Práctica 1.b

Búsqueda por Trayectorias

Selección de características

Algoritmos considerados: SFS,LS,SA,BT básico

Luis Suárez Lloréns
DNI: 75570369-M
luissuarez@correo.ugr.es
5º Doble Grado Ingeniería Informática y Matemáticas
Grupo de Prácticas: 3

Índice

1. Descripción del problema

Cuando se trata un problema de clasificación o de aprendizaje automático, nunca sabemos a priori los datos que nos serán útiles. Es más, añadir datos innecesarios puede incluso empeorar el rendimiento de nuestro clasificador.

El fin del problema de selección de características es tratar de tomar un conjunto de datos de calidad, que nos permita afrontar el posterior aprendizaje de una manera más rápida y con menos ruido en los datos.

Pese a no ser este un problema directamente de clasificación, vamos a necesitarla para valorar la calidad de una solución del problema. Por tanto, necesitamos un clasificador sencillo para esta tarea. Utilizaremos el clasificador k-nn — para ser más concreto, 3-nn —, y trataremos de encontrar las características con las que mejor clasifique un conjunto de prueba.

Entonces, usando el clasificador 3-nn, nuestro objetivo va a ser maximizar la función:

$$\frac{Instancias\, bien\, clasificadas}{Total\, de\, instancias}$$

2. Consideraciones generales

En esta sección veremos los componentes en común de los diferentes algoritmos.

- Representación: Array binario con la misma longitud que el número de datos.
- Función objetivo: Porcentaje de acierto del clasificador 3-nn. Para evaluarlo Tendríamos que hacer lo siguiente:
  - Tomar las columnas que nos indique la solución.
  - Entrenar el clasificador con los datos de entrenamiento y sus etiquetas.
  - Clasificar los datos de test y comprobar si coinciden con sus verdaderas etiquetas.

  Además, para poder ver lo bien que clasifica al propio conjunto de entrenamiento, realizamos "Leave One Out", que consiste en, para cada dato del conjunto de entrenamiento, quitarlo de los datos de entrenamiento, clasificarlo y ver si hemos acertado al clasificar o no.
- Generación de vecindario: El vecindario serán las soluciones que solo difieran de la actual un bit. La generación del vecino i-ésimo podría realizarse de la siguiente forma: Si el valor en la posición i es 1, ponerlo a 0.Si no, ponerlo a 1.
- Uno de los parámetros de los algoritmos a la hora de ejecutarlos es la solución inicial. Esto nos permitirá utilizar estos mismos métodos para la realización de búsqueda multiarranque, por ejemplo. Por tanto, el calculo de una solución de inicio aleatoria se encuentra fuera de los algoritmos.

3. Explicación de los algoritmos

3.1. BL

Para generar de manera aleatoria los vecinos de una solución dada, realizamos lo siguiente:

- Crear una lista de números de 0 al número de características del problema
- Reordenar aleatoriamente dicha lista

La lista reordenada después se recorre, y modificando el elemento indicado de la solución de partida, obtenemos los vecinos.
Búsqueda local:

- Hasta que no encontremos mejora en el bucle interno o superemos el número máximo de iteraciones, repetir:
- Para cada vecino, ordenados aleatoriamente —bucle interno—
    - Calcular la función objetivo.
    - Si mejora la función objetivo de la solución actual, pasa a ser la solución actual y termina el búcle interno.

3.2. ES

Temperatura inicial:

- Calculamos la función objetivo de la solución inicial
- Realizamos la operación $\frac{\mu * puntuacion}{-log(\phi)}$

Enfriamiento:

- Al inicio del programa, calculamos β con la fórmula dada.
- Para enfriar, realizamos $t_{k+1} = \frac{t_k}{1+\beta t_k}$

Enfriamiento simulado:

