Práctica 2: Búsquedas con trayectorias múltiples

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

8 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

Veamos los esquemas de cada algoritmo en pseudocódigo:

- Búsqueda Multiarranque Básica:

```
mejor_solucion = [False,False,...,False]
mejor_tasa = 0
numero_busquedas = 25

Desde 1 hasta numero_busquedas
```

```
        caracteristicas_seleccionadas, tasa = LS(datos_entrenamiento,
            etiquetas_entrenamiento, clasificador)

        Si tasa > mejor_tasa:
            mejor_tasa = tasa
            mejor_solucion = caracteristicas_seleccionadas

    return mejor_solucion, mejor_tasa
```

- GRASP:

```
mejor_solucion = [False,False,...,False]
mejor_tasa = 0
numero_busquedas = 25

Desde 1 hasta numero_busquedas
    caracteristicas_seleccionadas, tasa = SFSrandom(datos_entrenamiento,
        etiquetas_entrenamiento, clasificador)

    # Lanzamos una Local Search con la solución encontrada
    # con SFSrandom como solución inicial
    caracteristicas_seleccionadas, tasa = LS(datos_entrenamiento,
        etiquetas_entrenamiento, clasificador, caracteristicas_seleccionadas)

    Si tasa > mejor_tasa:
        mejor_tasa = tasa
        mejor_solucion = caracteristicas_seleccionadas

return mejor_solucion, mejor_tasa
```

- Iterated Local Search:

```
solucion_inicial = solucionAleatoria()
mejor_solucion = solucion_inicial
num_searchs = 25
mejor_tasa = 0

caracteristicas_seleccionadas, _ = LS(datos_entrenamiento,
    etiquetas_entrenamiento, clasificador, solucion_inicial)

Desde 1 hasta numero_busquedas-1
    # Lanzamos una Local Search con la solución
    # anterior mutada como solución inicial
    nuevas_caracteristicas, nueva_tasa = LS(datos_entrenamiento,
        etiquetas_entrenamiento, clasificador,
         mutacion(caracteristicas_seleccionadas))

    # Como la búsqueda local nunca empeora la solución, no tenemos
    # que comparar la anterior con la nueva,
    # comparamos directamente con la mejor
```

```
            Si nueva_tasa > mejor_tasa:
                mejor_tasa = nueva_tasa
                mejor_solucion = nuevas_caracteristicas

        return mejor_solucion, mejor_tasa
```

Para el GRASP se ha tenido que implementar una versión aleatorizada del SFS, la cual mostramos aquí:

```
caracteristicas_seleccionadas = [False,False,...,False]
mejor_tasa_temporal = 0
peor_tasa_temporal = 0
mejor_caracteristica = 0
mejor_tasa = 0
alpha = 0.3

Mientras mejor_caracteristica no sea None
    mejor_caracteristica = None

    caracteristicas_disponibles = Índices de caracteristicas de
      caracteristicas_seleccionadas que están a False
    tasas = [0,0,...,0]
    caracteristicas_restringidas = ListaVacía

    # Enumeración devuelve las características con su índice en el vector
    # 0 -> caracteristica 0
    # 1 -> caracteristica 1
    # ...
    Para indice,caracteristica en enumeración(caracteristicas_disponibles)

        tasas[indice] = coste al añadir caracteristica a las características seleccion

        Si tasas[indice] > mejor_tasa_temporal
            mejor_tasa_temporal = tasas[indice]
        Si no tasas[indice] < peor_tasa_temporal
            peor_tasa_temporal = tasas[indice]

    Para indice,caracteristica en enumeración(caracteristicas_disponibles)
        Si tasas[indice] > umbral
            caracteristicas_restringidas.añadir(caracteristica)

    caracteristica_aleatoria = aleatorio de caracteristicas_restringidas

    tasa = coste al añadir caracteristica_aleatoria a las características seleccionada

    Si tasa > mejor_tasa
        mejor_tasa = tasa
        mejor_caracteristica = caracteristica_aleatoria

return caracteristicas_seleccionadas, mejor_tasa
```

Para el ILS el operador de mutación es darle la vuelta al 10 % de los bits de la máscara. Se ha implementado así:

```
cambios = EnteroPorArriba(0.1 * longitud(caracteristicas))
mascara = vector aleatorio con "cambios" True y el resto False
caracteristicas_mutadas = XOR(caracteristicas,mascara)
return caracteristicas_mutadas
```

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.

