueva población de hijos. Como ejemplo, supongamos una población de 10 individuos donde se mantiene un 20 % de elitismo: en ese caso, los 2 mejores individuos de la población anterior pasan directamente a la población de la siguiente generación; y por otro lado, hay que generar 8 individuos mediante cruce y mutación para formar parte de la nueva población (en ese caso, lo general es seleccionar solo 8 padres por tanto, para generar solo el número de individuos deseado).
- Reemplazo por ruleta: en este caso, se consideran en un mismo conjunto todos los individuos de la población actual y la población de hijos. De entre ellos, se seleccionan *nInd* individuos (con o sin repetición) donde, aquellos con un mejor valor de *fitness* tendrán más probabilidad de ser seleccionados[1] (ver Sección 6.1.2 en [2]).

En cualquier caso, el tamaño de la población (el número de individuos en ella) debe mantenerse constante entre distintas generaciones.

3. Descripción del trabajo

Junto con la propuesta de trabajo en PDF, se adjuntan algunos conjuntos de datos (tanto de entrenamiento como de validación) para probar el AG y evaluar la solución obtenida. El conjunto de entrenamiento debe ser el único que se utilice para obtener el modelo de regresión mediante el AG, mientras que el conjunto de validación permitirá al equipo de trabajo evaluar finalmente ese modelo sobre datos nuevos.

Analizando en mayor detalle el primero de los objetivos específicos propuestos en la introducción, el objetivo del diseño del AG cubriría:

- **Diseño de la representación de los individuos**. Debe permitir representar de forma simple soluciones al problema de regresión del tipo visto en la Ecuación 1, para un conjunto de datos con n atributos de entrada.
- **Diseño de la función de *fitness***. La función de *fitness* permite evaluar la calidad de una solución. Dado que las soluciones representan modelos de regresión, sería buena idea evaluar cada una de las soluciones mediante alguna métrica de calidad de modelos de regresión vista en la asignatura.

[1] En problemas de maximización, la probabilidad de seleccionar un individuo será proporcional a su valor de fitness. En problemas de minimización, la probabilidad de seleccionar a cada individuo suele ser proporcional a 1/fitness

- **Diseño de los métodos de selección**, tanto selección de padres como selección de individuos para la siguiente generación. Para la selección de padres, se puede tener en cuenta el método comentado en la sección anterior, o diseñar uno propio. Para la selección de la población para la siguiente generación, en la convocatoria de **junio** se implementará el reemplazo generacional con elitismo (o alguna pequeña variación del mismo), mientras que para las convocatorias de **julio** y **noviembre** se debe implementar el reemplazo por ruleta (o alguna pequeña variación del mismo).

- **Diseño de operadores de cruce y mutación**. Se puede tomar como base los operadores comentados en la sección anterior, aunque depende de la representación de los individuos, es posible que no puedan aplicarse tal cual. Debe diseñar, al menos, un operador de cruce y otro de mutación válidos para la representación escogida. Sin embargo, sería posible implementar más de un operador, realizar un proceso de experimentación y seleccionar el mejor de ellos. En caso de entregar el trabajo en dos convocatorias, se debe diseñar e implementar, al menos, un operador (de cruce o de mutación) extra.

- **Diseño del algoritmo genético**. Se puede tomar como base el AG presentado en el Algoritmo 1, el cual debe funcionar sin problema si el resto de elementos se diseñaron correctamente. Aun así, se deja libertad si se desea modificar la estructura del algoritmo.

Es importante que el algoritmo implementado responda correctamente al código de ejemplo proporcionado en el fichero *prueba_AG.py*, que se muestra también en la Figura 3, donde `AG()` será una clase definida por el grupo/estudiante, que reciba el nombre de los ficheros de datos de entrenamiento y test, y los parámetros específicos para el AG. Además, al ejecutar la función `run()`, devuelva una tupla con la solución encontrada y un array con las predicciones sobre el conjunto de **test**.

4. Entrega y defensa

La documentación del trabajo se realizará en formato de artículo científico, cuya plantilla se puede encontrar en la web de la asignatura. Se valorará positivamente el uso de LaTeX frente a otros sistemas. En cualquier caso, se entregará la documentación en formato PDF.