- Hasta que no encontremos mejora en el bucle interno, superemos el número máximo de iteraciones o la temperatura sea menor que la temperatura final, repetir:
- Poner a 0 el número de soluciones aceptadas en este enfriamiento y hacer el número máximo de iteraciones en el bucle interno lo siguiente:
  - Si has aceptado las suficientes, termina el bucle
  - Si no, toma un vecino aleatorio y calcula su función objetivo
  - Si es el mejor hasta ahora, tomalo como mejor solución y como siguiente solución y aumenta en uno el número de soluciones aceptadas.
  - Si no es el mejor, y tiene el mismo valor que el actual, pasamos al siguiente vecino —Se hace así para evitar ciclos infinitos que nos impedían salir por la condición "no encontramos mejora—
  - Si no, si aleatoriamente el enfriamiento nos permite aceptar la solución, la consideramos como siguiente solución y aumentamos el número de soluciones aceptadas.
- Calculamos enfriamiento

3.3. BT simple

Manejo de la lista tabú:

- Elimina el primer elemento
- Añade el nuevo elemento al final

Búsqueda Tabú:

- Hasta que superemos el número máximo de iteraciones, repetir:
- Para cada uno de los 30 vecinos generados aleatoriamente:
    - Calcular la función objetivo
    - Si es mejor que la mejor solución hasta el momento
        - Considerarlo la mejor solución global y el mejor vecino del grupo
        - Pasar a la siguiente vecino
    - Si es mejor que el mejor vecino hasta el momento y no está en la lista tabú, considerarlo el mejor vecino del grupo
- Modificamos la solución actual al mejor vecino encontrado en el grupo y modificamos la lista tabú

4. Algoritmo de comparación

El algoritmo de comparación es el algoritmo greedy SFS, que consiste en:

- Partimos de la solución completamente a 0.
- Hasta que no encontremos mejora, realizar:
  - Para cada bit que sea 0, ponerlo a uno y calcular la función objetivo.
  - Tomamos la mejor de todas, y si mejora a la solución que teníamos, hacemos permanente el cambio y seguimos iterando.

5. Procedimiento

Para la realización de las prácticas, he usado el lenguaje Python 3 y varios paquetes adicionales. El uso de estos paquetes, en especial scikit, es tanto por comodidad como por eficiencia, esto último de vital importancia para las prácticas, pues el lenguaje Python suele ser lento.

Para la función objetivo —k-nn— y para separar los datos, ya sea para formar los conjuntos de test y entrenamiento o para Leave One Out, utilizamos el paquete scikit. Este paquete incorpora todo lo necesario para hacer, de forma automática, toda la parte de aprendizaje del problema.

Para la realización de los algoritmos, se utilizó Python 3 de manera directa, basandose en los códigos de la asignatura. Con el fin de poder empezar la ejecución del programa desde una partición intermedia, cada partición tiene una seed asociada en vez de usarse una única seed para todo el fichero. Las seeds son, por orden: $12345678, 90123456, 78901234, 456789012, 34567890$.

Para usar el programa, hay que ejecutar la orden python3 main.py BaseDatos Heurística Semilla. Si no se introduce semilla, se utilizan las usadas para obtener los resultados.

6. Resultados

Para los resultados he usado todos los parámetros dados en la práctica menos el número de iteraciones máximo, que he reducido a 5000 por problemas de tiempo.

SFS:

	Wdbc			Movement_Libras			Arrhythmia		
	% clas	*% red*	T	*% clas*	*% red*	T	*% clas*	*% red*	T
Partición 1-1	97,54	86,67	40,02	79,44	91,11	145,26	77,60	98,56	276,36
Partición 1-2	97,54	90,00	32,53	78,33	91,11	145,49	75,26	97,48	448,97
Partición 2-1	95,77	90,00	32,05	80,00	90,00	160,01	68,75	99,28	167,46
Partición 2-2	97,19	83,33	46,76	80,56	92,22	129,37	77,32	97,84	389,97
Partición 3-1	97,18	86,67	39,68	83,89	91,11	145,12	74,48	97,84	381,53
Partición 3-2	96,84	80,00	53,82	75,00	87,78	189,60	76,29	98,56	274,74
Partición 4-1	96,13	90,00	32,23	76,67	90,00	159,10	74,48	97,84	382,05
Partición 4-2	98,95	80,00	53,47	80,56	87,78	188,03	83,51	97,48	434,82
Partición 5-1	97,18	90,00	31,98	72,78	92,22	127,34	85,94	96,04	638,46
Partición 5-2	97,54	86,67	39,75	78,89	92,22	127,70	79,90	96,76	553,53
Media	97,19	86,33	40,23	78,61	90,56	151,70	77,35	97,77	394,79

