Metaheurísticas

Estudio sobre Enfoques de la Computación Evolutiva para la Seleccion de Características

Miguel Morales Castillo 75576013Z
miguemc@correo.ugr.es Grupo Lunes

Indice

Introducción

En el problema de selección de caracteristicas (en adelante APC) disponemos de una muestra de objetos ya clasificados w_i,…, w_n , representados en función de sus valores en una serie de atributos:

w_i tiene asociado el vector ($x_1(w_i)$,…,$x_n(w_i)$)

Además, cada objeto pertenece a una de las clases existentes {C_1,…,C_M}

El objetivo es clasificar estos objetos de un modo automático.

Se conocen las clases existentes en el problema, y además, también se conoce la clase concreta a la que pertenece cada objeto del conjunto de datos.

El conjunto de ejemplos se divide en dos subconjuntos:

- Entrenamiento: utilizado para aprender el clasificador.

- Prueba: Se usa para validarlo. Se calcula el porcentaje de clasificación sobre los ejemplos de este conjunto (desconocidos en la tarea de aprendizaje) para conocer su poder de generalización

Además necesitamos un clasificador, que sera el 1-NN vecinos mas cercanos.

Ya que en el mundo real el número de caracteristicas que se presentan es muy numeroso, nos planteamos si es posible reducir el conjunto de características a considerar manteniendo un alto rendimiento. En todos los algoritmos a estudiar se define un número de características que serán seleccionadas, y este número es fijo para todas las ejecuciones, el número varia según el algoritmo en cuestión.

El problema se presenta entonces en que conjunto de caracteristicas son las mas representativas, para ello estudiaremos tres algoritmos de diferentes características y veremos como se comportan en la tarea de seleccionar características, además de comparar su rendimiento con otros algoritmos ya conocidos.

Algoritmos comunes

En este apartado describiremos los algoritmos comunes usados en el problema:

-normalizarDatos: Este método permite normalizar los datos por columna, siguiendo la fórmula

$x_j=x_j\text{-}min_{wj}/max_{wj}\text{-}min_{wj}$

todos los datos estarán siempre normalizados para evitar priorizar unas caracteristicas sobre otras.

-KNN: Este método es nuestra función de evaluación, es el 1-NN pero con ciertos cambios, usa los pesos que estamos optimizando con los distintos algoritmos, y además no trabaja con test sobre train, sino que trabaja con train sobre train usando leave one out, además también lo usaremos como clasificador.

```
FuncionObjetivo (conjuntoTrain, pesos)
    Begin
        Para cada elemento e_i de conjuntoTrain
            buscar su vecino mas cercano en conjuntoTrain e_j i!=j
            si la clase de e_i es igual que la clase de e_j
                acierto+=1;
        tasaClasificación= acierto/conjuntoTrain.size()
        return tasaClasificación
    End
```

-Cruce: Este es el operador de cruce HUX:

```
HUX padre 1, padre 2
    begin
    para cada atributo cj de padre1 y padre2
        hasta que numeroIntercambiados sea padre1.size()/2
            si padre1.cj != padre2.cj
                intercambiar padre1.cj con padre2.cj
    añadimos los dos nuevos cromosomas generados
```

-mutación: Este método simplemente se cambiara la característica de 0 a 1 o de 1 a 0 según corresponda.

-selección: Este método, usa el torneo binario para la selección de los hijos que posteriormente se cruzarán, mutarán, y reemplazarán a la generación anterior.

Particle Swarm Optimization

Introducción

Este algoritmo basado en poblaciones que simula el comportamiento social de organismos como las bandadas de pájaros o los bancos de peces.

La clave de este tipo de algoritmo es que el conocimiento es optimizado mediante la interacción social entre los miembros de la población, en este aspecto se comportarian de forma similar a como lo hacen los cromosomas en los algoritmos genéticos, con la ventaja de que PSO es mas fácil de implementar que un genético ya que no hay operadores de cruce o mutación, y el movimiento de las partículas se lleva a cabo a través de la función velocidad.

Esta implementación de PSO se diferencia del PSO continuo en dos aspectos clave:

- La representación de la solución: En los PSO discretos, la particula la forma un vector de bits donde un 1 representa que la caracteristica esta seleccionada y un 0 que no lo está.

-El vector velocidad: En los PSO discretos, el vector velocidad esta representado por un vector de probabilidad, donde cada elemento representa la probabilidad de que esa caracteristica sea 1.

