Metaheurísticas (2015-2016)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 5.b: Búsquedas Híbridas para el Problema de Selección de Características

Laura Tirado López
DNI: 77145787S
email: muscraziest@correo.ugr.es
Grupo de prácticas 1
Horario: Jueves 17:30-19:30

13 de junio de 2016

Índice

1. Introducción al problema de Selección de Características

El problema de selección de características consiste en determinar la clase a la que pertenecen un conjunto de datos o instancias caracterizadas por una serie de atributos. En el aprendizaje supervisado utilizamos un conjunto de instancias con sus correspondientes atributos y un atributo adicional que indica la clase a la que pertenece cada instancia. La idea es crear y generalizar una regla o un conjunto de reglas a partir de este conjunto que nos permita clasificar de la mejor manera posible un conjunto de instancias. Para clasificar los datos existen muchos métodos o reglas. En nuestro caso, vamos a implementar un clasificador k-NN que sigue el criterio de los k vecinos más cercanos. Els número de vecinos que utilizaremos será 3.

A pesar de esto, en muchos casos se observa que no todas las variables o características aportan la misma calidad de información, pudiendo incluso complicar la creación de las reglas de clasificación y hacer que clasifiquen de forma errónea. Por lo tanto, el objetivo del problema de selección de variables no es sólo clasificar las instancias sino también encontrar un subconjunto de variables con las que se pueda clasificar de forma óptima, clasificando la mayor cantidad posible de instancias de forma correcta.

La búsqueda de este subconjunto de variables es un problema NP-duro, por lo cual el uso de metaheurísticas en este problema nos permite obtener soluciones razonablemente buenas en un tiempo razonable sin llegar a explorar todo el espacio de soluciones.

Para resolver este problema, vamos a utilizar metaheurísticas de búsquedas por trayetoria: búsqueda local primero el mejor, enfriamiento simulado y búsqueda tabú. Además, haremos una comparación entre estas heurísticas y un algoritmo de comparación greedy.

2. Aplicación de los algoritmos

Primero tenemos que definir la representación de las soluciones. En este caso, hemos utilizado una representación de un vector binario, en el que si en la posición i hay un 1 significa que la característica i es seleccionada y si hay un 0 no es seleccionada. Esta representación es lo que llamamos la máscara.

La función objetivo consiste en llamar al clasificador 3-NN y hacer el recuento de aciertos (coste) con la máscara que le pasamos. En pseudocódigo sería:

```
1 funcion_objetivo{
2         Para cada caracteristica
3                 Si knn(datos,mascara) = datos[caracteristica][
                      total caracteristicas-1] \\
4                       coste++
5         return (coste)
6 }
```

A la hora de evaluar un vecino, si el coste con el nuevo vecino es mayor que el coste de la mejor solución hasta el momento, la mejor solución pasa a ser el nuevo vecino y actualizamos el valor del coste.

Para la generación de vecinos, los generamos haciendo el cambio de pertenencia $Flip(s, i)$, en el que modificamos el valor de la posición i de la máscara s a su valor contrario. En el caso de la búsqueda local generamos tantos vecinos como características tengamos. Para cada vecino se aplica un movimiento $Flip(s, i)$ empezando la i en 0 para el primer vecino, 1 para el segundo y así hasta generarlos todos. La función se llama $generarVecinos$. El pseudocódigo es el siguiente:

```
generarVecinos{
        Desde 0 hasta el numero de caracteristicas:

                Hacemos el cambio Flip

                Aniadimos la mascara a la matriz de vecinos

                Deshacemos el cambio Flip

        return (vecinos)
}
```

También podemos utilizar el movimiento $Flip(s, i)$ escogiendo la posición i de forma aleatoria y haciendo el cambio de valor de esa posición. La función se llama $generarVecinoAleatorio$.

```
generarVecinoAleatorio{

        Generamos un indice aleatorio
        Cambiamos el valor de la mascara en dicho indice

        return (mascara)
}
```

