Práctica 3: Algoritmos genéticos

Antonio Álvarez Caballero 15457968-J
5º Doble Grado Ingeniería Informática y Matemáticas
Grupo de prácticas del Viernes 17:30-19:30
analca3@correo.ugr.es

11 de mayo de 2016

Índice

1. Descripción del problema

El problema a resolver es el problema de *Selección de características*.En el ámbito de la *Ciencia de Datos*, la cantidad de datos a evaluar para obtener buenos resultados es excesivamente grande. Esto nos lleva a la siguiente cuestión: ¿Son todos ellos realmente importantes? ¿Podemos establecer dependencias para eliminar los que no nos aportan información relevante? La respuesta es que sí: en muchas ocasiones, no todos los datos son importantes, o no lo son demasiado. Por ello, se intentará filtrar las características relevantes de un conjunto de datos.

La selección de características tiene varias ventajas: se reduce la complejidad del problema, disminuyendo el tiempo de ejecución. También se aumenta la capacidad de generalización puesto que tenemos menos variables que tener en cuenta, además de conseguir resultados más simples y fáciles de entender e interpretar.

Para conseguir este propósito se deben usar técnicas probabilísticas, ya que es un problema *NP-hard*. Una técnica exhaustiva sería totalmente inviable para cualquier caso de búsqueda medianamente grande. Usaremos metaheurísticas para resolver este problema, aunque también podríamos intentar resolverlo utilizando estadísticos (correlación entre características, medidas de separabilidad o basadas en teoría de información o consistencia, etc).

2. Descripción de aplicación de los algoritmos al problema

Los elementos comunes de los algoritmos son:

- Representación de las soluciones: Se representan las soluciones como vectores 1-dimensionales binarios (los llamaremos *bits* para poder hacer uso de términos como *darle la vuelta a un bit*):

$$s = (x_1, x_2, \ldots, x_{n-1}, x_n);\ x_i \in \{True, False\}\ \forall i \in \{1, 2, \ldots, n\}$$

- Función objetivo: La función a maximizar es la tasa de clasificación de los datos de entrada:

$$tasa\_clas = 100 \cdot \frac{instancias\ bien\ clasificadas}{instancias\ totales}$$

- Generación de vecino: La función generadora de vecinos es bien simple. Se toma una solución y se le da la vuelta a uno de sus bits, el cual se escoge aleatoriamente.

```
Tomar un vector de características "caracteristica"
indice = generarAleatorio(0, numero_caracteristicas)
caracteristicas[indice] = not caracteristicas[indice]
```

3. Descripción de la estructura del método de búsqueda

Ambos algoritmos se han implementado con el mismo código. Sólo varía un booleano en la llamada a la función, que altera el número de cromosomas cogidos para la selección y en la presencia del elitismo.

- Algoritmo genético:

```
tamaño_cromosoma = número de características
evaluaciones = 0
max_evaluaciones = 15000
```

```
probabilidad_cruce = 0.7
probabilidad_mutacion = 0.001

tamaño_poblacion = 30

Si es generacional, numero_seleccionados = tamaño_poblacion
si no, numero_seleccionados = 2

numero_cruces = EnteroPorArriba(probabilidad_cruce *
                  numero_seleccionados / 2)

# Esta probabilidad es la probabilidad de que un cromosoma mute.
# Esto lo hacemos para evitar trabajar a nivel de gen y generar el
# mínimo número de aleatorios posible, además de homogeneizar
# la implementación de ambas variantes del algoritmo
probabilidad_total_mutacion = probabilidad_mutacion *
        numero_seleccionadas * tamaño_cromosoma

# Inicializar la población y evaluarla
poblacion = Generar "tamaño_poblacion" cromosomas aleatorios
evaluar(poblacion)
# Ordenamos la población por tasa en orden creciente
ordenar(poblacion)

Mientras evaluaciones < max_evaluaciones
  # Cogemos la solución que está en último lugar,
  # ya que la población está ordenada
  mejor_solucion = poblacion[-1]

  # Selección
  Para cada i en [1,2,...,numero_seleccionados]
    eleccion = Coger 2 individuos aleatorios de la población
    ordenar(poblacion)
    mejor = eleccion[-1]
    seleccionados.añadir(mejor)

  # Cruce
  Para cada i en [1,2,...numero_cruces] tomados de dos en dos
    cruce(seleccionados[i], seleccionados[i+1])

  # Mutación
  Si aleatorio() < probabilidad_mutacion_completa
    cromosoma_mutado = aleatorio()
    gen_mutado = aleatorio()

    flip(cromosoma_mutado, gen_mutado)

  # Actualizar tasas
  evaluar(seleccionados)
  evaluaciones = evaluaciones + numero_seleccionados
```

```
    # Reemplazo
    Para cada i en [1,2,..., numero_seleccionados]
      # Reemplazamos los peores de la población por los mejores seleccionados
      poblacion[i] = seleccionados[-i]
    Si es generacional y mejor_solucion[tasa] > poblacion[0][tasa]
      # Sustituimos la peor solución de la población por la mejor que había
      # Elitismo
      poblacion[0] = mejor_solucion

  ordenar(poblacion)
  mejor_solucion, mejor_tasa = poblacion[-1][cromosoma], poblacion[-1][tasa]
  return mejor_solucion
```

