Universidad Simón Bolívar

Departamento de Computación y T.I.

Inteligencia Artificial II

Septiembre-Diciembre 2014

Estudiantes: Carnets:

Wilmer Bandres 10-10055

Juan Escalante 10-10227

**PROYECTO 2: GABIL**

* **Resumen**

Para esta entrega se implementó un algoritmo genético simple utilizando el sistema de Gabil. Se utilizaron cuatro funciones distintas de selección para sobrevivientes/padres que fueron: ruleta, ranking, torneo y uniforme.

Se utilizó un framework para python hecho en python que se llama pyevolve y que ya tiene implementada la mayoría de las funciones necesarias para el algoritmo genético. Sin embargo, se implementó ciertas funciones que son las especificadas en el sistema de Gabil ya que estas no vienen en pyevolve, así como también se ajustó el framework para utilizar la representación binaria para la clasificación de las Iris.

Además se implementaron dos funciones de fitness, que se diferencian porque una penaliza a los individuos que tienen un tamaño relativamente grande, mientras que la otra no.

* **Descripción de la implementación**

Se implementó únicamente todos los operadores de Gabil ya que las demás funciones del mismo algoritmo genético ya vienen implementadas en pyevolve.

Como operador de crossover se implementó la función descrita por el libro en el capítulo 9 que es un crossover de dos puntos con la variación de que se agarran dos puntos que están a cierta distancia de la regla más cercana y que debe ser la misma distancia para ambos padres y luego se procede a hacer la creación de los hijos de la misma forma que un crossover normal de dos puntos.

Como operador de mutación se implementó la mutación en un solo punto.

Todas las funciones de selección ya vienen implementadas en el framework que son la selección por ruleta, la selección por ranking, la selección uniforme y la selección por torneo. En el caso de pyevolve, no existe una forma de agregar selección de padres en particular pero sí para sobrevivientes, por lo tanto no existe una función de selección de padres como tal en esta entrega.

En cuanto a la representación de los individuos, cada individuo está representado por conjunto de reglas de 13 bits, es decir que el tamaño del individuo es un múltiplo de 13. Y cada regla significa lo siguiente:

* Los bits del 1 al 3 indican a qué clases podría pertenecer la primera característica de un ejemplo para que esta regla se cumpla y si ocurre que sus otras características también pertenecen a clases que acepte esta regla, podrá ser clasificada por la misma. La primera característica de cada ejemplo puede estar en la clase 1 (valores de -infinito a 5.6 sin incluir), en la clase 2 (valores de 5.6 a 6.6 sin incluir) o en la clase 3 (valores de 6.6 a infinito).
* Los bits del 4 al 5 indican a qué clases podría pertenecer la segunda característica de un ejemplo. La segunda característica de cada ejemplo puede estar en la clase 1 (valores de -infinito a 3.0 sin incluir) o en la clase 2 (valores de 3.0 a infinito).
* Los bits del 6 al 8 indican a qué clases podría pertenecer la tercera característica de un ejemplo. La tercera característica de cada ejemplo puede estar en la clase 1 (valores de -infinito a 2.5 sin incluir), en la clase 2 (valores de 2.5 a 5.0 sin incluir) o en la clase 3 (valores de 5.0 a infinito).
* Los bits del 9 al 11 indican a qué clases podría pertenecer la cuarta característica de un ejemplo. Puede estar en la clase 1 (valores de -infinito a 1.0 sin incluir), en la clase 2 (valores de 1.0 a 2.0 sin incluir) o en la clase 3 (valores de 2.0 a infinito).
* Los bits del 12 al 13 indican que si un ejemplo cumple con que sus características están en las clases que acepta esta regla, entonces será clasificada como: 01 (setosa), 10 (versicolor), 11 (virginica).
* **Descripción del AG**

El algoritmo que viene implementado en pyevolve es el algoritmo genético simple que va de generación en generación aplicando la operaciones de crossover y mutación y que además va eligiendo los sobrevivientes para la siguiente generación según las funciones de selección explicadas anteriormente.