La generación de la solución aleatoria inicial de los algoritmos BMB e ILS se crea simplementen generando valores aleatorios 0 o 1.

```
generarMascaraInicial{
        Desde 0 hasta el numero de caracteristicas
                Generamos valores aleatorios 0 o 1 para cada
                   elemento de la mascara

        return(mascara)
}
```

El algoritmo de búsqueda local empleado es el implementado en la práctica 1. El pesu-

docódigo del algoritmo es el siguiente:

```
bl{

        Generamos los vecinos

        Para cada vecino generado

                Calculamos el coste de cada vecino

                Si el coste del nuevo vecino es mayor
                        Actualizamos la solucion
                        Criterio de parada

        return(mejor-solucion)
}
```

La máscara inicial, la generación de los vecinos y el cálculo del coste de cada uno lo hacemos utilizando las funciones descritas anteriormente.

El método de exploración del entorno consiste en generar todos los vecinos asociados a nuestra máscara inicial y para cada vecino generado calculamos su coste. Si el coste del vecino seleccionado es mejor que el de la solución actual, actualizamos nuestra solución y salimos del bucle por ser una búsqueda local primero mejor. En caso contrario, evaluamos el siguiente vecino hasta encontrar una solución mejor o haber recorrido todo el vecindario.

2.1. Algoritmo AM(10,1.0)

El algoritmo AM(10,1.0) es un algoritmo híbrido que combina los algoritmos AGG y BL.

El algoritmo AGG es un algoritmo genético con un esquema generacional con elitismo. Este algoritmo genera una población nueva del mismo tamaño que la anterior en cada iteración pero sustituye el peor individuo de la población nueva por el mejor de la población anterior.

El algoritmo BL se ejecuta dentro del algoritmo AGG cada 10 generaciones y sobre todos los cromosomas de la nueva población seleccionada. De esta forma, se crea una mayor diversidad.

El pseudocódigo del algoritmo es el siguiente:

```
AM(10,1.0){

        Generamos una poblacion de 10 individuos de forma
           aleatoria
        Evaluamos el coste de cada individuo
```

```
5
6           num_evaluaciones = 0
7
8           Mientras num_evaluaciones < maximo_evaluaciones
9
10                  Seleccionamos una nueva poblacion
11
12                  Realizamos los cruces en la nueva poblacion
13
14                  Realizamos las mutaciones en la nueva poblacion
15
16                  Intercambiamos el peor elemento de la nueva
                       poblacion por el mejor de la antigua poblacion
17
18                  Sustituimos la antigua poblacion por la nueva
19
20                  Si hemos han pasado 10 generaciones
21
22                          Aplicamos busqueda local sobre toda la
                               poblacion
23
24          Devolvemos la mejor solucion
25
26  }
```

2.2. Algoritmo AM(10,0.1)

El algoritmo AM(10,0.1) es un algoritmo híbrido que combina los algoritmos AGG y BL. La diferencia entre este algoritmo y el anterior es que la búsqueda local sólo se aplica sobre uno de los individuos elegido de forma aletoria.

El pseudocódigo del algoritmo es el siguiente:

```
1   AM(10,0.1){
2
3           Generamos una poblacion de 10 individuos de forma
               aleatoria
4           Evaluamos el coste de cada individuo
5
6           num_evaluaciones = 0
7
8           Mientras num_evaluaciones < maximo_evaluaciones
9
10                  Seleccionamos una nueva poblacion
11
12                  Realizamos los cruces en la nueva poblacion
13
```

```
                    Realizamos las mutaciones en la nueva poblacion

                    Intercambiamos el peor elemento de la nueva
                       poblacion por el mejor de la antigua poblacion

                    Sustituimos la antigua poblacion por la nueva

                    Si hemos han pasado 10 generaciones

                            Aplicamos busqueda local sobre un
                               individuo aleatorio

            Devolvemos la mejor solucion

}
```

2.3. Algoritmo AM(10,0.1M)

El algoritmo AM(10,0.1M) es un algoritmo híbrido que combina los algoritmos AGG y BL. Este algoritmo, al igual que el AM(10,0.1) sólo aplica la BL sobre un individuo, pero en este caso no se escoge de forma aleatoria, sino que escoge el mejor individuo de la población.