Debemos detallar más el operador de cruce (el de mutación es claro). Se han implementado dos operadores: el clásico en dos puntos y el uniforme.

El de cruce en dos puntos no es más que tomar dos puntos aleatorios del cromosoma e intercambiar las regiones que quedan dentro de ambos.

```
# Tomamos los puntos de cruce en el interior
# En el interior es sin tomar los extremos,
# para evitar que sea un cruce en un punto.
puntos_cruce = tomar dos aleatorios entre 1 y tamaño-1
intercambiaCentros(padre, madre)
```

El otro operador es el cruce uniforme. Para cada gen, se intercambian los genes de los padres con probabilidad $\frac{1}{2}$.

```
Para cada gen
  Si aleatorio() < 0.5
    Intercambia(padre[gen], madre[gen])
```

En la tabla de resultados aparecen ambos operadores.

4. Descripción del algoritmo de comparación

El algoritmo de comparación es un algoritmo greedy: el *Sequential Forward Selection(SFS)*. La idea es muy simple: se parte del conjunto vacío de características (todos los bits a 0) y se recorren todas las características, evaluando la función de coste. La característica que más mejora ofrezca, se coje. Y se vuelve a empezar. Así hasta que ninguna de las características mejore el coste.

```
caracteristicas_seleccionadas = [False, False, ..., False]
mejor_caracteristica = 0
mejor_tasa = 0

Mientras mejor_caracteristica != -1
  mejor_caracteristica = -1
```

```
caracteristicas_disponibles = índices donde caracteristicas_seleccionadas vale False
Para cada característica c de caracteristicas_disponibles
  tasa = coste al añadir c a las características seleccionadas
  Si tasa > mejor_tasa
    mejor_tasa = tasa
    mejor_caracteristica = caracteristica
Si mejor_caracteristica != -1
  caracteristicas_seleccionadas.añadir(mejor_caracteristica)
```

5. Desarrollo de la práctica

En primer lugar, comentar que las bases de datos han sido modificadas en su estructura (que no en sus datos) para que sean homogéneas. Así, se han puesto todas las clases como numéricas (en Wdbc no lo estaban) y se han colocado en la última columna.

La práctica se ha desarrollado usando el lenguage de programación *Python*, ya que su velocidad de desarrollo es bastante alta. Para intentar lidiar con la lentitud que puede suponer usar un lenguaje interpretado, utilizaremos las librerías *NumPy, SciPy y Scikit-Learn*, que tienen módulos implementados en C (sobre todo *NumPy*) y agilizan bastante los cálculos y el manejo de vectores grandes. Para el KNN con Leave One Out se ha utilizado un módulo que ha desarrollado mi compañero Alejandro García Montoro[1], que usa *CUDA* para agilizar los cálculos usando la GPU.

Usaremos alguna funcionalidad directa de estas bibliotecas:

- *NumPy*: Generación de números aleatorios y operaciones rápidas sobre vectores.
- *SciPy*: Lectura de ficheros ARFF de WEKA.
- *Scikit-Learn*: Particionamiento de los datos, se han usado las particiones estratificadas de la validación cruzada 5x2.
- *ScorerGPU*: Para el KNN con Leave One Out.

Esta elección se ha hecho para poder preocuparme sólo y exclusivamente de la implementación de las metaheurísticas.