Algoritmo Propuesto

El PSO propuesto tiene la novedad de incluir una estrategia en el cálculo de el movimiento de la partícula, en lugar de hacerlo con la fórmula habitual, utiliza una subrutina que permite de forma progresiva seleccionar caracteristicas acorde a su relevancia ponderada y a su contribución predictiva, esto permite moverse de forma mas eficaz hacia una solución de calidad, ya que evalua la importancia característica por característica.

A continuación se descibe el proceso adaptativo de selección de características (para cada particula i):

Inicio

Inicialmente el conjunto de caracteristicas seleccionadas L_i es vacio

Aplicar la regla de aleatoriedad proporcional sobre el conjunto total de caracteristicas menos las que ya han sido seleccionadas usando s_{ij} con j=1,2,…,K para seleccionar una característica f.

Introducir f en L_i

Repetir hasta que $|L_i|$=k con k el número de caracteristicas que van a ser seleccionadas

Determinar el conjunto de características que han sido previamente añadidas a L_i

Aplicar la regla de aleatoriedad proporcional sobre las características restantes para seleccionar $|B_i|$ = min{b, número de caracteristicas restantes} características basandose en

Para cada j en B_i, repetir

Calcular h_{ij}

Calcular $m_{ij} = s_{ij} \times h_{ij}$

Aplicar la regla de aleatoriedad proporcional en $F \backslash L_i$ usando m_{ij} para seleccionar una caracteristica f

Añadir f a L_i

Terminar con la posición de la particula i donde la característica j es 1 si j pertenece a L_i y 0 en otro caso.

La regla de aleatoriedad proporcional para seleccionar un subconjunto A de un conjunto B basado en un vector uniforme de numeros aleatorios de tamaño |A| y valores X={x_j con j perteneciente a B} se realiza como sigue:

1.- Crear un vector normalizado a partir de X dividiendo cada elemento por la suma de todos los elementos, X_0 =0 y $X_{|B|+1}$=1.

2.- Seleccionar el elemento j de B para ser incluido en A si el número aleatorio correspondiente a la posición j se queda en el intervalo [x_{j-1}, x_j) para todo k=1,…,|A|.

Por otra parte, m_{ij} se define como la probabilidad ponderada de que la caracteristica j sea 1, y se calcula como el producto de s_{ij} y h_{ij}

h_{ij} representa la relevancia que tiene la característica j en el conjunto de características y se calcula como el fitness de la solución actual con la caracteristica j seleccionada menos el fitness de la solución actual sin esa característica seleccionada.

s_{ij} representa la probabilidad de que una caracteristica sea 1 y se calcula como sigue:

$$s_{ij}=1/1+\exp(-v_{ij})$$

donde v_{ij} representa la j-ésima componente del vector velocidad que en este caso se calcula como:

$$v_i=wv_i + c_1r_1(P_i-X_i)+c_2r_2(G_i-X_i)+c_3r_3(I_i-X_i)$$

donde c_1 , c_2 y c_3 son coeficientes que regulan la influencia de cada componente en el cálculo de la velocidad, y para nuestro estudio valen 2, 1.5 y 0.5 respectivamente, r_1, r_2 y r_3 son números aleatorios entre [0,1] , P es la mejor solucion encontrada por la particula i-ésima, G es la mejor solución encontrada hasta ahora por toda la nube de partículas y I es la mejor solucion de la iteración actual.

W es el peso inercial y se calcula como sigue:

$w = w_{max} – ((w_{max}-w_{min})t)/T$ t= iteración actual; T= total de iteraciones

Por último, pasamos a describir el proceso de nuestro PSO discreto con el proceso adaptativo de selección de características:

Inicio

Inicializar los parámetros c_1, c_2 , c_3, w_{min}, w_{max}, v_{min}, v_{max}

Inicializar $w=w_{max}$, v=0 y L_i vacio

Inicializar la nube, que consta de 20 partículas aleatoriamente de forma que el número de caracteristicas seleccionadas sea k, número fijado de antemano que en este caso vale 5.

Encontrar P, G e I, asi como sus fitness

T=150, t=0

Mientras t<=T

t=t+1, actualizar w según la fórmula vista anteriormente

Para cada particula i

Actualizar su velocidad siguiendo la fórmula vista

Actualizar su posición siguendo el proceso adaptativo de selección de características

Evaluar las soluciones y actualizar P, G e I si procede

Devolver G

SAGA

Introducción

Este algoritmo es una hibridación de la cual proviene su nombre, SAGA proviene de dos importantes algoritmos de búsqueda, Simulated Annealing y Genetic Algorithm, con esta hibridación se pretende suplir las carencias que tiene cada algoritmo por separado, con lo que se pretende obtener una solución de calidad.