El pseudocódigo del algoritmo es el siguiente:

```
AM(10,0.1M){

        Generamos una poblacion de 10 individuos de forma
           aleatoria
        Evaluamos el coste de cada individuo

        num_evaluaciones = 0

        Mientras num_evaluaciones < maximo_evaluaciones

                Seleccionamos una nueva poblacion

                Realizamos los cruces en la nueva poblacion

                Realizamos las mutaciones en la nueva poblacion

                Intercambiamos el peor elemento de la nueva
                   poblacion por el mejor de la antigua poblacion

                Sustituimos la antigua poblacion por la nueva

                Si hemos han pasado 10 generaciones
```

```
21
22                          Aplicamos busqueda local sobre el mejor
                               individuo
23
24          Devolvemos la mejor solucion
25
26  }
```

3. Algoritmo de comparación: SFS

El algoritmo de comparación es un algoritmo greedy SFS (Sequential Forward Selection). Este algoritmo parte de una máscara con todos los valores a 0 y calcula el coste de cada característica al ser añadida a la solución actual de forma individual, es decir, genera todas las posibles soluciones con cada característica. La característica que añadimos a la máscara es la que proporcione mayor ganancia. Si en algún momento no se encuentra ninguna característica que mejore la solución actual, el algoritmo finaliza. El pseudocódigo sería el siguiente:

```
1   sfs{
2
3           Mientras fin = false
4
5                   Calculamos efectividad
6                           Para cada mascara en la que cambiemos una
                               caracteristica calculamos su coste
7
8                   Si hay ganancia, actualizamos la solucion
9
10                  Si no hay ganancia
11                          fin = true
12
13          return(mejor-solucion)
14  }
```

4. Procedimiento del desarrollo de la práctica

Para el desarrollo de la práctica desarrolle el código desde 0 a partir de lo explicado en clase y en las diapositivas,

Para compilar el código de la práctica se necesitan el archivo practica5.cpp random_ppio.c libarff.a libgtest.a:

g++ -o p practica5.c random_ppio.c libarff.a libgtest.a

En la carpeta FUENTES se incluye un script para compilar el código y crear el ejecutable en la carpeta BIN.

Al ejecutar, nos muestra un menú de opciones:

- Opción 1: ejecutar AM(10,1.0)
- Opción 2: ejecutar AM(10,0.1)
- Opción 3: ejecutar AM(10,0.1M)
- Opcoón 4: salir

Y a continuación nos pide el número de simulación, es decir, el archivo con los datos que le vamos a pasar. Las opciones van desde 1 a 30. De cada conjunto de datos hay 10 archivos distintos:

- Opciones 1-10: archivos de datos de Movement_Libras
- Opciones 11-20: archivos de datos de Arrhytmia
- Opciones 21-30: archivos de datos Wdbc

5. Experimentos y análisis de los resultados

5.1. Casos del problema

Para los datos utilizados para probar los algoritmos, creé diez archivos de cada conjunto de datos mezclándolos para tener las 10 simulaciones.

El tamaño de las poblaciones es de 10 individuos y el criterio de parada de los algoritmos es de 1500 evaluaciones. La probabilidad de cruce para el algoritmo es de 0,7. La probabilidad de mutación es 0,001.

El valor de la semilla que he utilizado es 5 para todos los algoritmos.

