Práctica 1: Búsquedas con trayectorias simples

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

5 de abril 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 una solución
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 Local:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
```

```
Mientras no fin && evaluaciones < MAXIMO_EVALUACIONES
  lista_vecinos = aleatoriza(caracteristicas)
  Para cada caracteristica de lista_vecinos
    Si está activa
      Continuar
    Si evaluaciones >= MAXIMO_EVALUACIONES
      Break

    flip(caracteristicas, indice_caracteristica)
    tasa = coste(caracteristicas)
    flip(caracteristicas, indice_caracteristica)

    si tasa > mejor_tasa:
      mejor_tasa = tasa
      flip(caracteristicas,indice_caracteristica)
      Break
  Si no he terminado debido a un Break:
    fin = True
```

- Enfriamiento Simulado:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
mejor_solucion = caracteristicas
tasa_actual = mejor_tasa

Inicializar valores de temperatura y demás parámetros

Mientras temperatura >= temperatura_final && evaluaciones < MAXIMO_EVALUACIONES
    vecinos_aceptados = 0
    Para cada vecino en 0..max_vecinos
      Si vecinos_aceptados >= max_aceptados
        Break

      caracteristica = aleatorio()
      flip(caracteristicas, caracteristica)
      nueva_tasa = coste(caracteristicas)

      delta = tasa_actual - nueva_tasa

      Si delta != 0 && (delta < 0 || aleatorio < exp(-delta/temperatura))
        vecinos_aceptados++
        tasa_actual = nueva_tasa
        Si tasa_actual > mejor_tasa
          mejor_tasa = nueva_tasa
          mejor_solucion = caracteristicas
      Si no
        flip(caracteristicas, caracteristica)
```

```
    temperatura = actualizar(temperatura)
```

- Búsqueda Tabú:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
mejor_solucion = caracteristicas
tasa_actual = mejor_tasa

Inicializar valores de lista tabú y demás parámetros

Mientras evaluaciones < MAXIMO_EVALUACIONES
  tasa_actual = 0
  mejor_caracteristica = -1

  vecindario = generaAleatorio(tamaño_vecindario)

  Para cada vecino en vecindario
    flip(caracteristicas, vecino)
    nueva_tasa = coste(caracteristicas)
    flip(caracteristicas, vecino)

    evaluaciones++

    Si vecino está en lista_tabú
      Si nueva_tasa > mejor_tasa && nueva_tasa > tasa_actual
        tasa_actual = nueva_tasa
        mejor_caracteristica = vecino
    Else Si nueva_tasa > tasa_actual
      tasa_actual = nueva_tasa
      mejor_caracteristica = vecino

    Si evaluaciones >= MAXIMO_EVALUACIONES
      Break

    posicion_lista_tabu = (posicion_lista_tabu + 1) % tamaño_lista_tabu
    lista_tabu[posicion_lista_tabu] = mejor_caracteristica

    Si tasa_actual > mejor_tasa
      mejor_tasa = tasa_actual
      flip(caraceristicas, mejor_caracteristica)
```

- Búsqueda Tabú extendida:

```
caracteristicas = generaSolAleatoria()
fin = Falso
mejor_tasa = coste(caracteristicas)
evaluaciones = 0
```

```
mejor_solucion = caracteristicas
tasa_actual = mejor_tasa
iteraciones_sin_mejora = 0

Inicializar valores de lista tabú y demás parámetros

Mientras evaluaciones < MAXIMO_EVALUACIONES
  tasa_actual = 0
  mejor_caracteristica = -1

  if iteraciones_sin_mejora >= 10:
    iteraciones_sin_mejora = 0
    eleccion = generaAleatorio()
    Si eleccion < 0.25
      caracteristicas = generaSolAleatoria()
    Si eleccion < 0.5
      caracteristicas = mejor_solucion
    Si no
      total_sols = sum(frec)
      caracteristicas = generaVectorFrecuencias()

    eleccion = generaAleatorio()
    Si eleccion < 0.5
      tamaño_lista_tabu *= 1.5
    Si no
      tamaño_lista_tabu /= 1.5
    tamaño_lista_tabu = parteEnteraPorEncima(tamaño_lista_tabu)
    tabu_list = lista(-1,tamaño_lista_tabu)

  vecindario = generaAleatorio(tamaño_vecindario)

  Para cada vecino en vecindario
    flip(caracteristicas, vecino)
    nueva_tasa = coste(caracteristicas)
    flip(caracteristicas, vecino)

    evaluaciones++

    Si vecino está en lista_tabú
      Si nueva_tasa > mejor_tasa && nueva_tasa > tasa_actual
        tasa_actual = nueva_tasa
        mejor_caracteristica = vecino
    Else Si nueva_tasa > tasa_actual
      tasa_actual = nueva_tasa
      mejor_caracteristica = vecino

