Práctica 1: Búsquedas con trayectorias simples

Antonio Álvarez Caballero
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.

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.

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

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.

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 ou
```

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

Búsqueda Local

	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

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	23.6847	75.0	61.11111	88.88889	86.85704	77.08333	69.07216	98.92086	113.45124
Partición 1-2	97.54386	93.66197	86.66667	21.27275	80.55556	73.88889	81.11111	133.14515	75.25773	67.70833	97.48201	221.32613
Partición 2-1	95.42254	91.22807	83.33333	24.21075	67.22222	57.77778	92.22222	60.01379	78.125	70.61856	97.84173	195.53327
Partición 2-2	95.78947	91.90141	93.33333	12.81063	76.11111	73.88889	92.22222	59.83782	79.38144	75.0	96.04317	325.50824
Partición 3-1	96.12676	92.98246	90.0	16.16634	80.0	68.88889	87.77778	87.70363	77.60417	70.61856	98.92086	106.32187
Partición 3-2	97.54386	96.47887	86.66667	20.40403	70.0	63.88889	91.11111	67.70326	80.41237	73.95833	96.40288	292.26432
Partición 4-1	98.23943	96.49123	86.66667	19.97793	73.33333	65.0	91.11111	68.92823	76.5625	71.64948	98.92086	105.00263
Partición 4-2	94.73684	94.3662	90.0	16.57442	69.44444	65.0	91.11111	73.90462	81.4433	76.5625	97.1223	243.91016
Partición 5-1	96.47887	92.63158	90.0	16.81309	73.33333	55.55556	94.44444	45.33103	72.91667	66.49485	99.28058	78.50098
Partición 5-2	98.94737	93.66197	76.66667	31.40311	62.77778	53.33333	93.33333	57.74021	77.83505	68.75	98.56115	134.39024
Media	96.83642	93.56845	86.66667	20.33178	72.77778	63.83333	90.33333	74.11648	77.66216	71.04328	97.94964	181.62091

7. Referencias