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

- *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.
- -GRASP: Ejecuta el GRASP.
- -ILS: Ejecuta la Iterated Local Search

```
$ 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 -BMB -GRASP -ILS
```

Se ejecutarán BMB, GRASP e ILS 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

BMB

	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	98.23944	97.19298	26.66667	1.10589	73.33333	71.11111	47.77778	4.60245	70.83333	64.43299	44.96403	76.24205
Partición 1-2	97.89474	94.71831	46.66667	1.09142	72.77778	78.33333	58.88889	4.23419	69.58763	61.45833	47.84173	57.93331
Partición 2-1	97.53521	96.49123	36.66667	1.10377	80.55556	68.33333	57.77778	4.08022	70.3125	61.34021	49.64029	71.62338
Partición 2-2	97.89474	95.42254	43.33333	1.13489	74.44444	75.55556	47.77778	4.2708	70.61856	67.1875	46.76259	63.90443
Partición 3-1	98.23944	97.19298	20.0	1.03411	83.33333	67.77778	45.55556	4.40656	73.4375	63.91753	51.07914	72.72391
Partición 3-2	97.19298	95.77465	36.66667	1.08983	75.55556	76.66667	46.66667	4.12107	69.58763	65.625	46.40288	50.08824
Partición 4-1	97.88732	94.38596	30.0	0.98266	77.22222	71.11111	45.55556	4.29649	72.39583	65.46392	50.0	63.14757
Partición 4-2	98.24561	94.71831	46.66667	1.09845	76.11111	75.0	50.0	4.12791	72.16495	60.41667	43.16547	57.0331
Partición 5-1	98.23944	94.73684	43.33333	1.04432	75.55556	77.77778	43.33333	4.36412	69.79167	66.49485	49.28058	70.82738
Partición 5-2	97.89474	96.12676	30.0	0.99273	77.77778	77.77778	36.66667	4.46058	71.64948	67.1875	48.56115	61.47275
Media	97.92637	95.67606	36.00000	1.06781	76.66667	73.94445	48.00000	4.29644	71.03791	64.35245	47.76979	64.49961

GRASP

	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	98.94366	95.4386	73.33333	3.88926	79.44444	70.0	77.77778	16.1053	83.85417	70.61856	90.28777	33.56126
Partición 1-2	98.24561	94.01408	63.33333	4.17635	80.0	78.88889	83.33333	17.94241	79.38144	65.625	89.56835	34.73786
Partición 2-1	97.88732	95.4386	63.33333	3.44124	80.0	68.88889	83.33333	17.76505	84.89583	72.16495	92.80576	38.06696
Partición 2-2	97.89474	95.42254	63.33333	4.26887	78.33333	75.55556	84.44444	15.40904	80.41237	65.10417	90.28777	31.92918
Partición 3-1	97.88732	95.78947	60.0	3.7611	80.0	76.66667	82.22222	15.99317	82.8125	70.10309	93.88489	32.04193
Partición 3-2	98.94737	96.47887	66.66667	3.68251	80.55556	70.0	84.44444	16.34544	82.98969	75.52083	94.60432	37.01289
Partición 4-1	98.59155	97.19298	53.33333	4.31722	81.66667	72.22222	80.0	17.02402	83.85417	76.80412	94.60432	31.20414
Partición 4-2	97.89474	90.84507	73.33333	3.61313	79.44444	74.44444	74.44444	17.89669	82.47423	70.3125	90.64748	38.34605
Partición 5-1	98.94366	94.38596	53.33333	4.25316	81.11111	73.88889	75.55556	17.44564	79.16667	67.52577	94.60432	27.53237
Partición 5-2	97.54386	97.53521	66.66667	3.87753	78.33333	76.66667	77.77778	17.08545	83.50515	77.08333	93.88489	30.5054
Media	98.27798	95.25414	63.66667	3.92804	79.88889	73.72222	80.33333	16.90122	82.33462	71.08623	92.51799	33.49380

