PRÁCTICA 2:

Técnicas de Búsqueda Local , basadas en Poblaciones y Algoritmos Greedy 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:

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,BL,AGG,AGE y AM

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. El pseudocódigo que nos calcula la función objetivo es la siguiente:

Para los algoritmos genéticos y meméticos, he requerido hacer una nueva estructura de datos. En este caso ha sido una clase individuo que almacenará tanto el vector de pesos como el valor que da la función global. Esto lo he realizado de esta manera ya que a la hora de generar una población, se inicializa la función y no tengo que recalcularla en cada iteración sino que se calcula y solo habría que consultar el valor que tiene la W que se está evaluando. Así tarda menos tiempo a la hora de ejecutar el programa. Sin embargo, al devolver la W resultante, mantendrá la representación de la solución utilizada para los algoritmos RELIEF y BL.

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

Algoritmo Greedy

W = [0,0,...,0]

Desde i = 0 hasta len(datos)

busco el enemigo más cercano de datos[i] y lo almaceno en e

busco el amigo más cercano de datos[i] y lo almaceno en a

Desde c = 0 hasta len(W)

W[c] = W[c] + | datos[i][c] - e[c] | - | datos[i][c] - a[c] |

w_m = max(W)

Para j = 0 hasta len(W)

si el elemento es menor que 0

W[j] = 0

si no

W[j] = W[j] / w_m

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

Selección de AGG

Desde i = 0 hasta el 70% del número de individuos en la población

Elijo un individuo aleatoriamente y lo almaceno en i1

Elijo otro individuo aleatoriamente y lo almaceno en i2

Si el valor de i1 > el valor de i2

Introduzco i1 al conjunto de padres

Si no

Introduzco i2 al conjunto de padres

Selección de AGE - BLX

Desde i = 0 hasta 2

Elijo un individuo aleatoriamente y lo almaceno en i1

Elijo otro individuo aleatoriamente y lo almaceno en i2

Si el valor de i1 > el valor de i2

Introduzco i1 al conjunto de padres

Si no

Introduzco i2 al conjunto de padres

Selección de AGE - AC

Desde i = 0 hasta 4

Elijo un individuo aleatoriamente y lo almaceno en i1

Elijo otro individuo aleatoriamente y lo almaceno en i2

Si el valor de i1 > el valor de i2

Introduzco i1 al conjunto de padres

Si no

Introduzco i2 al conjunto de padres

Cruce de los AGG-BLX

Desde i = 0 hasta len(conjunto de padres)

Cojo el padres[i] y lo almaceno en padre1

Cojo el padres[i + 1] y lo almaceno en padre2

Para j = 0 hasta 2

Para todas las características de los padres

Asigno a c_max el máximo valor de la característica actual entre padre1 y padre2

Asigno a c_min el mínimo valor de la característica actual entre padre1 y padre2

Calculo I = c_max - c_min

En el hijo genero un valor en la característica actual en el intervalo [c_min-I*0.3,c_max+I*0.3] y lo trunco en el intervalo [0,1]

Introduzco el hijo en descendientes

i = i + 2

Cruce de los AGG-AC

descendiente 1 = []
descendiente 2 = []

Para i = 0 hasta len(padres) - 2

Para j = 0 hasta len(padres[i]):

Calculo la media entre padre[i][j] y padre [i + 1][j] y lo almaceno en descendiente 1

Calculo la media entre padre[i + 2][j] y padre [i + 3][j] y lo almaceno en descendiente 2

i = i + 2

Cruce de los AGE-BLX

Cojo el padres[0] y lo almaceno en padre1

Cojo el padres[1] y lo almaceno en padre2

Para j = 0 hasta 2

Para todas las características de los padres

Asigno a c_max el máximo valor de la característica actual entre padre1 y padre2

Asigno a c_min el mínimo valor de la característica actual entre padre1 y padre2

Calculo I = c_max - c_min

En el hijo genero un valor en la característica actual en el intervalo [c_min-I*0.3,c_max+I*0.3] y lo trunco en el intervalo [0,1]

Introduzco el hijo en descendientes

i = i + 2

Cruce de los AGE-AC

descendiente 1 = []
descendiente 2 = []

Calculo la media entre padre[0][j] y padre [i1][j] y lo almaceno en descendiente 1

Calculo la media entre padre[2][j] y padre [3][j] y lo almaceno en descendiente 2

Mutación de los AGs

hijos mutados = []

Desde m = 0 hasta len(hijos)

Para todas las características del individuo

Genero un número aleatorio entre 0 y 1 y lo almaceno en valor

Si valor < 0.001

Genero un valor en la distribución normal, lo sumo al valor de la característica y trunco en el intervalo [0,1] y modifico la característica del hijo

Añado en hijos mutados el hijo

Esquema de evolución de AGG

Partiendo de los hijos ya mutados

Como los hijos son un número total al 70 % de la población, completo estos con el mejor elemento de la población (garantizando elitismo) y con otros elementos aleatorios de la población hasta llegar al número de elementos igual a la población inicial.

Reemplazo la población con el conjunto generado en el paso anterior.

Esquema de evolución de AGE

Partiendo de dos hijos ya mutados

Busco el peor de la población inicial y comparo con el peor de los hijos

Si es mejor el hijo que el peor de la población se sustituye

Repito los dos pasos anteriores para el hijo restante

Esquema de búsqueda de AM(10,1.0) y AM(10,0.1)

Voy contando las generaciones que van pasando.

Cuando llega a un múltiplo de 10 generaciones

Genero un valor aleatorio entre 0 y 1

Si el valor generado es menor que la tasa (1.0 o 0.1)

Calculo soluciones a todos los individuos de la población con BL y los sustituyo

Esquema de evolución de AM(10,0.1mej)

Voy contando las generaciones que van pasando.

Cuando llega a un múltiplo de 10 generaciones

Calculo soluciones con el 10% de los mejores individuos de la población y los sustituyo

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.

ADVERTENCIA: Para realizar la ejecución de los algoritmos he reducido el número de evaluaciones debido a que con 15 mil evaluaciones tardaba como mínimo hora u hora y media en finalizar cada uno de los algoritmos.

Experimentos y Análisis de datos

He ejecutado los algoritmos para 1000 evaluaciones, ya que para un número total de 15000 tardaba cada ejecución 2 horas aproximadamente. Por lo que al ser 7 algoritmos requeriría un total de 14 horas, tiempo que no disponía. Al realizarlo para 1000 evaluaciones obtengo las siguientes tablas:

Tablas generadas

Las tablas nos permiten observar que los algoritmos que menos tardan en ejecutarse sin duda son el de 1-NN y el RELIEF.

Por otro lado, debido a que no he hecho muchas evaluaciones de los algoritmos genéticos no ha encontrado muy buenas soluciones, sin embargo se puede observar entre ellas que el AGG con cruce BLX con alfa 0.3 da mejores soluciones aunque tarda prácticamente lo mismo que AC. Además en el caso de AGE se observa que para BLX además es más rápido.

Es probable que si hubiera realizado los algoritmos para 15000 evaluaciones, habrían salido mejores soluciones.

Por último destacar que tras un rato ejecutando meméticos ha tardado demasiado y no me ha dado tiempo a completar las soluciones de estos. También añadir que rehice la práctica anterior para tratar de solventar todos los errores de esta y por lo tanto también he tenido que volver a ejecutar los algoritmos anteriores para completar desde cero las tablas.

Referencias bibliográficas

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