PRÁCTICA 3:

Enfriamiento Simulado, Búsqueda Local Reiterada y Evolución Diferencial para el Problema del Aprendizaje de Pesos en Características

Curso 2017/2018

Práctica realizada por:

Nombre: Santiago Vidal Martínez
Email: sanviflo@correo.ugr.es
Grupo 1 (Lunes, 17:30-19:30)
DNI: 77138012Z

CONTENIDO:

PRÁCTICA 3

Descripción del problema

El problema del APC consiste en conseguir asignarle unos pesos a las características de nuestros datos para que el sistema nos determine la característica que más va a influir a la hora de clasificar estos según los datos de entrenamiento.

Es decir, es importante tener en cuenta, que para algunas clasificaciones no todas las características son importantes. Para ver esto, basta con pensar en la forma de clasificar si un paciente tiene anemia. Los datos posibles serían los datos de una analítica, los cuales, indicarán mejor con la característica de la cantidad de hierro en sangre que los leucocitos.

Por lo tanto, nuestro algoritmo ha de calcular cuál de esas características influye más a la hora de clasificar un dato, de esta manera, nuestro clasificador será más óptimo. Se tendrán en cuenta las siguientes fórmulas para calcular la bondad de nuestro vector de pesos:

$$F(W) = \alpha\, tasa-clas(W) + (1-\alpha) tas-red(W)$$

$$tasa-clas = 100 \cdot \frac{nº\ instancias\ bien\ clasificadas\ en\ T}{nº\ instancias\ en\ T}$$

$$tasa-red = 100 \cdot \frac{nº\ valores\ w_i < 0.2}{nº\ características}$$

Sabiendo que cada elemento de la solución (W) está normalizada en el intervalo [0,1], el clasificador será el 1-NN, es decir, genero todos los vecinos y elijo el vecino más cercano. También tendremos en cuenta que T es el conjunto de datos con el que se evalúa el clasificador y α es la importancia entre el acierto y la reducción de la solución, que estará dentro del intervalo [0,1].

La forma de calcular el vector de peso será mediante diferentes algoritmos:

- El RELIEF será uno de ellos, donde se irá calculando mediante un Greedy el vector de pesos.

- El algoritmo de búsqueda local, consistirá en ir generando vecinos e ir seleccionando siempre los vecinos más cercanos.

- Los algoritmos genéticos con algunas de sus variantes (BLX,AC,generacional,estacionario) cuya idea básica será seleccionar los padres de una población (conjunto de soluciones) y cruzarlos para generar hijos (nuevas soluciones) que pasarán a ser parte de la población nueva.

- Los algoritmos meméticos con las variantes propuestas, donde se hibridará el algoritmo genético generacional y la búsqueda local. Cada n número de generaciones, se buscará el vecino más cercano de todos los elementos de la población.

Descripción de la aplicación de algoritmos

Al desarrollar el clasificador 1-NN, tenemos un número n de datos que poseen un número c de características. Para calcular el valor con el que clasifica un dato, aplicaremos la distancia euclídea, en la que se ponderará debidamente con los pesos que cualquiera de nuestros algoritmos implementados haya generado.

Por ejemplo, si tenemos dos características, en la que la primera tiene un peso de 1 y la segunda tiene otro peso de valor 0, la segunda característica no influirá en nada en la distancia euclídea y la primera será utilizada seguro para clasificar nuestros datos.

Por lo tanto, para poder calcular la distancia euclídea, requerimos un algoritmo que nos calcule los pesos, por lo que emplearemos alguno de los algoritmos que hemos implementado: RELIEF, 1-NN, ES, ILS, DE/rand/1 y DE/current-to-best/1

La representación de las soluciones será un vector de pesos, es decir, un vector con la longitud del número de características que tienen los datos. El dominio de los pesos estará entre 0 y 1. Para el valor 0, la característica no tiene ninguna importancia, por el contrario, el valor 1 significa que la característica es imprescindible para clasificar.

De esta manera, nuestra función objetivo será maximizar el valor de la función que se calcula a partir de la tasa de clasificación y la tasa de reducción. A continuación muestro todos los pseudocódigos de las funciones que he implementado.

