Metaheurísticas (2018-2019)
Grado en Ingeniería Informática
Universidad de Granada

Práctica 1.b:

Técnicas de Búsqueda Local y Algoritmos Greedy para el Problema del Aprendizaje de Pesos en Características

---

UNIVERSIDAD
DE GRANADA

Antonio Jesús Heredia Castillo
DNI:xxxxxxxxx
a.heredia.castillo@gmail.com
Grupo MH3: Jueves de 17:30h a 19:30h

5 de abril de 2019

Índice

Índice de tablas

1. Descripción del problema

En este problema nos encontramos con un conjunto de datos que previamente han sido clasificados. Estos datos tienen asociado un serie de características, cada una con un valor en concreto. Con estos datos y haciendo uso de diferentes algoritmos y caricaturistas queremos crear un modelo que nos permita clasificar de forma automática cualquier valor nuevo que tengamos. En este caso vamos a comprar el resultado de tres diferentes técnicas.

La primera técnica que usamos es usar un clasificador K-NN, utilizando la técnica de buscar el vecino mas cercano. Este método busca simplemente el elemento que esta mas cercano a el y lo clasifica con la misma etiqueta que el.

La siguiente técnica que vamos a usar es el clasificador RELIEF este algoritmo usando amigos mas cercanos y enemigos mas cercano obtiene un vector de pesos que se puede usar para un clasificador 1-NN para modificar la importancia a la hora de obtener la distancia del vecino mas cercano.

La ultima técnica utilizada es una BL, que modifica los valores del vector de pesos usando el operador de vecino por mutación normal, que consiste en coger un atributo y modificarlo por un valor de la normal.

Todos estas técnicas estarán descritas mas adelante.

2. Descripción de los algoritmos comunes

Para cargar los datos simplemente utilizo **scipy** [1], que tiene una función para cargar directamente los ficheros arff. Como lo carga en formato Pandas, los normalizo con **preprocessing** de **sklearn** [2] y los paso a array de Numpy.

Para los grupos de datos, he usado un $datos[F, C]$ donde F es la cantidad de datos y C la cantidad de atributos que tiene esa clase de datos:

Atributos / Datos	atributo0	atributo1	...	atributoC
dato0				
dato1				
...				
datoF				

2.1. Representación de los datos

La representación para para las etiquetas es igual de sencilla en este caso se representa de la siguiente forma, $vector[F]$, un vector de tamaño F donde cada etiqueta de una posición corresponde al dato que se encuentra en la misma posición en el array de datos:

dato	etiqueta
dato0	a
dato1	b
...	
datoC	a

Para representar los pesos que obtenemos tanto en RELIEF como en Local Search, usaremos tambien un vector, en este caso sera $pesos[C]$, donde C sera el numero de atributos que tienen nuestros datos:

peso	valor
peso0	0.22335
peso1	0.78335
...	
pesoF	0.5436

Por otro lado también tendremos datos estadísticos para valorar las soluciones. Para cada uno de los siguientes datos también existirá su correspondiente media de todas las ejecuciones. Estos son:

- tiempo inicial: Recogido antes de inicial la computación de los datos.
- tiempo final: Cuando acaba la computación de los datos
- tasa acierto: Porcentaje de aciertos que ha tenido una ejecución.
- tasa reducción: Porcentaje de pesos que son inferior a 0.2
- función objetivo: Función que intentaremos maximizar en cada clasificador.

2.2. Ejecución de métodos de búsqueda

La forma de ejecutar todos los algoritmos la realizo de la misma forma. Realizo un separación de los datos usando la técnica de validación cruzada

de 5 iteraciones. Esto consiste en separar los datos de forma uniforme. Para ello estoy usando la biblioteca **StratifiedKFold** de **sklearn**[2]
En el pseudocodigo del algoritmo 2.3 podemos ver como se usa. Lo primero que se realiza es obtener las 5 particiones de datos diferentes. StratifiedKFold nos proporciona **k** grupos de datos. Cada grupo de datos esta formado dos partes una del 80 % y otra del 20 %. Es decir, la parte del test ira variando de un grupo a otro. En el algoritmo se puede ver cuando obtenemos los tiempos, para cada ejecución. También recogemos los datos que usaremos en para realizar las comparaciones.