Búsqueda local:

	Wdbc			Movement_Libras			Arrhythmia		
	% clas	*% red*	T	*% clas*	*% red*	T	*% clas*	*% red*	T
Partición 1-1	97,89	46,67	24,72	72,78	48,89	43,77	69,79	50,00	105,96
Partición 1-2	98,25	53,33	16,99	75,00	51,11	23,89	68,56	47,48	210,64
Partición 2-1	96,83	56,67	11,47	81,11	44,44	40,74	69,79	47,48	199,97
Partición 2-2	96,84	56,67	9,21	77,22	48,89	31,35	72,68	50,72	314,39
Partición 3-1	97,18	40,00	16,33	82,78	46,67	73,62	70,83	54,68	130,42
Partición 3-2	96,84	50,00	10,21	76,67	64,44	36,86	72,16	54,68	164,92
Partición 4-1	96,83	43,33	26,85	75,56	50,00	33,07	70,83	48,56	136,27
Partición 4-2	98,25	50,00	16,60	79,44	37,78	28,80	70,10	60,79	220,97
Partición 5-1	96,83	50,00	17,15	70,56	55,56	22,39	67,71	53,96	258,49
Partición 5-2	97,89	43,33	20,09	78,33	51,11	32,15	70,62	55,76	117,08
Media	97,36	49,00	16,96	76,94	49,89	36,66	70,31	52,41	185,91

Enfriamiento simulado:

	Wdbc			Movement_Libras			Arrhythmia		
	% clas	*% red*	T	*% clas*	*% red*	T	*% clas*	*% red*	T
Partición 1-1	97,18	53,33	101,63	72,22	54,44	209,14	73,96	53,24	829,94
Partición 1-2	97,19	50,00	103,81	75,00	53,33	209,32	64,95	49,64	789,90
Partición 2-1	97,54	36,67	105,04	77,22	42,22	214,48	65,63	50,72	778,82
Partición 2-2	97,54	40,00	102,67	76,67	53,33	215,73	73,20	56,47	770,50
Partición 3-1	97,89	46,67	196,18	82,78	51,11	401,32	67,71	50,00	768,08
Partición 3-2	97,19	56,67	102,89	73,89	58,89	209,43	68,04	53,24	802,52
Partición 4-1	95,77	60,00	194,40	72,78	51,11	207,45	66,15	46,40	795,08
Partición 4-2	98,25	60,00	193,65	77,22	45,56	402,77	66,49	61,51	788,50
Partición 5-1	96,13	46,67	195,92	71,67	47,78	209,76	65,10	53,24	775,46
Partición 5-2	96,84	53,33	193,53	77,22	51,11	211,62	68,04	55,76	767,92
Media	97,15	50,33	148,97	75,67	50,89	249,10	67,93	53,02	786,67

Búsqueda tabú:

	Wdbc			Movement_Libras			Arrhythmia		
	% clas	*% red*	T	*% clas*	*% red*	T	*% clas*	*% red*	T
Partición 1-1	98,94	46,67	1522,08	78,33	52,22	1025,90	75,00	49,28	1286,71
Partición 1-2	98,95	53,33	1505,77	81,67	60,00	1016,52	71,65	50,72	1288,07
Partición 2-1	98,94	50,00	1591,13	85,56	55,56	1009,39	72,40	48,92	1267,58
Partición 2-2	98,60	53,33	1581,08	85,00	60,00	1017,46	70,10	50,36	1310,06
Partición 3-1	98,59	46,67	1578,85	87,22	50,00	1008,53	73,96	57,55	1325,55
Partición 3-2	98,95	60,00	1621,55	78,89	52,22	1033,35	72,68	53,96	1378,55
Partición 4-1	97,89	40,00	1560,35	80,56	52,22	1014,97	69,79	51,44	1300,25
Partición 4-2	99,65	53,33	1531,12	80,56	45,56	1006,37	74,23	58,99	1266,84
Partición 5-1	98,24	60,00	1575,49	78,89	62,22	1015,79	69,79	51,80	1370,68
Partición 5-2	98,60	56,67	1606,21	82,22	53,33	1023,21	78,35	57,91	1224,24
Media	98,73	52,00	1567,36	81,89	54,33	1017,15	72,79	53,09	1301,85

Total:

	Wdbc			Movement_Libras			Arrhythmia		
	% clas	*% red*	T	*% clas*	*% red*	T	*% clas*	*% red*	T
3-NN	96,06	0,00	0,00	76.77	0,00	0,00	63,82	0,00	0,00
SFS	97,19	86,33	40,23	78,61	90,56	151,70	77,35	97,77	394,79
BL	97,36	49,00	16,96	76,94	49,89	36,66	70,31	52,41	185,91
ES	97,15	50,33	148,97	75,67	50,89	249,10	67,93	53,02	786,67
BT básica	98,73	52,00	1567,36	81,89	54,33	1017,15	72,79	53,09	1301,85
BT extendida	x	x	x	x	x	x	x	x	x

Lo primero que podemos ver es que, con respecto al 3-NN conseguimos mejorar la clasificación en general, reduciendo el espacio de características a tener en cuenta. Esto es algo que podríamos esperar, pues muchos de esos datos podrían no tener importancia para el problema y ser fuente de ruido para clasificar. Por tanto, esto ya es de gran utilidad para el problema de clasificación.

En cuanto a las heurísticas, destacar primero la gran velocidad y el buen trabajo que hace la búsqueda local. Obtiene resultados buenos en muy poco tiempo comparado con los demás. Esto la hace atractiva para mejorar soluciones puntuales de manera muy rápida. Además, las soluciones de partida eran aleatorias, con posiciones de partida buenas — obtenidas por otros procesos de búsqueda— tiene que ser necesariamente aún más rápida. Esto lo seguiremos estudiando en las siguientes prácticas.

Por otro lado, el enfriamiento simulado no ha rendido como se esperaba. Sabiendo su funcionamiento, debería por lo menos igualar, si no mejorar, a la búsqueda local. Sin embargo, los resultados obtenidos son peores. Esto pue-

de ser debido a que los parámetros no estén bien ajustados, probablemente por el cambio al máximo de 5000 iteraciones, pues los parámetros estaban ajustados para 15000.

La búsqueda Tabú gana en 2 de 3 bases de datos, y queda segunda en la otra. Advertir que esto se consigue en parte por ser la que más soluciones consigue explorar, esto lo podemos ver claramente en el tiempo empleado. Esto es así por ser la única de las búsquedas implementadas que siempre ejecuta el máximo de las iteraciones.

Destacar por último el tema de la reducción. Vemos como todas las heurísticas se quedan en torno al 50 %. Esto ya es una gran mejora, pero puede que haya momentos donde queramos quedarnos con menos datos aún. Para conseguir hacer esto, podríamos cambiar la función objetivo para que penalice el número de características que utiliza, pero con la versión de la función objetivo de la práctica, esto no se controla.

En resumen, viendo estos resultados, si necesitamos una búsqueda rápida, usaremos una búsqueda local. Para una búsqueda más profunda usando caminos, usaríamos la búsqueda tabú, a la que aún le podríamos hacer mejoras.

7. Referencias

Aparte de la documentación de la asignatura, he usado las páginas de referencia del software usado para desarrollar las prácticas:

- Python: `https://docs.python.org/3/`
- Numpy y Scipy: `http://docs.scipy.org/doc/`
- Scikit-learn: `http://scikit-learn.org/stable/documentation.html`