Cálculo de la función objetivo

alfa = 0.5

Calculo la tasa de reducción y almaceno en t_red

Calculo la tasa de clasificación y almaceno en t_clas

Devuelvo t_red*alfa + (1-alfa)*t_clas

Cálculo de la tasa de reducción

Cuento pesos menores que 0.2 en la solución y almaceno en número

Calculo el tamaño del vector solución y almaceno en tamaño

Devuelvo (número/tamaño)*100

Cálculo de la tasa de clasificación:

Cuento los datos bien clasificados y almaceno en número

Calculo el tamaño del vector solución y almaceno en tamaño

Devuelvo (número/tamaño)*100

Buscar enemigo

Posiciono i en el primer enemigo

Calculo la distancia al primer enemigo y almaceno en distancia

Calculo la clase del primer enemigo y almaceno en clase

Asigno a elemento el primer enemigo

Para j = i hasta len(datos)

Si datos[j] es enemigo y está a menor distancia que datos[i]

Asigno a distancia la distancia a datos[j]

Asigno a clase la clase de datos[j]

Asigno datos[j] a elemento

Devuelvo elemento

Buscar amigo

Posiciono i en el primer amigo

Calculo la distancia al primer amigo y almaceno en distancia

Calculo la clase del primer amigo y almaceno en clase

Asigno a elemento el primer amigo

Para j = i hasta len(datos)

Si datos[j] es amigo y está a menor distancia que datos[i]

Asigno a distancia la distancia a datos[j]

Asigno a clase la clase de datos[j]

Asigno datos[j] a elemento

Devuelvo elemento

Cabe destacar que amigos significa elementos con la misma clasificación y enemigos elementos que tienen diferente clasificación

Algoritmo de Búsqueda Local

Genero una solución aleatoria y la almaceno en Solución inicial

Asigno Solución inicial a Solución actual

Asigno Solución actual a Solución mejor

Mientras no se haya evaluado15000 veces y se encuentre nueva solución en menos de 20.n iteraciones

Genero un nuevo vecino y lo asigno a vecino

Asigno vecino a Solución actual

Calculo la función de evaluación y la asigno a f

Si f de Solucion actual > f de Solucion mejor

Asigno a Solucion mejor la Solución actual

Generar vecino

Genero valor manteniendo la distribución normal y se lo asigno a v

Asigno solucion a s

Sumo v a una característica de s

Trunco al intervalo [0,1]

Devuelvo s

Generar solución aleatoria

Genero la solución de forma aleatoria dentro del intervalo

[0,1] y lo devuelvo

Generación de soluciones aleatorias

Población = [ ]

Desde i = 0 hasta n

Genero solución aleatoria y almaceno en solución

Creo un individuo y lo almaceno en ind (al crear un individuo lo evalúa y almacena ese valor)

Añado ind a Población

Devuelvo Población

Algoritmo de enfriamiento simulado

Genero solución aleatoria y almaceno en sol

Calculo temperatura inicial y almaceno en T

Evalúo con función objetivo y almaceno en coste

Desde i=0 hasta M iteraciones

Desde j=1 hasta num_características

Genero un vecino y almaceno en s

Evalúo s y almaceno en coste2

Si coste - coste2 < 0 o número aleatorio generado < $e^{(-(coste - coste2)/T)}$

Actualizo s con la nueva solución

Si el coste de la nueva s es mejor que el mejor coste

Actualizo la mejor solución

Finaliza el segundo for si exitos >= max_exitos

Enfrío la temperatura T

Finalizo primer bucle si exitos = 0

Devuelvo la mejor solución

Temperatura inicial de enfriamiento simulado

T = (0.3*coste)/(-np.log(0.3))

Esquema de enfriamiento de enfriamiento simulado

Genero nueva temperatura : T/(1+(T_inicial-T_final)/15000*T_inicial*T_final)*T

Modifico T por la nueva temperatura generada

Algoritmo ILS

Genero solución y la almaceno en sol

Evalúo la solución y la almaceno en coste

Mejor_sol = sol

Desde i = 0 hasta 15

Genero solución nueva con BL y almaceno en sol_nueva

Evalúo sol_nueva y almaceno el valor en coste2

Si coste2 es el mejor coste

Mejor_sol = sol_nueva

Muto la mejor solución

Devuelvo la mejor solución

Mutación ILS

lista = []

Desde i=0 hasta 10% de características

Introduzco en lista un valor aleatorio entre 0 y numero de características

Modifico características con los indices generados en lista por valores aleatorios

