Práctica 1.b

Búsqueda por Trayectorias

Selección de características

Algoritmos considerados: BMB,Grasp,SA,ILS

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 verdadero, ponerlo a falso.Si no, ponerlo a verdadero.
- 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.
- **Generación de soluciones aleatorias:** Se guardan en la máscara los resultados de muestrear una binomial que nos devuelve valores verdadero y falso con la misma probabilidad.

2.1. 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. Explicación de los algoritmos

3.1. BMB

Búsqueda multiarranque básico

- Realizar 25 veces:
  - Calcular solución aleatoria.
  - Realizar búsqueda local a la solución.
  - Si es mejor que la mejor solución hasta el momento, guardarla como nueva mejor solución
- Devuelve la mejor solución encontrada y el valor de su función objetivo.

3.2. GRASP

ASFS:

- La solución empieza completamente a falso.
- Mientras que encontremos mejora:
  - Creamos un vector para guardar los valores de la función objetivo
  - Inicializamos valores para guardar los máximos y mínimos encontrados.
  - Para cada característica, si no es verdadera:
    - Guardar su valor de función objetivo y posición.
    - Actualizar el mejor y peor valor de la función objetivo si fuera necesario.
  - Calculamos el mínimo aceptado como $M - t(M - m)$, siendo M el máximo valor de la función objetivo, m el mínimo y t la tolerancia admitida.
  - Tomamos los vecinos que superen el mínimo y elegimos uno aleatoriamente.
  - Si el vecino supera a la solución anterior, se toma como siguiente solución y se marca que se ha realizado mejora.
- Devuelve la solución encontrada.

GRASP:

- Realizar 25 veces:
  - Calcular solución inicial con ASFS.
  - Realizar búsqueda local a la solución.
  - Si es mejor que la mejor solución hasta el momento, guardarla como nueva mejor solución
- Devuelve la mejor solución encontrada y el valor de su función objetivo.

3.3. ILS

Función de mutación:

- Tomamos s elementos sin remplazamiento de los números enteros desde el 0 hasta el número de características del problema, sin contar este último.
- Modificamos los valores seleccionados en la solución.

Iterated Local Search:

- Realizamos la búsqueda local sobre la solución inicial.
- Guardamos el resultado como mejor función objetivo encontrada y mejor solución encontrada.
- Realizamos 24 veces:
  - Muta la solución.
  - Realiza una búsqueda local sobre la solución mutada.
  - Si la solución encontrada por la búsqueda local mejora la mejor hasta el momento, la sustituye.
- Devuelve la mejor solución encontrada y el valor de su función objetivo.

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.

Usamos scikit para la creación de particiones y para normalizar los datos.

Para el uso del clasificador 3-nn, tanto para el cálculo del acierto del test como para Leave One Out, utilizamos una implementación en CUDA realizada por Alejandro García Montoro, pues la mejora de tiempo es sustancial con respecto al k-nn implementado en scikit, que sólo usa la CPU del ordenador.

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

Mostramos los resultados.

Cuadro 1: KNN

	Wdbc				Libras				Arrhythmia			
	% train	% test	% red	tiempo	% train	% test	% red	tiempo	% train	% test	% red	tiempo
P 1-1	96.13	96.14	0.0	0.02	66.67	70.0	0.0	0.04	62.5	65.98	0.0	0.14
P 1-2	96.84	95.77	0.0	0.02	65.56	85.56	0.0	0.04	61.86	61.46	0.0	0.12
P 2-1	96.83	95.79	0.0	0.02	75.0	69.44	0.0	0.04	64.58	63.4	0.0	0.13
P 2-2	95.44	96.13	0.0	0.02	71.67	75.56	0.0	0.04	65.46	63.02	0.0	0.12
P 3-1	97.18	96.49	0.0	0.02	75.0	74.44	0.0	0.04	61.98	61.86	0.0	0.13
P 3-2	97.54	94.72	0.0	0.02	68.89	75.0	0.0	0.04	64.43	65.1	0.0	0.12
P 4-1	95.42	97.54	0.0	0.02	65.56	71.67	0.0	0.04	64.06	63.92	0.0	0.13
P 4-2	97.54	95.42	0.0	0.02	68.33	73.33	0.0	0.04	60.82	64.06	0.0	0.12
P 5-1	95.42	95.79	0.0	0.02	62.78	72.78	0.0	0.04	62.5	65.98	0.0	0.13
P 5-2	96.14	96.83	0.0	0.02	69.44	76.67	0.0	0.04	65.46	60.42	0.0	0.12
Medias	96.44	96.06	0.0	0.02	68.89	74.44	0.0	0.04	63.36	63.52	0.0	0.12

Cuadro 2: SFS

	Wdbc				Libras				Arrhythmia			
	% train	% test	% red	tiempo	% train	% test	% red	tiempo	% train	% test	% red	tiempo
P 1-1	95.77	95.09	50.0	0.26	67.78	67.22	48.89	1.38	66.67	65.46	48.56	76.73
P 1-2	97.19	94.72	46.67	0.27	70.0	79.44	50.0	1.37	67.01	64.06	46.04	33.5
P 2-1	96.83	94.74	53.33	0.39	75.56	68.89	40.0	2.13	68.23	63.92	44.24	56.79
P 2-2	97.54	95.07	43.33	0.51	73.89	73.89	42.22	3.58	69.07	63.54	50.36	66.12
P 3-1	95.77	97.19	40.0	0.25	76.11	76.11	44.44	2.85	69.79	60.31	53.24	47.85
P 3-2	97.54	95.07	33.33	0.74	74.44	73.89	60.0	3.25	66.49	64.06	50.36	33.19
P 4-1	96.48	96.84	36.67	0.49	65.56	73.89	53.33	1.36	66.15	60.82	46.4	47.68
P 4-2	98.6	95.77	43.33	0.64	72.78	72.78	35.56	3.62	68.04	67.19	58.27	47.3
P 5-1	96.13	94.74	43.33	0.39	68.33	72.78	54.44	2.69	66.15	62.37	50.36	67.2
P 5-2	96.84	96.13	56.67	0.5	73.89	75.0	50.0	1.35	72.16	64.58	57.55	39.65
Medias	96.86	95.53	44.66	0.44	71.83	73.38	47.88	2.35	67.97	63.63	50.53	51.60

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`
- K-nn CUDA: `https://github.com/agarciamontoro/metaheuristics`