Los requisitos para ejecutar mis prácticas son *Python3* (importante que sea la 3), *NumPy*, *SciPy*, *Scikit-Learn* y *CUDA*, por lo que es necesario una gráfica nVidia. En mi plataforma (Archlinux) están disponibles desde su gestor de paquetes.

Una vez instalados los paquetes, sólo hay que ejecutar la práctica diciéndole al programa los algoritmos que queremos ejecutar. La semilla aleatoria está fijada dentro del código como 12345678 para no inducir a errores. Veamos algunos ejemplos de llamadas a la práctica. Primero notamos que los algoritmos disponibles son:

- -SFS: Ejecuta el algoritmo greedy SFS.
- -LS: Ejecuta la Local Search.
- -SA: Ejecuta el Simulated Annealing.
- -TS: Ejecuta la Tabu Search.
- -TSext: Ejecuta la Tabu Search extendida.
- -BMB: Ejecuta la Búsqueda Multiarranque Básica.

[1] https://github.com/agarciamontoro/metaheuristics

- -GRASP: Ejecuta el GRASP.
- -ILS: Ejecuta la Iterated Local Search.
- -EGA: Ejecuta el genético estacionario.
- -GGA: Ejecuta el genético generacional.

```
$ python featureSelection.py -TS
```

Se ejecutará la Tabu Search. Pero no sólo se limita el programa a un algoritmo. Si le pasamos varios, los ejecutará en serie uno detrás de otro. Esto ha cambiado desde la práctica anterior por la entrada de *CUDA*, que hay que iniciarlo debidamente y no es tan sencillo de ejecutar cosas en paralelo.

```
$ python featureSelection.py -EGA -GGA
```

Se ejecutarán EGA y GGA en serie.

Una vez ejecutado, irán saliendo por pantalla mensajes de este tipo, que proporcionan datos en tiempo real del estado de la ejecución:

```
INFO:__main__:W - TS - Time elapsed: 2265.526112794876.
Score: 98.2394337654. Score out: 95.0877192982 Selected features: 15
```

Este mensaje nos dice todo lo necesario: W es la base de datos (Wdbc), TS el algoritmo, el tiempo transcurrido para esta iteración (recordemos que hay 10), el score de entrenamiento, el score de validación y las características seleccionadas.

6. Experimentos

Como se ha comentado antes, la semilla está fija a 12345678 para no tener problemas de aleatoriedad. El número de evaluaciones máxima de todos los algoritmos es de 15000. Por lo demás, todos los demás parámetros propios de cada algoritmo están tal y como se explica en el guión (25 número de búsquedas en BMB e ILS, 10 % de mutación en ILS, etc).

KNN

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	96.12676	96.84211	0.0	0.0	66.66667	65.0	0.0	0.0	62.5	66.49485	0.0	0.0
Partición 1-2	96.49123	96.83099	0.0	0.0	65.55556	80.55556	0.0	0.0	61.85567	62.5	0.0	0.0
Partición 2-1	96.12676	96.49123	0.0	0.0	68.88889	74.44444	0.0	0.0	64.0625	64.43299	0.0	0.0
Partición 2-2	95.78947	96.12676	0.0	0.0	75.55556	68.88889	0.0	0.0	61.34021	64.58333	0.0	0.0
Partición 3-1	96.47887	95.4386	0.0	0.0	75.55556	71.66667	0.0	0.0	63.02083	63.91753	0.0	0.0
Partición 3-2	96.84211	96.83099	0.0	0.0	68.88889	65.55556	0.0	0.0	62.37113	64.58333	0.0	0.0
Partición 4-1	97.53521	96.14035	0.0	0.0	66.66667	66.11111	0.0	0.0	65.625	64.43299	0.0	0.0
Partición 4-2	93.33333	97.88732	0.0	0.0	72.77778	72.77778	0.0	0.0	60.30928	64.58333	0.0	0.0
Partición 5-1	96.47887	96.84211	0.0	0.0	75.0	71.11111	0.0	0.0	65.625	65.97938	0.0	0.0
Partición 5-2	97.89474	95.42254	0.0	0.0	70.0	70.0	0.0	0.0	61.85567	63.54167	0.0	0.0
Media	96.30974	96.48530	0.0	0.0	70.55556	70.61111	0.0	0.0	62.85653	64.50494	0.0	0.0

