PRÁCTICA 1B:

Técnicas de Búsqueda Local 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 darle unos pesos a las características de nuestros conjuntos para que el sistema nos determine la característica que más nos va a influir a la hora de clasificar nuestros datos 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 tendrá en cuenta las siguientes fórmulas:

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

$$tasa - clas = 100 \cdot \frac{n^{o}\ instancias\ bien\ clasificadas\ en\ T}{n^{o}\ instancias\ en\ T}$$

$$tasa - red = 100. \frac{n^{o}\ valores\ w_i < 0.2}{n^{o}\ características}$$

Sabiendo que cada elemento de la solución (W) está normalizada en el intervalo [0,1], el clasificador será el 1-NN, es decir, genera un vecino a partir de los pesos de W. 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].

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 nuestro algoritmo (bien sea Greedy o BL) ha generado.

Por ejemplo, si tenemos dos características, 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.

Por lo tanto, para poder calcular la distancia euclídea, requerimos un algoritmo que nos calcule los pesos, por lo que emplearemos dos algoritmos: Greedy y BL.

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:

---

def función_objetivo:
    alfa = 0.5
    Calcula la tasa de reducción → t_red
    Calcula la tasa de clasificación → t_clas
    Devuelve t_red*alfa + (1-alfa)*t_clas

---

def cálculo_tasa_reducción:
    Cuento pesos menores que 0.2 en la solución → número
    Calculo el tamaño del vector solución → tamaño
    Devuelvo (número/tamaño)*100

---

def cálculo_tasa_clasificación:
    Cuento los datos bien clasificados → número
    Calculo el tamaño del vector solución → tamaño
    Devuelvo (número/tamaño)*100

Algoritmo Greedy

El pseudocódigo de este algoritmo es el siguiente:

```
W = [0,0,...,0]
Recorro todos los datos → i
        Para cada dato
                busco el enemigo mas cercano → e
                busco el amigo mas cercano → a
                Para cada caracteristica de W → c
                        W[i] = W[i] + | c[i] - e[i] | - | c[i] - a[i] |
w_m = max(W)
Para cada elemento de W → j
        si el elemento es menor que 0
                w[j] = 0
        si no
                w[j] = w[j]/w_m
```

A continuación mostraré el pseudocódigo de la busqueda de amigos y enemigos de la solución:

```
def buscar_enemigo:
        Me posiciono en el primer enemigo → i
        distancia = distancia al primer enemigo
        clase = clase del primer enemigo
        elemento = primer enemigo
        Recorro los datos
                Para cada dato → date
                        Si el dato es enemigo y está a menor distancia
                                distancia =  distancia a date
                                clase = clase de date
                                elemento = date
```

```
def buscar_amigo:
        Me posiciono en el primer amigo → i
        distancia = distancia al primer amigo
        clase = clase del primer amigo
        elemento = primer amigo
        Recorro los datos
                Para cada dato → date
                        Si el dato es amigo y está a menor distancia
                                distancia =  distancia a date
                                clase = clase de date
                                elemento = date
```

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

Algoritmo de Búsqueda Local

El pseudocódigo del algoritmo de BL es el siguiente:

Solución inicial = Solución aleatoria generada
Solución actual = Solución inicial
Solución mejor = Solución actual

Mientras no se haya iterado 15000 veces y se encuentre nueva solución en menos de 20.n iteraciones
    Genero un nuevo vecino → vecino
    Solución actual = vecino
    Calculo la función de evaluación → f
    Si f de la solucion actual > f de la mejor solucion
        mejor solucion = solución actual

---

def genera_vecino:
    v = genero valor manteniendo la distribución normal
    s = solucion
    Sumo v a una característica de s
    Trunco al intervalo [0,1]
    Devuelvo s

---

def generar_solucion_aleatoria:
    Genero la solución de forma aleatoria dentro del intervalo [0,1]

Descripción de algoritmos de comparación

El pseudocódigo del algoritmo 1-NN es:

cmin = clase del primer elemento del train
dmin = distancia del primer train respecto a

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. Para elegir el fragmento que queremos evaluar, se le asignará a t un valor desde t1 hasta t5. En caso de querer utilizar otro fichero, bastará con descomentar uno de los vectores del programa principal y cambiar el nombre del fichero. Si se quiere utilizar más de un conjunto, se podrá utilizar en t el siguiente formato:

t = t1+t2

Experimentos y Análisis de datos

He ejecutado los algoritmos para sacar los valores de las tablas siguientes:

Tablas generadas

En estas tablas, al ser tan lento el algoritmo de BL lo he ejecutado exclusivamente para un conjunto de datos (Parkinsons.arff)

Se puede observar que la búsqueda local es mucho más lenta que otros algoritmos. Sin embargo, hay algun error en el código de mi programa, ya que, teóricamente deberían ser mejores las soluciones de la búsqueda local.

Se observa además que siempre, o casi siempre, la tasa de clasificación es 100%. Dicho hecho es un tanto extraño pero debido a que he estado revisando mi código y no he logrado solucionarlo, he optado por enviar lo que tenía implementado.

Referencias bibliográficas

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