* **Descripción y análisis de los experimentos realizados**

Todos los experimentos se realizaron con la función de selección de sobrevivientes por torneo ya que tuvo los mejores resultados. Las corridas se hicieron con tasas de crossover de 90%, 80% y 65%, combinados con una tasa de mutación de 1%, 1.5% y 0.5%, es decir, hay 9 posibles combinaciones y para cada combinación se realizaron 10 corridas y se promedió la cantidad de Iris bien clasificadas. A continuación los resultados:

|  |  |  |
| --- | --- | --- |
| Tasa de crossover | Tasa de Mutación | Promedio de las 10 corridas de la cantidad de ejemplos bien clasificados (sobre un total de 150 ejemplos) |
| 90 | 1 | 128.5 |
| 90 | 1.5 | 125.0 |
| 90 | 0.5 | 125.0 |
| 80 | 1 | 133.5 |
| 80 | 1.5 | 141.6 |
| 80 | 0.5 | 133.5 |
| 65 | 1 | 128.7 |
| 65 | 1.5 | 132.7 |
| 65 | 0.5 | 119.7 |

Las corridas en su totalidad están ubicadas en un archivo llamado "reporte".

Cada una de estas corridas se realizó con la función de fitness que penaliza los tamaños ya que el tiempo de ejecución era mucho menor y además el promedio de ejemplos bien clasificados no variaba mucho.

Como se observa en la tabla, la mejor combinación o configuración de parámetros fue la que tiene tasa de crossover de 80% y tasa de mutación de 1.5% con un promedio de ejemplos bien clasificados de 141.6 sobre un total de 150 ejemplos. Además durante la experimentación se observó que aquellas corridas que tenían una tasa de crossover de 80% tendían a converger más rápido, hecho que puede estar influenciado por el buen balance de escoger una gran cantidad de individuos con un fitness alto para una siguiente generación así como también por el alto porcentaje de nuevos individuos creados (esto último es lo que hace falta en las corridas con tasa de crossover de 90% para poder mejorar).

Por otro lado, las peores corridas son las que tienen una tasa de crossover de 65% como consecuencia de que cada generación varía demasiado.

El mejor conjunto de reglas hallado por el algoritmo genético fue el individuo:

1111110010101 1111111011010 1111111111011

que logró clasificar un total de 142 ejemplos correctamente. Esta compuesto por tres reglas que se traducen a:

* 111-11-100-101-01: No importa cual sea el valor de la primera y la segunda característica del ejemplo, si el ejemplo posee el valor 1 en su tercera característica y el valor 1 o 3 en su cuarta característica, entonces ese ejemplo es una Iris setosa.
* 111-11-110-110-10: No importa cual sea el valor de la primera y la segunda característica del ejemplo, si el ejemplo posee el valor 1 o 2 en su tercera característica y el valor 1 o 2 en su cuarta característica, entonces ese ejemplo es una Iris versicolor.
* 111-11-111-110-11: No importa cual sea el valor de la primera, la segunda y la tercera característica del ejemplo, si el ejemplo posee el valor 1 o 2 en su cuarta característica, entonces ese ejemplo es una Iris virgínica.
* **Función de fitness utilizada**

Las funciones de fitness implementadas son:

* La del sistema de Gabil: El número de aciertos (ejemplos bien clasificados) al cuadrado.
* Una función que penaliza el tamaño de los clasificadores, que simplemente es la misma función de gabil dividida por el tamaño del clasificador lo que trae como consecuencia que el tamaño de los clasificadores se mantenga bajo.

Incluir una penalización en la función de fitness es útil por dos razones fundamentales:

* Dado a que ayuda a mantener la cantidad de reglas relativamente baja, generaliza un poco más que cuando se tiene una cantidad grande de reglas.
* Como la cantidad de reglas se mantiene relativamente baja, el tiempo que tarda en ejecutar el algoritmo es menor ya que el conjunto de datos a manejar (la representación de cada individuo) se reduce considerablemente.

Además la diferencia de porcentaje de clasificados correctamente no difiere considerablemente entre corridas con ambas funciones de fitness por lo que vale la pena utilizar la segunda.

Sin embargo, en otro tipo de problemas donde la cantidad de reglas que debe tener un individuo para poder clasificar, o separar bien los ejemplos, es alta, posiblemente sea mejor usar la primera función de fitness.

En el caso de este proyecto, un buen clasificador de Iris no requiere de muchas reglas por lo que la segunda función de fitness fue ideal.