    Si evaluaciones >= MAXIMO_EVALUACIONES
      Break

    posicion_lista_tabu = (posicion_lista_tabu + 1) % tamaño_lista_tabu
    lista_tabu[posicion_lista_tabu] = mejor_caracteristica
```

```
Si tasa_actual > mejor_tasa
  mejor_tasa = tasa_actual
  flip(caraceristicas, mejor_caracteristica)
  mejor_solucion = caracteristicas
```

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 = (1,2,...,n)
caracteristicas_seleccionadas = (0,0,...,0,0)
fin = falso
mejor_caracteristica = 0


Mientras mejor_caracteristica != -1
  mejor_tasa = 0
  mejor_caracteristica = -1
  Para cada característica
    tasa = coste(característica)
    Si tasa > mejor_tasa
      mejor_tasa = tasa
      mejor_caracteristica = caracteristica
  Si mejor_caracteristica != -1
    caracteristicas_seleccionadas.añadir(mejor_caracteristica)
```

5. Desarrollo de la práctica

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.

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, tanto las particiones estratificadas de la validación cruzada 5x2 como las de *Leave One Out* para calcular la función de coste. También se ha tomado un clasificador KNN, ya que está implementado usando estructuras de datos complejas como *Ball Tree* y lo hace muy rápido.

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* y *Scikit-Learn*. 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.

```
$ python practica1.py -TS
```

Se ejecutará la Tabu Search. Pero no sólo se limita el programa a un algoritmo. Si le pasamos varios, los ejecutará en paralelo con el criterio de ejecutar tantos procesos como el mínimo entre los algoritmos pasados por argumento y el número de CPU's del sistema. Así, cada algoritmo podrá ir a una CPU distinta sin interferir en el rendimiento de ninguno de ellos.

```
$ python practica1.py -SFS -LS -SA
```

Se ejecutarán en paralelo SFS, LS y SA paralelamente. Si se tuvieran dos núcleos, se ejecutarían los dos primeros y el tercero esperaría, para así no lastrar el rendimiento de ambos.

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 ($\phi = \mu = 0{,}3$, los valores de vecinos máximos, soluciones máximas aceptadas, etc).

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	23.75914	75.0	61.11111	88.88889	83.25275	77.08333	69.07216	98.92086	109.22605
Partición 1-2	97.54386	93.66197	86.66667	19.95866	80.55556	73.88889	81.11111	131.8161	75.25773	67.70833	97.48201	220.29433
Partición 2-1	95.42254	91.22807	83.33333	23.5695	67.22222	57.77778	92.22222	61.30044	78.125	70.61856	97.84173	186.84346
Partición 2-2	95.78947	91.90141	93.33333	12.28126	76.11111	73.88889	92.22222	61.0694	79.38144	75.0	96.04317	326.04926
Partición 3-1	96.12676	92.98246	90.0	16.11261	80.0	68.88889	87.77778	91.08489	77.60417	70.61856	98.92086	107.04789
Partición 3-2	97.54386	96.47887	86.66667	20.00407	70.0	63.88889	91.11111	70.31223	80.41237	73.95833	96.40288	297.28291
Partición 4-1	98.23943	96.49123	86.66667	19.77167	73.33333	65.0	91.11111	69.40039	76.5625	71.64948	98.92086	107.34481
Partición 4-2	94.73684	94.3662	90.0	16.1619	69.44444	65.0	91.11111	68.592	81.4433	76.5625	97.1223	243.54894
Partición 5-1	96.47887	92.63158	90.0	16.1582	73.33333	55.55556	94.44444	46.21479	72.91667	66.49485	99.28058	79.62791
Partición 5-2	98.94737	93.66197	76.66667	30.27225	62.77778	53.33333	93.33333	54.1916	77.83505	68.75	98.56115	134.51725
Media	96.83642	93.56845	86.66667	19.80493	72.77778	63.83333	90.33333	73.72346	77.66216	71.04328	97.94964	181.17828

LS

	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	97.88733	95.08772	43.33333	2.78482	71.66666	66.66667	41.11111	13.14446	65.625	68.04124	49.64029	41.85221
Partición 1-2	98.24561	96.47887	16.66667	3.0734	70.55556	80.0	52.22222	17.15517	67.01031	64.0625	46.76259	37.49884
Partición 2-1	96.47887	97.54386	40.0	3.39736	69.44444	71.11111	48.88889	8.06262	66.66667	59.79381	50.0	45.0822
Partición 2-2	98.24561	94.3662	43.33333	3.48262	69.44444	75.55556	47.77778	5.76119	67.01031	63.02083	44.96403	29.12605
Partición 3-1	98.23943	93.33333	43.33333	3.53381	68.88889	66.11111	47.77778	5.42649	66.14583	66.49485	46.40288	70.93236
Partición 3-2	95.78947	97.1831	43.33333	3.88581	67.77778	71.11111	43.33333	4.16039	70.10309	66.14583	47.84173	53.18718
Partición 4-1	96.83099	96.14035	33.33333	2.40023	71.11111	67.22222	43.33333	4.07424	70.3125	69.07216	48.20144	25.208
Partición 4-2	97.19298	96.47887	33.33333	3.65401	70.0	66.66667	36.66667	4.39171	67.52577	64.0625	48.92086	80.78107
Partición 5-1	96.47887	96.14035	50.0	5.34267	72.77778	72.22222	44.44444	4.84611	73.4375	65.46392	40.64748	61.97022
Partición 5-2	97.89473	94.01408	40.0	4.88932	76.11111	76.11111	46.66667	11.53286	66.49485	64.0625	42.08633	37.40543
Media	97.32839	95.67667	38.66666	3.64440	70.77778	71.27778	45.22222	7.85552	68.03318	65.02201	46.54676	48.30436