Algorithm 1 Algoritom para ejecutar los metodos de clasificacion

```
tiempoInicio = getTiempo()
tiempoAnterior = tiempoInicio
particiones = obtenerParticiones(datos, etiquetas,k=5)
for particion en particiones do
    datosEstadisticos = clasificador(particion)
    datosTotales += datosEstadisticos
    tiempoActual = getTiempo()
    tiempoAnterior = tiempoActual
    mostrarDatosPorPantalla(datosEstadisticos,tiempoActual-
tiempoAnterior)
end for
tiempoActual = getTiempo()
mostrarDatosPorPantalla(datosTotales, tiempoActual-tiempoInicio)
```

2.3. Evaluador de soluciones

Para evaluar nuevos datos usando los pesos he usado un algoritmos que me ha cedido mi compañero de clase Antonio Molner Domenec. En el se hace uso de KDDTree, el cual genera un árbol con los vecinos. Realizando así una búsqueda de cual es el mas cercano.

Algorithm 2 Algoritmo para la evaluación de las soluciones

```
X = datos * pesos
X = eliminaDespreciables(X)
árbol = generoArbol(X)
vecinos = arbol.consulta(nuevoDato)
aciertos = media(clase(vecinos) == claseNuevoDato)
reducción = media(pesos ¡0.2)
return aciertos, reducción, (aciertos+reducion)/2
```

3. Descripción de los métodos de búsqueda

3.1. Clasificador 1-NN

Lo primero que hay que explicar en este apartado es el clasificador 1-NN. Ya que es la base. Para este algoritmo tendremos que tener una función que nos devuelva la distancia entre entre dos datos. En nuestro caso usaremos la distancia euclidea. Que viene definida por la siguiente ecuación, donde e_1 y e_2 es un vector con las características de cada elemento:

$$d_e\left(e_1, e_2\right) = \sqrt{\sum_i \left(e_1^i - e_2^i\right)^2}$$

Algorithm 3 Algoritmo 1-NN

```
1: claseMinima = etiquetas.primera
2: distanciaMinima = distancia(datos,datos.primero,nuevoDato)
3: for indice = 0 hasta tamaño(datos) do
4:     distancia = distancia(datos,datos[indice],nuevoDato)
5:     if distancia ¡ distanciaMinima) then
6:         distanciaMinima = distancia
7:         claseMinima = etiquetas[indice]
8:     end if
9: end for
10: return claseMinima
```

Una vez que tenemos la función de distancia lo único que tenemos que realizar es un simple bucle en el que recorramos todos los datos, y busquemos con cual tiene menor distancia el nuevo dato. Cada vez que encontramos una distancia menor la guardamos y esa sera la clase de nuestro nuevo elemento. Al terminar lo único que hacemos es devolver la clase de ese dato.

Una vez que tenemos el algoritmo que clasifica lo único que tendremos que realizar es bucle que vaya pasando nuevos datos al clasificador y comparar la solución que nos da. Si es igual a la etiqueta real del dato, tendremos un acierto mas. Por ultimo se devuelve la media de los datos y listo.