La longitud mínima de la memoria será de 6 páginas, e incluirá, al menos: 1) Introducción, 2) Descripción y diseño de los elementos clave del AG, 3) Resultados obtenidos en la experimentación, 4) Conclusiones, y 5) Bibliografía. Por otro lado, se entregará el código implementado (ya sean ficheros *.py* o cuadernos de Jupyter), incluyendo además, el fichero de Python de la Figura 3 con las modificaciones necesarias para que se pueda ejecutar (modificaciones tales como importar el resto de ficheros implementados, etc.). Todo ello se entregará en un único fichero comprimido

```
ag = AG(
    # datos de entrenamiento (para el proceso del AG)
    datos_train = "fichero_train.csv",
    # datos de validacion/test (para predecir)
    datos_test = "fichero_test.csv",
    # semilla para numeros aleatorios
    seed=123,
    # numero de individuos
    nInd = 50,
    # maximo de iteraciones
    maxIter = 100
)

ag.run()
# (<ind...>, [0.18, 1.24, ..., -1.13])
```

Figura 3: Código que debe ejecutar la propuesta del trabajo.

zip, llamado necesariamente:
AG_<*ApellidosEstudiante1*>_<*ApellidosEstudiante2*>.zip;
por ejemplo, en caso de hacer el trabajo entre los alumnos Jose María Moyano Murillo y Álvaro Romero Jiménez, el fichero comprimido debería llamarse:
AG_MoyanoMurillo_RomeroJimenez.zip.
En caso de realizarse por un único estudiante en convocatorias extraordinarias, solo llevará sus apellidos.

Como parte de la evaluación del trabajo, se realizará una defensa del mismo, para lo que se citará en su momento al grupo de trabajo. Se realizará una defensa con una pequeña presentación de 10 minutos de duración, en la que deben participar activamente todos los miembros del grupo. La defensa debe seguir a grandes rasgos la estructura de la memoria, haciendo especial hincapié en el diseño de los distintos aspectos del AG y su justificación. Posteriormente a la defensa, el profesor realizará también ciertas preguntas sobre la totalidad del trabajo.

5. Evaluación del trabajo

Para que el trabajo sea evaluado, es indispensable que el código en la Figura 3 funcione correctamente, y devuelva unas predicciones al conjunto de datos de test dado.

Para la evaluación del trabajo se tendrán en cuenta los siguientes criterios, con hasta un máximo de 3 puntos:

- Diseño y justificación de los distintos aspectos del AG (hasta 1 punto): se valorará el correcto diseño de la representación de los individuos, función de fitness, y operadores genéticos, entre otros aspectos del AG, así como que la justificación de los mismos sea adecuada.

- Memoria del trabajo (hasta 1 punto): se valorará la presentación, correcto uso del lenguaje, claridad en las explicaciones, y presentación de resultados, entre otros. Se valorará también, si se realiza, el proceso de experimentación llevado a cabo para encontrar la mejor configuración. Por ejemplo, si se prueban distintos valores para las probabilidades de cruce y mutación, para el valor de k en el torneo, o distintos operadores genéticos, y se selecciona finalmente uno de ellos, debe quedar reflejado qué experimentos se han llevado a cabo, qué resultados se han obtenido, y finalmente qué configuración se ha escogido. La configuración final del AG debe ser la misma para todos los conjuntos de datos. En cualquier caso, la elaboración de la memoria debe ser original, por lo que no se evaluará el trabajo si se detecta cualquier copia de contenido.

- Código fuente (hasta 0.5 puntos): se valorará la claridad y buen estilo de programación[2], corrección y eficiencia en la implementación, calidad de los comentarios, así como el reflejo del diseño de los distintos aspectos del AG en el código. En cualquier caso, el código debe ser original, por lo que no se evaluará el trabajo si se detecta código copiado o descargado de internet.

- Resultados obtenidos (hasta 0.5 puntos): se valorarán los resultados obtenidos en distintos conjuntos de datos. Para ello, se evaluará la solución obtenida por el AG no solo en nuevas particiones de test de los mismos conjuntos de datos proporcionados, sino también sobre otros conjuntos de datos nuevos. Para la evaluación de los resultados se tendrá en cuenta tanto el RMSE y el coeficiente R^2 sobre el conjunto de test, como el tiempo requerido por el AG para obtener dicha solución. En todos los casos, se ejecutarán los algoritmos con un número de individuos e iteraciones prefijado por el profesor; el resto de parámetros serán los que haya decidido el equipo de trabajo y que se incluyan como valores por defecto al ejecutarlo.

- Presentación y defensa (hasta 1 punto): se valorará la claridad de la presentación, la buena explicación de los contenidos del trabajo, y las respuestas a las preguntas realizadas por el profesor.

IMPORTANTE: cualquier plagio, compartición de código o uso de material que no sea original y del que no se cite convenientemente la fuente, significará automáticamente la calificación de cero en la asignatura para todos los alumnos involucrados. Por tanto, a estos alumnos no se les conserva, ni para la actual ni para futuras convocatorias, ninguna nota que hubiesen obtenido hasta el momento. Todo ello sin perjuicio de las correspondientes medidas disciplinarias que se pudieran tomar.

[2] https://docs.python-guide.org/writing/style/

Referencias

[1] Abdelmalik Moujahid, Iñaki Inza, and Pedro Larrañaga. Algoritmos genéticos. http://www.sc.ehu.es/ccwbayes/docencia/mmcc/docs/t2geneticos.

[2] Marcos Gestal Pose. Introducción a los algoritmos genéticos. http://sabia.tic.udc.es/mgestal/cv/AAGGtutorial/aagg.html.