Otra peculiaridad de este algoritmo es que basa su ejecución no en un número de iteraciones si no en tiempo total de ejecución permitiendo asi dejar trabajar a cada parte de el algoritmo de forma indefinida hasta que su tiempo se agote.

En nuestro caso hemos optado por asignar un número de evaluaciones a cada parte de forma que para cada parte se mantenga la proporción de tiempo asignado.

Algoritmo Propuesto

Como se ha comentado anteriormente el algoritmo que estudiamos en este caso esta compuesto principalmente de dos algoritmos, Simulated Annealing y Genetic Algorithm, pero además tambien emplea un algoritmo de ascenso de colina para optimizar aun mas la solución.

El algoritmo se divide en tres fases:

-Fase 1: SAGA emplea un SA para guiar la búsqueda global en el espacio de soluciones. Cuando la temperatura es muy alta, SA acepta todas las nuevas soluciones, es casi como una búsqueda aleatoria por el espacio. Por otra parte, cuando la temperatura se vuelve cercana a cero, solo acepta soluciones que mejoren a la actual, como ya sabemos por la literatura, el inconveniente de SA es su lenta convergencia, que se intenta solucionar en la fase 2. Esta fase se ejecuta el 50% del tiempo total disponible.

-Fase 2: SAGA emplea un GA para realizar la optimizacion. La población inicial es de 100 individuos, los cuales han pasado previamente por la fase 1, el proposito de esta fase es el intercambio de genes entre buenas soluciones con la esperanza de conseguir soluciones aún mejores, el operador de cruce utilizado es HUX y permite una rápida convergencia que era lo que se pretendia paliar de la fase 1, además el operador de mutación permite añadir diversidad genetica a la población agregando nuevos genes. Esta fase se ejecuta un 30% del tiempo total disponible.

-Fase 3: SAGA aplica un algoritmo de ascenso de colina sobre la solución devuelta por GA y realiza una busqueda local en las mejores soluciones, esto permita afinar aún mas en la convergencia, además genera un vecindario para cada nueva solucion aceptada, permitiendo asi salir de posibles

óptimos locales donde se pudiese caer durante la fase 2 ya que como sabemos GA no es tan bueno sorteando optimos locales como SA.

-Pseudocódigo de Simulated Annealing

SA (particionTrain)

Inicializar temperatura inicial, solucion inicial

Begin

mientras no se llegue a t evaluaciones

Elegir aleatoriamente el peso a mutar

Mutar el peso elegido a 1 si era 0 y viceversa

Evaluar la nueva solución

Si solucion actual es mejor que solucion mutada o numAleatorio(0,1)<exp(Δf/temp)

solucion actual = solucion mutada

Si es mejor que la solucion global

solucion global=solucion mutada

si no

se revierte la mutacion

temp=temp-numEvaluaciones

return mejor solucion

End

-Pseudocódigo de GA, con los operadores de selección, mutación y cruce definidos al inicio de este estudio.

GA(particionTrain)

mientras Evaluaciones<TotalEvaluaciones

Realizar la selección

Realizar el cruce

Realizar la mutación

Realizar el reemplazamiento

Devolver el mejor de los 100 individuos que forman la población

-Pseudocódigo del algoritmo HillClimbing

HillClimbing(particionTrain)

Dada la mejor solución obtenida por GA

Generar 300 vecinos cambiando solo un bit cada vez

Sustituir las que sean mejores que la solucion original

Para cada una repetir el proceso anterior hasta que se agote el número de evaluaciones.

Devolver la mejor solución encontrada.

-Pseudocódigo SAGA

SAGA(particionTrain)

Inicializar 100 individuos de forma aleatoria

Aplicar SA a cada uno con 200 iteraciones por individuo

Aplicar GA (12000 iteraciones) tomando como población inicial los 100 individuos devueltos por SA

Aplicar HillClimbing(8000 iteraciones) a la mejor solución devuelta por GA

Devolver la mejor solución encontrada por HillClimbing

Evolución Diferencial

Introducción

Este algoritmo se basa en crear una nueva solución (V) a partir de otras soluciones si se cumple el criterio de mutación, o a partir de el padre en caso contrario, la formacion de esta nueva solucion se realiza componente a componente.