Algorithm 4 Algoritmo para contar datos bien clasificados con 1-NN

```
aciertos = 0
for indice, dato en datos_test do
    if clasificador(datos,etiquetas,test) == etiquetas_test[indice] then
        aciertos++
    end if
end for
return aciertos / tamaño(datos_test)
```

3.2. Relief

El algoritmo **??** de Relief tiene un funcionamiento muy sencillo. Lo primero que realizo es crear una matriz de distancia con sckit-learn[2]. Esto lo realizo para evitar tener que calcularla cada vez que evaluó un dato. Para crear el vector de pesos, tenemos que buscar el amigo y el enemigo mas cercano a cada dato. Para evitar realizar un bucle por cada, realizo una comparacion dentro del for que recorre buscando amigos y enemigos. La primera comparación busca que tengan diferente clase, si es asi y ademas tiene una distancia mejor que el mejor enemigo, ese se convierte en el mejor enemigo. La siguiente comparación busca que tengan la misma etiqueta, pero que sea un dato diferente, ademas miramos si la distancia entre ellos y si es menor, ese dato se convierte en el mejor amigo.
Una vez que hemos encontrado el mejor amigo y enemigo, podemos modificar el vector de pesos según aparece en las transparencias, donde e_i es el dato que estamos comparando y e_e y e_a son mejor enemigo y mejor amigo respectivamente:

$$W = W + |e_i - e_e| - |e_i - e_a|$$

Algorithm 5 Algoritmo RELIEF

```
pesos = vector.zeros(tamaño(datos))
matrizDistancias = crearMatrizDistancias(datos, datos)
for indice, dato en datos do
    mejorEnemigo = inf
    mejorAmigo = inf
    enemigoMasCercano = -1
    amigoMasCercano = -1
    for indice2, dato2 en datos do
        if etiquetas[indice] != etiquetas[indice2] then
            if matrizDistancias[indice,indice2] less mejorEnemigo then
                mejorEnemigo = matrizDistancias[indice,indice2]
                enemigoMasCercano = dato2
            end if
        end if
        if etiquetas[indice] == etiquetas[indice2] and  then
            if matrizDistancias[indice,indice2] less mejorAmigo then
                mejorAmigo = matrizDistancias[indice,indice2]
                amigoMasCercano = dato2
            end if
        end if
    end for
    pesos = pesos + abs(dato-enemigoMasCercano) - abs(dato - amigo-
MasCercano)
end for
maxPeso=maximo(pesos)
for indice en pesos do
    if peso[indice] less 0.2  then
        peso[indice] = 0
    else
        peso[indice] = peso[indice] / maxPeso
    end if
end for
return pesos
```

3.3. Búsqueda local

Este algoritmo de búsqueda local se basa en realizar mutaciones aleatorias en los pesos del modelo. Para esto tiene un máximo de iteraciones de 15000 o que no se haya modificado el vector completo por 20 veces. Las mutaciones

la realiza a cada atributo por separado y si existe una mejora, se queda con ese nuevo peso, si no lo mejora vuelve a los pesos anteriores. La mutación se realiza en base a la Normal centrada en 0 y el valor se coge de forma aleatoria. Este algoritmo tarda mucho mas en ejecutarse que los demás ya que tiene que realizar muchas mas evaluaciones de la solución y esto es un proceso costoso.

Algorithm 6 Algoritmo de busqueda local

```
1: pesos = generarPesosAleatorio()
2: mejorEstadisticos = evaluar(pesos,datos,etiquetas)
3: cantCaractarteristicas = tamaño(peso)
4: cantEjecuciones = 0
5: cantExplorados = 0
6: while cantEjecuciones less MAX and cantExplorados less (20*cantCa-
   ractarteristicas) do
7:     for indice hasta cantCaractarteristicas do
8:         cantExplorados += 1
9:         cantEjecuciones += 1
10:        pesosactual = copia(pesos)
11:        Z = random.normal(0.0, pow(0.3,2))
12:        pesosactual[indice] += Z
13:        if pesosactual[indice] great 1  then
14:            pesosactual[y]=1
15:        else if pesosactual[indice]less0 then :
16:            pesosactual[y]=0
17:        end if
18:        if  pesosactual[indice] != W[y] and pesos[indice] great 0.2 then
19:            estadisticos = evaluar(pesos,datos,etiquetas)
20:            if  estadisticos great mejorEstadisticos then
21:                mejorEstadisticos = estadisticos
22:                pesos = copia(pesosactual)
23:                cantExplorados = 0
24:                break
25:            end if
26:        end if
27:    end for
28: end while
29: return pesos
```

Una vez que el algoritmo ha terminado de ejecutar se evaluara el resultado de los pesos generado con los datos de train y se obtendrán los datos

estadísticos que se muestran.