Algoritmo de Evolución Diferencial

- **Rand**

  Genera una población y lo almaceno en población

  Evalúo todos los elementos de la población

  Mientras el número de evaluaciones < 15000

  Desde i = 0 hasta el tamaño de la población

  Escojo 3 padres aleatorios (sin repetir) y almaceno en padre1,padre2 y padre3

  Si rand < 0.5

  offspring[k] = padre1[k] + 0.5*(padre2[k] -padre3[k])

  Sino

  offspring[k] = poblacion[i][k]

  Si el nuevo individuo es mejor que el actual individuo

  Sustituye el individuo por el nuevo

  Si el nuevo individuo es mejor que el mejor individuo

  Sustituyo el mejor individuo por el nuevo individuo

- **Current**

  El pseudocódigo de current es el mismo pero cambiando la función para generar un nuevo individuo(offspring).

  Si rand < 0.5

```
        offspring[k] = poblacion[i][k] + 0.5*(mejor_sol[k] - poblacion[i][k]) + 0.5*(padre1[k] -
        padre2[k])
    Sino
        offspring[k] = poblacion[i][k]
```

Procedimiento de desarrollo de la práctica

En esta práctica he partido de cero, no he utilizado nada de lo que se nos ha facilitado. Para realizar la práctica he utilizado el lenguaje Python y el entorno de desarrollo ha sido spyder.

El sistema operativo que he utilizado para realizar el desarrollo ha sido Windows 10.

Para utilizarlo, se deberá comentar todos los vectores, y descomentar el del fichero que queramos ejecutar. En caso de querer utilizar otro fichero, bastará con descomentar uno de los vectores del programa principal y cambiar el nombre del fichero. Se procederá a hacer el mismo proceso en caso de querer ejecutar diferentes algoritmos.

Experimentos y Análisis de datos

Al realizar la ejecución de los algoritmos las tablas que han resultado con los datos obtenidos son las siguientes:

Tablas generadas

Adjunto también en este documento los resultados globales.

	Ozone				Parkinsons				Spectf-heart			
	%_clas	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T	*%_clas*	*%red*	*Agr.*	T
1-NN	**72,45**	**0,00**	**36,22**	**0,00**	**74,64**	**0,00**	**37,32**	**0,00**	**66,61**	**0,00**	**33,30**	**0,00**
RELIEF	**76,88**	**29,01**	**51,51**	**1,05**	**74,64**	**5,71**	**40,18**	**0,38**	**69,50**	**11,16**	**40,33**	**1,21**
ES	**68,32**	**96,34**	**82,33**	**15338,85**	**74,60**	**97,14**	**85,87**	**2147,41**	**70,51**	**97,67**	**84,09**	**1162,99**
ILS	**87,17**	**75,21**	**81,19**	**1217,60**	**87,49**	**90,48**	**88,98**	**69,81**	**87,08**	**80,00**	**83,54**	**742,58**
DE/rand/1	**93,74**	**86,76**	**90,25**	**1908,91**	**94,24**	**94,29**	**94,29**	**699,65**	**91,90**	**93,02**	**92,46**	**2357,30**
DE/current-to-best/1	**87,78**	**74,08**	**80,93**	**1896,90**	**92,15**	**87,62**	**89,88**	**696,71**	**89,07**	**73,02**	**81,05**	**2378,94**

Se puede observar que ES, ILS y ambos DE dan buenos resultados con un valor de la función objetivo por encima del 80%. En el caso de ES se puede observar que es el algoritmo que más tarda, y en nuestros casos, la tasa de clasificación de este es peor que en el resto de algoritmos. La tasa de reducción sin embargo, es mejor en ES que en el resto.

El algoritmo ILS es el algoritmo que menos tarda de los cuatro mencionados anteriormente, sin embargo es el que peor función de evaluación tiene en la mayoría de casos. Se puede ver que para el conjunto de datos más pequeño, ILS funciona mucho mejor.

Por último nos encontramos ambos algoritmos de evolución diferencial. En este caso se puede observar que el rand tiene mejores resultados que DE current-to-best. Sin embargo, esto tiene un coste extra en tiempo aunque no es demasiado, solo ha tardado más current-to-best para Spectf. El DE rand es el mejor en valor de evaluación que cualquiera de los otros y en tiempo, no es el mejor, pero tiene un tiempo considerablemente bueno para obtener una buena solución.

En nuestro problema, me quedaría sin duda con evolución diferencial rand.

Referencias bibliográficas

Para la práctica, se ha consultado exclusivamente el material proporcionado en clase (seminarios, transparencias de teoría...)