SFS

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	97.53521	92.2807	83.33333	0.15793	72.22222	66.66667	93.33333	0.50295	80.20833	70.61856	96.76259	2.41429
Partición 1-2	96.84211	94.01408	86.66667	0.12852	77.22222	66.66667	87.77778	0.91927	73.19588	67.1875	97.48201	1.84876
Partición 2-1	95.42254	91.22807	83.33333	0.15797	84.44444	68.88889	85.55556	1.0978	79.16667	69.58763	97.48201	1.83412
Partición 2-2	97.54386	92.60563	86.66667	0.12911	77.77778	61.66667	93.33333	0.49637	74.2268	64.58333	97.48201	1.88374
Partición 3-1	96.12676	92.98246	90.0	0.09982	83.33333	71.66667	87.77778	0.92519	76.5625	68.5567	98.20144	1.30598
Partición 3-2	97.54386	96.47887	86.66667	0.12927	72.22222	60.0	93.33333	0.50101	71.64948	66.14583	98.20144	1.33008
Partición 4-1	98.23944	96.49123	86.66667	0.12883	70.55556	62.22222	91.11111	0.66223	76.04167	69.58763	97.84173	1.55438
Partición 4-2	94.73684	94.3662	90.0	0.10095	80.55556	65.55556	90.0	0.74934	82.98969	73.95833	97.1223	2.16781
Partición 5-1	94.71831	91.92982	90.0	0.09997	78.88889	66.11111	92.22222	0.57865	77.08333	68.04124	97.48201	1.85395
Partición 5-2	98.94737	93.66197	76.66667	0.21516	76.66667	66.66667	90.0	0.74815	82.98969	71.35417	96.40288	2.75023
Media	96.76563	93.60390	86.00000	0.13475	77.38889	65.61111	90.44444	0.71810	77.41140	68.96209	97.44604	1.89433

EGA

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	97.88732	96.84211	46.66667	25.85457	70.55556	69.44444	53.33333	32.88918	66.14584	64.94845	46.40288	138.21688
Partición 1-2	97.54386	95.42254	53.33333	26.1036	70.0	82.22222	54.44444	33.26382	64.94846	64.0625	52.51799	110.26249
Partición 2-1	97.88732	93.68421	40.0	25.41497	68.88889	67.77778	61.11111	32.50879	67.70834	60.82474	51.07914	133.59249
Partición 2-2	97.19299	95.07042	56.66667	24.91522	73.33334	74.44444	50.0	33.14175	65.46392	64.58333	48.56115	110.40672
Partición 3-1	96.47887	96.14035	46.66667	24.71563	72.77778	73.33333	56.66667	32.22729	66.66666	62.8866	52.8777	133.46811
Partición 3-2	98.24561	95.77465	50.0	25.85523	70.55556	70.0	56.66667	32.61259	64.94846	66.66667	52.51799	112.73535
Partición 4-1	97.53521	95.78947	46.66667	25.12814	72.77778	70.55556	52.22222	32.67926	65.625	67.52577	52.15827	135.97518
Partición 4-2	95.08772	94.3662	40.0	25.07912	73.33334	75.0	45.55556	32.62447	67.01031	59.89583	51.07914	110.88663
Partición 5-1	97.53521	94.73684	50.0	25.09109	74.44444	72.22222	47.77778	33.21744	66.66666	63.91753	55.39568	132.59312
Partición 5-2	97.19299	95.42254	40.0	25.64409	71.66666	79.44444	54.44444	32.45579	65.97938	65.10417	50.71942	109.71494
Media	97.25871	95.32493	47.00000	25.38017	71.83334	73.44444	53.22222	32.76204	66.11630	64.04156	51.33094	122.78519