Algoritmo Propuesto

En el estudio de este algoritmo no se pretende realizar un avance en cuanto a un nuevo algoritmo de evolución diferencial se refiere, mas bien se pretende demostrar un nuevo método para la elección de las caracteristicas, este metodo esta basado en ruletas y se describe como sigue:

1. Primero se fija el número de caracteristicas seleccionadas que tendrán nuestras soluciones y ese será el número total de ruletas a crear.

2.Distribuimos uniformemente todas las caracteristicas en las ruletas

3.Una nueva solución se forma seleccionando una caracteristica de cada rueda y el resto de caracteristicas a 0.

-Pseudocódigo del algoritmo DEWheel

DEWheel(particionTrain)

Identificar el número de ruletas a usar y distribuir las caracteristicas uniformemente

Generar 50 individuos seleccionando una caracteristica de cada rueda

Evaluar los individuos e identificar a los k mejores (en nuestro caso k=5)

Mientras iter<TotalIteraciones

> Generar nuevos individuos aplicando combinación diferencial o cruce uniforme según corresponda, con probabilidad de cruce 0.5
>
> Si los nuevos individuos generados son mejores, aceptamos la solución.
>
> Volvemos a identificar a los k mejores si estos han cambiado
>
> Si iter mod 70 ==0
>
> > Fijar las caracteristicas de los k mejores en sus respectivas rueletas
> >
> > Distribuir aleatoriamente el resto de caracteristicas.
> >
> > Generar 50-k nuevos individuos y evaluarlos

Devolver la mejor solucion encontrada

Para la combinacion diferencial escogemos aleatoriamente dos soluciones distintas de i, una de las k mejores y otra de el resto de soluciones, llamaremos a estas soluciones m y n respectivamente, V por tanto sera calculado de la siguiente forma

$$V_{i,j} = X_{i,j} + F \cdot (X_{i,m} - X_{i,n})$$

donde V_i es la nueva solucion formada a partir de X_i, F vale 0.5 y pondera el peso de las soluciones externas en la formación de la nueva.

Para el operador de cruce se utiliza la solucion m elegida entre los k mejores y se obtiene de la siguiente forma

$$V_{i,j} = X_{i,m}$$

PSO Modificado

Algoritmo Propuesto

Como mejora al PSO descrito en la sección anterior, hemos tomado la decisión de hibridarlo con el algoritmo HillClimbing de SAGA a la mejor solución devuelta por PSO dando resultados ligeramente mejores sin un coste muy superior de tiempo.

Proceso considerado para desarrollar la práctica

Para el desarrollo de la práctica he seguido una implementación desde cero, usando el lenguaje de programación C++, para usar la práctica solo es necesario hacer ejecutar el makefile, y ./bin/APC,

el main esta hecho para que ejecute todos los algoritmos para las tres bases de datos, por tanto se recomienda comentar los algoritmos que no sean necesarios ejecutar para no gastar tiempo innecesario de ejecución.

En este apartado me gustaría mencionar a mi compañero Luis Liñán Villafranca en agradecimiento por facilitarme su código de lectura de ARFF en C++, que además proporciona una estructura de datos para almacenar las particiones y los objetos por separado.

Experimentos y análisis de resultados

Realizamos para cada dataset 5 particiones Test y 5 particiones Train al 50% de los datos, en nuestro experimento contamos con 5 dataset de diferentes tamaños para probar la capacidad de clasificacion que tienen los algoritmos que son objeto de estudio, tres de ellas ya nos son conocidas, Wdbc, Sonar y SpamBase, pero para este estudio en concreto hemos añadido dos dataset mas.

Ionosfera: Este dataset clasifica los resultados devueltos por radares soltados en la ionosfera, las posibles clases son “malo” o “bueno” representadas como b y g respectivamente

Diabetes: Este dataset al igual que Wdbc clasifica si el paciente en cuestion padece diabetes o no.

Debido a que nuestra capacidad computacional y tiempo disponible es bastante menor al de los autores de los respectivos papers en los que basamos nuestro estudio los cuales se adjuntan con esta memoria de trabajo, se han modificado todos los parametros para reducir su tiempo de ejecucion maximizando la capacidad de predicción de los mismos.