5.2. Resultados

Los resultados obtenidos se muestran en las siguientes tablas:

Tabla 5.2.1: Resultados obtenidos por el algoritmo SFS en el problema de la SC

	Wdbc			Movement_Libras			Arrhythmia		
	%_clas	*%_red*	T	*%_clas*	*%_red*	T	*%_clas*	*%_red*	T
Partición 1-1	94,39	83,33	45,93	81,67	97,78	59,75	84,46	97,84	1639,31
Partición 1-2	97,54	86,67	38,79	85,56	94,44	118,25	81,87	96,76	2344,13
Partición 2-1	96,49	90,00	31,22	86,67	95,56	102,38	80,83	97,84	1603,89
Partición 2-2	98,95	86,67	38,69	90,00	93,33	150,93	81,87	97,48	1824,78
Partición 3-1	96,14	86,67	38,67	77,78	96,67	89,21	85,49	97,48	1814,64
Partición 3-2	95,44	86,67	38,48	90,56	95,56	112,43	84,46	97,84	1579,54
Partición 4-1	96,49	83,33	45,79	91,67	95,56	112,47	84,97	97,48	1794,46
Partición 4-2	97,45	83,33	46,46	79,44	97,78	68,92	82,38	96,76	2245,55
Partición 5-1	98,60	86,67	38,55	85,00	96,67	92,94	78,76	97,48	1800,08
Partición 5-2	97,54	86,67	38,66	85,00	97,78	71,42	86,01	96,40	2445,29
Media	96,90	86,00	40,12	85,34	96,11	97,87	83,11	97,34	1909,17

Tabla 5.2.2: Resultados obtenidos por el algoritmo AM(10,1.0) en el problema de la SC

	Wdbc			Movement_Libras			Arrhythmia		
	%_clas	*%_red*	T	*%_clas*	*%_red*	T	*%_clas*	*%_red*	T
Partición 1-1	92,98	50,00	439,45	97,78	52,22	411,13	79,79	49,28	1266,11
Partición 1-2	84,56	43,33	412,88	92,22	54,44	496,68	79,27	50,00	1495,71
Partición 2-1	94,74	53,33	479,09	86,11	47,78	378,17	79,79	45,32	1483,67
Partición 2-2	99,30	40,00	400,25	87,22	40,00	550,98	82,38	49,28	1272,04
Partición 3-1	91,58	60,00	402,14	88,33	46,67	431,71	79,79	46,40	1663,32
Partición 3-2	94,04	50,00	410,62	94,44	53,33	430,88	83,94	51,08	1528,28
Partición 4-1	94,74	43,33	454,87	87,78	50,00	538,63	82,90	46,04	1714,85
Partición 4-2	92,98	40,00	471,23	93,89	51,11	447,83	82,38	51,44	2293,74
Partición 5-1	94,74	46,67	479,65	92,22	45,56	489,59	81,35	48,92	1256,98
Partición 5-2	96,49	46,67	423,72	93,33	46,67	612,41	77,20	52,52	1690,46
Media	93,61	47,33	437,39	91,33	48,78	478,80	80,88	49,03	1566,52

Tabla 5.2.3: Resultados obtenidos por el algoritmo AM(10,0.1) en el problema de la SC

	Wdbc			Movement_Libras			Arrhythmia		
	%_clas	*%_red*	T	*%_clas*	*%_red*	T	*%_clas*	*%_red*	T
Partición 1-1	94,14	56,67	403,63	93,33	48,89	412,40	83,94	50,36	1260,56
Partición 1-2	97,89	50,00	408,76	91,11	54,44	386,63	85,49	47,48	1307,06
Partición 2-1	97,54	63,33	402,66	91,11	56,67	385,92	79,27	45,32	1346,70
Partición 2-2	97,89	46,67	403,06	90,00	56,67	385,46	80,83	51,80	1387,62
Partición 3-1	94,14	43,33	403,13	94,44	54,44	384,37	79,79	50,72	1278,37
Partición 3-2	97,89	53,33	404,50	92,78	52,22	384,28	80,83	45,32	1337,65
Partición 4-1	95,44	26,67	441,64	85,00	47,78	388,72	79,27	48,56	1263,77
Partición 4-2	97,19	43,33	490,16	89,44	45,56	392,40	81,35	53,24	1259,25
Partición 5-1	90,88	36,67	494,46	90,00	55,56	397,03	79,79	50,36	1312,34
Partición 5-2	96,49	30,00	486,27	87,78	50,00	383,73	74,61	48,56	1348,76
Media	95,95	45,00	433,83	90,50	52,22	390,09	80,52	49,17	1310,21