UXEGA

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	97.88732	96.84211	46.66667	25.81546	70.55556	69.44444	53.33333	33.03504	66.14584	64.94845	46.40288	138.25126
Partición 1-2	97.19299	92.95775	50.0	24.87192	70.55556	85.55556	44.44444	32.94226	65.97938	66.66667	56.47482	108.40969
Partición 2-1	97.1831	94.38596	50.0	25.64479	77.22222	68.33333	48.88889	32.87196	67.70834	61.85567	54.31655	132.27767
Partición 2-2	98.59649	92.95775	53.33333	25.26133	74.44444	66.11111	53.33333	32.97657	65.46392	64.58333	47.48201	108.63862
Partición 3-1	97.53521	92.98246	33.33333	24.61111	75.0	74.44444	61.11111	31.99638	67.70834	58.76289	54.31655	137.30787
Partición 3-2	96.49123	97.88732	50.0	25.89185	75.55556	70.55556	52.22222	32.43681	64.94846	67.70833	53.23741	107.38349
Partición 4-1	96.12676	96.14035	56.66667	26.06434	75.0	70.55556	46.66667	33.30501	66.14584	65.97938	46.40288	132.53498
Partición 4-2	97.89474	93.66197	53.33333	25.49768	71.66666	74.44444	47.77778	33.20667	66.49484	61.97917	55.7554	109.89033
Partición 5-1	95.77465	95.78947	46.66667	24.7668	70.0	72.77778	42.22222	32.44332	64.58334	61.34021	52.15827	132.00001
Partición 5-2	97.89474	95.42254	53.33333	25.1974	76.11111	73.88889	50.0	33.04486	68.5567	65.10417	51.07914	107.38715
Media	97.25772	94.90277	49.33333	25.36227	73.61111	72.61111	50.00000	32.82589	66.37350	63.89283	51.76259	121.40811

GGA

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	98.59155	97.19298	30.0	28.83669	74.44444	68.88889	42.22222	34.59718	71.35416	63.40206	35.61151	166.04058
Partición 1-2	97.89474	94.71831	43.33333	25.61547	75.0	83.88889	53.33333	30.46199	70.1031	64.58333	35.61151	136.57915
Partición 2-1	99.29578	95.78947	13.33333	31.92366	80.0	74.44444	40.0	36.10118	72.91666	65.97938	38.1295	159.50125
Partición 2-2	97.54386	95.42254	30.0	27.91651	76.66666	69.44444	40.0	34.41523	66.49484	63.02083	28.05755	154.27823
Partición 3-1	98.23943	94.03509	23.33333	29.77402	77.22222	76.11111	47.77778	33.23709	69.79166	61.85567	34.17266	160.12333
Partición 3-2	97.54386	96.47887	36.66667	27.30309	74.44444	76.66667	37.77778	36.9158	70.61855	63.54167	38.48921	126.20965
Partición 4-1	98.59155	96.49123	26.66667	29.28416	74.44444	77.22222	36.66667	37.32392	66.66666	64.94845	28.41727	169.19619
Partición 4-2	97.89474	93.30986	16.66667	31.61072	77.22222	77.22222	37.77778	36.73804	71.13402	64.58333	40.28777	112.31727
Partición 5-1	98.59155	95.78947	20.0	30.07224	75.55556	68.88889	44.44444	34.33466	71.875	59.79381	38.1295	150.0746
Partición 5-2	97.54386	96.12676	26.66667	29.15791	73.88889	75.0	43.33333	34.69679	70.1031	64.58333	33.09353	137.37018
Media	98.17309	95.53546	26.66667	29.14945	75.88889	74.77778	42.33333	34.88219	70.10577	63.62919	35.00000	147.16904

UXGGA

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
Partición 1-1	98.23943	96.49123	20.0	30.30276	75.0	67.77778	47.77778	32.89708	72.39584	64.94845	30.21583	173.09707
Partición 1-2	97.89474	96.12676	30.0	28.91638	76.11111	82.22222	40.0	34.97655	72.16495	65.625	34.17266	139.76554
Partición 2-1	97.88732	96.49123	36.66667	26.85122	78.88889	69.44444	43.33333	34.86884	78.125	60.82474	29.13669	169.13744
Partición 2-2	98.59649	96.47887	30.0	28.821	75.0	66.66667	44.44444	34.17999	70.61855	66.14583	31.65468	145.12878
Partición 3-1	98.23943	95.08772	36.66667	27.50646	79.44444	76.11111	43.33333	34.04839	72.91666	61.34021	28.05755	168.97172
Partición 3-2	97.54386	95.77465	13.33333	32.1256	78.88889	77.77778	48.88889	32.21731	72.68041	64.58333	33.45324	138.67895
Partición 4-1	97.88732	97.54386	23.33333	30.18058	75.0	75.0	45.55556	33.56199	73.95834	64.43299	31.29496	164.1239
Partición 4-2	97.89474	96.47887	26.66667	29.17963	80.55556	68.33333	43.33333	34.29262	75.25773	64.58333	33.09353	142.89496
Partición 5-1	98.23943	94.73684	40.0	26.35397	77.22222	72.22222	38.88889	34.12305	71.875	67.52577	35.61151	153.91935
Partición 5-2	97.89474	96.47887	16.66667	31.91886	81.66666	68.88889	43.33333	34.21568	74.22681	67.1875	37.41007	123.20709
Media	98.03175	96.16889	27.33333	29.21565	77.77778	72.44444	43.88889	33.93815	73.42193	64.71971	32.41007	151.89248