A continuación se presentan los resultados obtenidos

1-NN

	sonar		wdbc		spambase		ionosfera		diabetes	
	%_class	T	%_class	T	%_class	T	%_class	T	%_class	T
Particion 1-1	73,7900	0,0038	93,9900	0,0158	82,0200	0,0168	87,4286	0,007093	69,3717	0,0208841
Particion 1-2	79,0500	0,0037	95,4500	0,0178	80,1700	0,0173	82,5843	0,00788	69,4301	0,0210314
Particion 2-1	82,5200	0,0038	95,0500	0,0162	81,1400	0,0171	86,8571	0,00684	70,9424	0,0180278
Particion 2-2	79,0500	0,0037	93,3600	0,0164	80,6000	0,0169	83,7079	0,00753	67,8756	0,018314
Particion 3-1	85,4400	0,0037	91,8700	0,0155	83,3300	0,0214	86,8571	0,006908	70,9424	0,0179993
Particion 3-2	77,1400	0,0040	93,7100	0,0162	81,4700	0,0178	84,8315	0,007286	67,8756	0,0183
Particion 4-1	79,61	0,004	92,23	0,0164	81,14	0,0173	88	0,006911	67,2775	0,0185671
Particion 4-2	81,9	0,0045	91,96	0,0168	78,45	0,0175	84,8315	0,007273	69,171	0,0192684
Particion 5-1	83,5	0,0037	95,41	0,0155	83,77	0,0168	87,4286	0,006979	67,5393	0,0180721
Particion 5-2	85,71	0,0037	96,85	0,0162	76,72	0,0168	83,1461	0,007222	64,7668	0,0182421
Media	80,7710	0,0039	93,9880	0,0163	80,8810	0,0176	85,5673	0,0072	68,5192	0,0189

SAGA

	sonar		wdbc		spambase		ionosfera		diabetes	
	%_class	T	%_class	T	%_class	T	%_class	T	%_class	T
Particion 1-1	83,4951	110,9060	93,6396	381,0600	87,2807	518,7660	87,4286	258,345	68,5864	650,487
Particion 1-2	80,0000	105,9380	94,0559	373,4260	82,7586	492,5180	91,0112	248,951	63,2124	811,342
Particion 2-1	81,5534	111,3120	92,9329	438,6270	81,5789	518,5750	90,8571	267,723	68,3246	657,233
Particion 2-2	80,9524	105,2590	95,1049	407,3520	85,3448	502,9970	86,5169	257,684	66,0622	644,905
Particion 3-1	83,4951	110,3270	92,2262	406,6690	82,0175	516,8880	86,2857	266,791	71,9895	663,351
Particion 3-2	80,9524	104,9830	92,6573	371,1780	85,7759	481,6750	87,6404	257,757	69,1710	647,0640
Particion 4-1	81,5534	109,324	92,9329	464,604	78,0702	511,752	86,2857	264,388	68,5864	657,626
Particion 4-2	77,1429	103,403	88,8112	407,148	71,5517	475,712	87,6404	230,128	70,2073	651,035
Particion 5-1	79,6117	108,365	94,3463	460,739	82,8947	505,038	88	265,734	69,8953	792,064
Particion 5-2	85,7143	102,858	95,4545	433,042	76,2931	480,307	91,573	258,164	72,0207	783,406
Media	81,4471	107,2675	93,2162	414,3845	81,3566	500,4228	88,3239	257,5665	68,8056	695,8513

PSO

	sonar		wdbc		spambase		ionosfera		diabetes	
	%_class	T	%_class	T	%_class	T	%_class	T	%_class	T
Particion 1-1	67,9612	208,6790	93,2862	859,7000	80,2632	923,6830	89,1429	281,92	68,5864	944,451
Particion 1-2	68,5714	191,7810	94,0559	866,1430	77,1552	877,2010	89,3258	339,964	63,2124	892,271
Particion 2-1	79,6117	203,9640	92,2262	861,5180	77,6316	878,3800	89,7143	299,356	64,1361	949,779
Particion 2-2	65,7143	217,9820	90,9091	840,5080	78,0172	842,7980	86,5169	319,239	66,0622	929,391
Particion 3-1	62,1359	204,2330	92,2262	867,7070	86,8421	876,8990	81,7143	308,288	67,5393	947,794
Particion 3-2	71,4286	218,8890	88,1119	844,9460	84,9138	850,2200	85,3933	322,382	70,2073	910,4730
Particion 4-1	71,8447	203,213	95,053	866,087	79,8246	870,534	85,7143	323,909	66,7539	931,198
Particion 4-2	75,2381	217,327	91,958	845,037	86,2069	842,638	88,2022	321,529	70,2073	913,087
Particion 5-1	66,9903	203,246	91,5194	861,332	82,8947	872,078	88,5714	309,752	62,3037	942,609
Particion 5-2	64,7619	218,001	93,007	840,449	84,4828	841,521	87,0787	320,901	70,4663	930,58
Media	69,4258	208,7315	92,2353	855,3427	81,8232	867,5952	87,1374	314,7240	66,9475	929,1633