SA

	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	96.49123	40.0	100.29172	70.0	67.22222	46.66667	213.54383	71.35417	68.04124	53.59712	855.73377
Partición 1-2	97.89473	96.83099	63.33333	95.93822	70.55556	77.77778	51.11111	212.27381	67.52577	62.5	43.16547	894.88636
Partición 2-1	97.53521	94.38596	50.0	97.93278	73.88889	63.33333	48.88889	212.83528	69.27083	65.97938	53.59712	824.82599
Partición 2-2	96.49123	95.07042	36.66667	98.17512	72.22222	73.88889	47.77778	213.95732	69.58763	64.58333	52.51799	850.47581
Partición 3-1	97.1831	96.14035	26.66667	98.46561	75.0	71.11111	53.33333	211.40456	70.3125	66.49485	45.68345	1016.30123
Partición 3-2	97.19298	95.42254	40.0	98.32049	69.44444	71.11111	46.66667	210.78197	73.19587	64.0625	50.0	1001.46568
Partición 4-1	98.94366	93.33333	46.66667	97.33434	73.33333	73.88889	46.66667	212.77572	70.3125	63.91753	49.64029	948.78478
Partición 4-2	96.14035	96.83099	30.0	113.81931	65.55555	74.44444	55.55556	211.13621	68.5567	60.9375	46.76259	979.51487
Partición 5-1	96.83099	94.38596	46.66667	110.18032	71.66666	65.0	38.88889	216.25208	68.75	64.43299	52.8777	851.86418
Partición 5-2	98.24561	96.12676	50.0	109.40455	72.77778	74.44444	48.88889	191.48629	70.10309	69.27083	56.11511	919.57164
Media	97.39931	95.50185	43.00000	101.98625	71.44444	71.22222	48.44445	210.64471	69.89691	65.02201	50.39568	914.34243

TS

	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	95.08772	50.0	2265.52611	72.22222	64.44444	50.0	1412.43625	65.625	67.01031	51.43885	1905.89863
Partición 1-2	97.19298	93.30986	53.33333	2241.31689	73.33333	75.55556	67.77778	1357.15518	70.10309	63.02083	56.11511	1889.47549
Partición 2-1	98.59155	91.22807	63.33333	2174.18065	72.77778	67.22222	47.77778	1415.82773	68.75	65.97938	52.8777	1878.14355
Partición 2-2	95.78947	96.83099	46.66667	2230.53801	78.33334	68.33333	55.55556	1391.32474	70.61856	67.1875	50.71942	2012.20162
Partición 3-1	97.1831	97.19298	40.0	2218.62374	71.11111	73.33333	52.22222	1404.90508	70.3125	65.46392	49.64029	2003.8009
Partición 3-2	97.54386	94.3662	53.33333	2167.24572	63.33333	74.44444	58.88889	1380.78229	70.10309	63.02083	50.0	2036.88396
Partición 4-1	96.47887	96.84211	43.33333	2193.77008	69.44444	68.33333	52.22222	1404.57579	69.27083	64.43299	47.48201	2056.87424
Partición 4-2	97.54386	95.77465	43.33333	2201.88719	66.11111	69.44444	38.88889	1442.75849	71.64949	69.79167	50.71942	1962.32756
Partición 5-1	97.1831	97.19298	46.66667	2191.31027	68.33333	68.33333	63.33333	1369.07262	67.70833	62.37113	52.8777	1901.45506
Partición 5-2	99.29824	96.47887	53.33333	2180.73688	73.88889	72.77778	60.0	1377.44638	71.64949	64.0625	50.35971	1964.19788
Media	97.50445	95.43044	49.33333	2206.51355	70.88889	70.22222	54.66667	1395.62846	69.57904	65.23411	51.22302	1961.12589

El caso de la búsqueda tabú extendida no lo tengo completo: Por alguna razón, en la 5º ejecución del algoritmo en la base de datos *Arrhythmia* la ejecución me ofrecía un *Memory Error*, después de más de 12 horas ejecutando.

7. Referencias

Las referencias utilizadas han sido:

- *Scikit-Learn*: La propia documentación[1] de la biblioteca.
- *SciPy*: La propia documentación[2] de la biblioteca.

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