Media

	Wdbc				Movement_Libras				Arrhythmia			
	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T	% clas in	% clas out	% red	T
KNN	96.30974	96.48530	0.0	0.0	70.55556	70.61111	0.0	0.0	62.85653	64.50494	0.0	0.0
SFS	96.76563	93.60390	86.00000	0.13475	77.38889	65.61111	90.44444	0.71810	77.41140	68.96209	97.44604	1.89433
EGA	97.18829	95.43020	45.66667	25.29545	71.83334	73.44444	53.22222	32.77355	66.11630	64.04156	51.33094	121.72561
UXEGA	97.25772	94.90277	49.33333	25.36227	73.61111	72.61111	50.00000	32.82589	66.37350	63.89283	51.76259	121.40811
GGA	97.96219	95.57030	29.66667	28.58018	76.22222	75.38889	44.00000	33.98448	69.53125	63.36877	34.49640	146.04693
UXGGA	98.03175	96.16889	27.33333	29.21565	77.77778	72.44444	43.88889	33.93815	73.42193	64.71971	32.41007	151.89248

En primer lugar, comentamos los 2 algoritmos principales: el KNN y el SFS. El primero de ellos obtiene una buena tasa de acierto en la base de datos pequeña, *Wdbc*, y cuanto más grande es la base de datos, más solapamiento se produce y menos tasa de acierto va teniendo.

El SFS se caracteriza por tener una tasa de acierto que sólo mejora en la base de datos más grande a la del KNN, pero tiene una tasa de reducción bastante alta, ya que al partir de una solución sin características y al parar en cuanto no hay mejora, es muy fácil que se quede con muy pocas características.

Ahora evaluamos las metaheurísticas implementadas. En general, ninguno de los algoritmos genéticos (estacionario o generacional, cruce en dos puntos o uniforme), supera al *GRASP* de la práctica anterior: sus tasas de clasificación son, en general, peores, y las tasas de reducción no tienen comparación.

En particular, el genético estacionario se caracteriza por una convergencia más lenta, ya que apuesta por la exploración del espacio de búsqueda. Es por esto por lo que consigue unas tasas de reducción mayor, en detrimento de la tasa de clasificación, que al no explotar (recordamos que la función fitness es la tasa de acierto) no mejora lo suficiente la tasa de acierto. Por otro lado, el generacional no reduce tanto el número de características, pero es un poco mejor que el generacional en todos los casos excepto en *Movement Libras*, pero la pérdida es mínima.

En el caso del operador de cruce, no hay un cambio muy grande: el cruce uniforme da más diversidad que el cruce en dos puntos, por lo que algoritmo que lo use, algoritmo que ganará en diversidad y perderá explotación, lo que se traduce en más reducción y menos tasa de clasifiación. Aunque no hay un cambio brutal (menos de un punto de diferencia en clasificación, más de 2 en reducción), tomaremos este como el "canónico".

Hablando de tiempos, como todos hacen el mismo número de evaluaciones, 15000, no hay apenas diferencia en tiempo entre uno y otro. La diferencia máxima es de unos 30 segundos en la base de datos más grande, *Arrhythmia*, lo cual no es una pérdida muy grande. Por esta razón no debemos fijarnos en el tiempo para elegir uno de estos algoritmos.

En esta práctica yo me quedaría personalmente con el genético estacionario con cruce uniforme. El cruce uniforme está muy usado en la literatura por ser bastante eficiente en su cometido, ya que introduce más diversidad que el cruce en dos puntos. El estacionario hemos visto que tiene muy poca pérdida con respecto al generacional, con la ventaja de que tiene tasa de reducción de columnas notable. Un poco menos de precisión, pero el tamaño del problema se hará notablemente más pequeño.

7. Referencias

Las referencias utilizadas han sido:

- *Scikit-Learn*: La propia documentación[2] de la biblioteca.
- *SciPy*: La propia documentación[3] de la biblioteca.

[2] http://scikit-learn.org/stable/modules/classes.html
[3] http://docs.scipy.org/doc/scipy/reference/