DE

	sonar		wdbc		spambase		ionosfera		diabetes	
	%_class	T	%_class	T	%_class	T	%_class	T	%_class	T
Particion 1-1	77,6699	68,8995	94,6996	318,1050	78,0702	295,7670	88,5714	166,847	65,544	489,501
Particion 1-2	69,5238	71,0128	92,3077	308,4700	83,6207	287,2180	84,2697	162,777	68,344	459,311
Particion 2-1	74,7573	73,5809	92,9329	315,3810	82,8947	327,0370	89,1429	176,284	64,524	490,29
Particion 2-2	73,3333	69,1692	94,0559	308,6700	84,0517	381,9570	84,2697	158,416	70,233	475,632
Particion 3-1	79,6117	73,8348	91,5194	317,2550	86,8421	327,7290	87,4286	164,244	69,766	500,541
Particion 3-2	72,3810	69,9762	91,6084	309,2160	82,7586	383,4590	84,8315	170,729	64,7490	466,5570
Particion 4-1	76,699	73,355	93,9929	316,464	76,7544	326,535	84	165,698	68,224	432,528
Particion 4-2	86,6667	69,4522	92,3077	309,311	85,7759	382,789	86,5169	171,819	67,713	478,514
Particion 5-1	75,7282	73,0076	90,106	315,849	85,5263	325,812	88	165,239	70	458,552
Particion 5-2	80,9524	69,2205	96,5035	307,863	85,3448	382,323	89,3258	170,264	71,917	477,561
Media	76,7323	71,1509	93,0034	312,6584	83,1639	342,0626	86,6357	167,2317	68,1014	472,8987

	sonar		wdbc		spambase		ionosfera		diabetes	
	%_class	T	%_class	T	%_class	T	%_class	T	%_class	T
1-NN	80,7710	0,0039	93,9880	0,0163	80,8810	0,0176	85,5673	0,0072	68,5192	0,0189
SAGA	81,4471	107,2675	93,2162	0414,38	81,3566	500,4228	88,3239	257,5665	68,8056	695,8513
PSO	69,4258	208,7315	92,2353	0855,34	81,8232	867,5952	87,1374	314,7240	66,9475	929,1633
DE	76,7323	71,1509	93,0034	0312,66	83,1639	342,0626	86,6357	167,2317	68,1014	472,8987
PSOModificado	77,7282	195,3160					89,5714	469,8760		

Análisis de los datos

Como análisis de los resultados obtenidos podemos ver que los que los algoritmos que realizan una búsqueda local en el proceso, dan mejores resultados, como es el caso de SAGA y PSO modificado.

SAGA en particular, debido a su composicion en tres fases supliendo las carencias de cada parte con la siguiente, es el que mejores resultados ofrece además de hacerlo en un tiempo no demasiado elevado si lo comparamos con PSO.

SAGA además debe la calidad de sus soluciones a la potencia de exploración de el SA y su capacidad para sortear óptimos locales, al usarse el 50% de el tiempo, permite una gran exploracion durante las temperaturas elevadas y se ejecuta el tiempo suficiente como para que la temperatura disminuya considerablemente y solo se acepten soluciones mejores.

Al llegar al genético ofrece una rápida convergencia la cual no puede ofrecer SA, llegando asi rápidamente a una solución en principio mejor que las que ofrecia SA ya que además gracias al operador de mutación ofrece cierta diversidad en el espacio de soluciones.

Por último influye mucho en la calidad de la solucion respecto a PSO que seria su competidor directo por tratarse ambos de algoritmos basados en poblaciones, el hecho de que a la solución devuelta le aplique sea búsqueda local generando un vecindario y aceptando todos aquellos vecinos mejores que la solución original.