Tabla 5.2.4: Resultados obtenidos por el algoritmo AGG en el problema de la SC

	Wdbc			Movement_Libras			Arrhythmia		
	%_clas	*%_red*	T	*%_clas*	*%_red*	T	*%_clas*	*%_red*	T
Partición 1-1	97,54	56,67	400,26	98,33	54,44	440,25	80,83	54,68	1293,85
Partición 1-2	97,54	50,00	423,84	86,67	45,56	440,85	81,35	48,56	1307,52
Partición 2-1	97,54	60,00	403,44	90,56	50,00	369,34	82,38	52,52	1233,18
Partición 2-2	95,09	43,33	406,30	91,67	48,89	380,97	80,83	55,04	1423,94
Partición 3-1	93,68	36,67	401,36	92,78	55,56	385,31	81,87	52,16	1505,25
Partición 3-2	94,39	60,00	426,13	93,33	51,11	375,19	80,83	51,44	1492,48
Partición 4-1	93,68	33,33	414,42	87,22	56,67	376,02	82,38	44,24	1254,57
Partición 4-2	93,33	36,67	421,45	88,89	57,78	386,40	78,76	51,08	1259,92
Partición 5-1	97,89	63,33	422,05	91,11	52,22	374,47	77,20	47,48	1345,40
Partición 5-2	96,14	40,00	427,18	89,44	44,44	377,16	79,27	45,68	1396,75
Media	95,68	48,00	414,64	91,00	51,67	390,60	80,57	50,29	1351,29

Tabla 5.2.5: Resultados globales en el problema de la SC

	Wdbc			Movement_Libras			Arrhythmia		
	%_clas	*%_red*	T	*%_clas*	*%_red*	T	*%_clas*	*%_red*	T
SFS	96,90	86,00	40,12	85,34	96,11	97,87	83,11	97,34	1909,17
AM(10,1)	93,61	47,33	437,39	91,33	48,78	478,80	80,88	49,03	1566,52
AM(10,0.1)	95,95	45,00	433,83	90,50	52,22	390,09	80,52	49,17	1310,21
AM(10,0.1M)	95,68	48,00	414,64	91,00	51,67	390,60	80,57	50,29	1351,29

5.3. Conclusiones

Podemos ver como los resultados de las distintas versiones del algoritmo AM dan resultados bastante similares, tanto en porcentaje de acierto como de reducción. Además, tienen tiempos de ejecuciones muy similares. Las versiones de AM que sólo realizan la búsqeuda local sobre el 10 % de la población son algo más rápidas debido a que la búsqueda local no se ejecuta sobre toda la población. Aún así, no hay una gran mejora en los resultados del clasificador.

Comparado con el algoritmo SFS, de nuevo SFS es el que obtiene el mayor porcentaje de reducción. Respecto a los porcentajes de aciertos, los de AM son similares o mejores que los del algoritmo SFS, excepto en el conjunto de datos Arrythmia. Sin embargo, los tiempos de AM son más bajos que los de SFS.

Como conclusión, podemos ver que las distintas implementaciones del algoritmo AM son bastante similares respecto a los resultados, aunque las que no ejecutan una búsqueda local sobre toda la población tienen unos tiempos algo más reducidos. De nuevo, si queremos un clasificador con una gran capacidad de reducción el algoritmo SFS sería el más apropiado, además es el que da mejores resultados en tiempo excepto con el conjunto Arrythmia, donde las variantes de AM dan resultados mejores.