ILS

	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	98.23944	95.08772	33.33333	0.7342	72.77778	71.66667	32.22222	3.7392	70.3125	64.43299	47.48201	65.25784
Partición 1-2	98.24561	95.07042	26.66667	0.72968	73.88889	83.88889	30.0	3.6977	72.16495	63.54167	46.04317	53.2398
Partición 2-1	98.59155	96.49123	26.66667	0.64179	74.44444	75.55556	33.33333	3.27899	71.35417	61.85567	44.2446	65.07589
Partición 2-2	97.89474	94.71831	30.0	0.87536	78.88889	70.55556	31.11111	3.46639	72.16495	66.66667	39.20863	51.47445
Partición 3-1	98.59155	95.78947	36.66667	0.79232	74.44444	74.44444	34.44444	3.39178	69.27083	60.82474	41.00719	66.90349
Partición 3-2	97.19298	95.77465	20.0	0.76906	78.88889	72.22222	35.55556	3.36413	70.61856	63.02083	36.69065	45.84797
Partición 4-1	97.53521	96.49123	36.66667	0.76827	79.44444	67.22222	32.22222	3.14595	72.91667	63.40206	45.68345	58.95558
Partición 4-2	98.24561	95.42254	30.0	0.70802	77.22222	72.77778	36.66667	4.04606	69.07216	61.97917	47.48201	43.58709
Partición 5-1	98.94366	95.4386	43.33333	0.99296	76.11111	70.0	38.88889	3.89511	69.27083	61.34021	40.64748	65.45638
Partición 5-2	98.24561	95.77465	26.66667	0.6954	73.88889	75.0	24.44444	2.98533	73.71134	61.97917	43.52518	53.22385
Media	98.17260	95.60588	31.00000	0.77071	76.00000	73.33333	32.88889	3.50106	71.08570	62.90432	43.20144	56.90223

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
BMB	97.92637	95.67606	36.00000	1.08608	76.66667	71.16667	48.00000	4.38071	71.03791	65.49077	47.76979	65.61655
GRASP	98.27798	95.25414	63.66667	4.05062	79.88889	70.72222	80.33333	17.57407	82.33462	71.49646	92.51799	34.47605
ILS	98.17260	95.60588	31.00000	0.80009	76.00000	70.38889	32.88889	3.61344	71.08570	64.09472	43.20144	57.19147

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 tres metaheurísticas implementadas. La BMB funciona bastante bien sobre estas tres bases de datos. En *Wdbc* tiene la segunda mejor tasa de clasficiación en el conjunto de test, con una tasa de reducción del 36 %, lo cual no es nada despreciable. En *Movement Libras* es la metaheurística que consigue la mejor tasa de clasificación en el conjunto de test, con una reducción de casi el 50 % de las características. En estas dos bases de datos el tiempo que tarda es muy pequeño, ni siquiera 5 segundos. No es así en el caso de *Arrhythmia*, donde tarda más de un minuto y se queda a mitad de la tabla en cuanto a tasa de acierto, aunque reduce en casi la mitad también el número de características.

Ahora vamos con el GRASP. Es el algoritmo con mejor tasa de acierto dentro de la muestra en las tres bases de datos. Fuera de la muestra se comporta bastante bien, sobre todo en las bases de datos grandes: en Arrhythmia consigue una grandísima tasa de acierto de más del 71 % con un 92 % de reducción de las características, tardando además sólo 34 segundos en esta gran base de datos. En las pequeñas también tiene una tasa de reducción bastante notable.

Por último, el ILS. Sólo en *Arrhythmia* se comporta peor que cualquier otro fuera de la muestra, pero en las tres bases de datos es la que menos reduce el número de características.

Posiblemente, el primer algoritmo que probaría de estos 3 en este problema sería un GRASP. Tasa de acierto muy competitiva, tasa de reducción casi insuperable y unos tiempos de ejecución muy razonables.

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/