Por otra parte, observando los datos obtenidos en el PSO vemos que perjudica considerablemente a los dataset que tienen un alto número de caracteristicas como es el caso de Sonar, esto se debe a que fijamos de antemano un número fijo de caracteristicas a seleccionar, el cual nunca se ve modificado y que aumentarlo demasiado supone un coste computacional muy elevado, en nuestro caso ese número era 5, y por tanto para el caso de Sonar donde el número de caracteristicas es 60, se ve perjudicado ya que no selecciona todas las que tienen relevancia en la tarea de clasificación.

Si que funciona bien en datasets con pocas caracteristicas aunque tenga un elevado número de ejemplos, a pesar de que las soluciones no sean tan buenas como las que ofrece SAGA, pero debemos tener en cuenta que para este estudio hemos tenido que reducir los parámetros para reducir el tiempo de ejecución, como por ejemplo es el caso de el número de caracteristicas a seleccionar, o el conjunto de caracteristicas que se selecciona cada iteracion en el proceso adaptativo de selección de características.

Como mejora a esto, hemos modificado el PSO original añadiendo a la solucion devuelta una búsqueda local, la misma que usa SAGA, quitando asi la limitacion de las 5 caracteristicas y permitiendo explorar un mayor número de soluciones con un mayor número de caracteristicas seleccionadas, y como vemos los resultados mejoran considerablemente, hemos ajustado el número de iteraciones de esta búsqueda local de manera que el tiempo de ejecución no aumentase demasiado ya que de por si el PSO tiene un tiempo de ejecución elevado.

Gracias a esta mejora, PSO modificado es capaz de competir con SAGA en cuanto a calidad de soluciones se refiere, esta mejora sera mayor cuantas mas características tenga el dataset sobre el que se aplique, que era la principal carencia que poseia esta implementacion de PSO, que con esta mejora quedaría solucionada, solo se han realizado ejecuciones en los data set con menor número de ejemplos ya que simplemente queriamos probar su eficacia sobretodo en el dataset Sonar que era donde el PSO original mas lo habia perjudicado.

El algoritmo de Evolucion diferencial es el que mejor rendimiento tiene ya que ofrece soluciones de una calidad muy buena y es el que menos iteraciones necesita (solo 560), esto se ve reflejado en que es el algoritmo que tiene un menor tiempo de ejecución con bastante diferencia respecto de los demás, esta capacidad de ofrecer soluciones tan alta calidad en tan poco tiempo se debe a su método para la selección de caracteristicas en el cual fija siempre las mejores en su rueleta correspondiente permitiendo asi construir estas soluciones a partir de las mejores de la generacion anterior. Esto nos hace pensar que al igual que se utiliza para un algoritmo de evolución diferencial, sería posible adaptarlo para usarlo sobre un algoritmo genético ya que el esquema es el mismo, ambos usan combinación, cruce y reemplazamiento.

Como se ha expresado alguna vez a lo largo de este estudio, esta implementación de DE no pretende mejorar al resto si no que pretende demostrar la utilidad de el método de selección mediante ruletas que es donde se encuentra la innovación, se podia haber elegido un genético o incluso un PSO, ha quedado demostrado tanto en tiempo de ejecución como en calidad de

soluciones que el método de selección por ruletas es mas que válido, ya que además gracias a su reemplazamiento cada cierto número de iteraciones permite que si dos buenas características se quedan atascadas en una ruleta y por tanto no podrian estar en la misma solución, una de ellas quedaria fija y la otra tendria posibilidad de ser redistribuida a otra ruleta para asi pertenecer ambas a la misma solución.

Conclusión

Como conclusión a este estudio podemos afirmar que en este caso los algortimos basados en trayectorias han dado un mejor resultado que los basados en poblaciones, sobre todo en lo que a tiempo de ejecución se refiere

Por otra parte comentar lo innovador que resultan las tecnicas usadas en los tres algoritmos estudiados:

-El proceso adaptativo de selección de caracteristicas de PSO ya no usa solo el vector velocidad si no que además las elige según su importancia en la tarea de clasificación dando una mayor orientación a las partículas hacia soluciones de calidad.

-El proceso en tres fases de SAGA resulta muy sencillo de implementar, y aunque parece demasiado obvio el combinar algoritmos que ya sabemos que son potentes, el hecho de saber distribuir los recursos en cada fase hace que sea una potente hibridación

-El proceso de selección basado en ruletas de DE es el que resulta mas innovador ya que además de dar muy buenos resultados es posible aplicarlo a muchos algoritmos y queda demostrada su efectividad.