4. Experimentos

Para los experimentos que he utilizado he usado una semilla con valor 42. El experimento ha consistido en en realizar la validación cruzada de 5 iteraciones en todos los algoritmos y en todos los set de datos. Y con ellos he obtenido los resultado que detallare a continuacion para cada uno de los metodos. Todos los estadisticos estan expresados en porcentaje 100 excepto el tiempo que esta expresado en segundo. He mostrado los datos por separado ya que en una sola tabla no me entraban todos.

4.1. Clasificador 1-NN

Los datos de clasificación y tiempo obtenidos para colposcopy con esta clasificación se pueden ver en la Tabla 1

Tabla 1: Los datos obtenidos para colposcopy en 1-NN

	Tasa clasificacion	Tiempo
Particion 1	90	0.346
Particion 2	94	0.342
Particion 3	90	0.408
Particion 4	91.81818	0.354
Particion 5	94.54545	0.342
Media	92.1818	0.3586

Los datos de clasificación y tiempo obtenidos para texture con esta clasificación se pueden ver en la Tabla 2

Tabla 2: Los datos obtenidos para texture en 1-NN

	Tasa clasificación	Tiempo
Particion 1	77.9661	0.094
Particion 2	73.6842	0.087
Particion 3	70.1754	0.092
Particion 4	75.4385	0.089
Particion 5	71.9298	0.09
Media	73.8388	0.0904

Los datos de clasificación y tiempo obtenidos para ionosphere con esta clasificación se pueden ver en la Tabla 3

Tabla 3: Los datos obtenidos para ionosphere en Relief

	Tasa clasificacion	Tiempo
Particion 1	84.5070	0.279
Particion 2	90	0.133
Particion 3	85.7142	0.129
Particion 4	87.1428	0.131
Particion 5	84.2857	0.131
Media	86.3299	0.16060

4.2. Relief

Los datos estadisticos obtenidos para texture con esta clasificación se pueden ver en la Tabla 7

Tabla 4: Los datos obtenidos para texture en Relief

	Tasa clasificacion	Tasa reducción	Tasa agregación	Tiempo
Particion 1	69.4915	45.1612	57.3264	0.134
Particion 2	82.45614	45.161	63.808	0.102
Particion 3	61.4035	27.4193	44.41143	0.093
Particion 4	63.1578	27.41935	45.2886	0.095
Particion 5	75.438	41.93548	58.6870	0.142
Media	70.3895	37.41935	53.9044	0.1136

Los datos estadisticos obtenidos para ionosphere con esta clasificación se pueden ver en la Tabla 5

Tabla 5: Los datos obtenidos para ionosphere en 1-NN

	Tasa clasificacion	Tasa reducción	Tasa agregación	Tiempo
Particion 1	78.8732	2.9411	40.9072	0.134
Particion 2	78.57142	2.9411	40.7563	0.102
Particion 3	77.1428	2.9411	40.0420	0.093
Particion 4	82.8571	2.9411	42.8991	0.095
Particion 5	81.4257	2.94115	42.1887	0.142
Media	79.7764	2.9411	41.3579	0.1136

Los datos estadisticos obtenidos para colposcopy con esta clasificación se pueden ver en la Tabla 7

Tabla 6: Los datos obtenidos para colposcopy en Relief

	Tasa clasificacion	Tasa reducción	Tasa agregación	Tiempo
Particion 1	92.7272	15	53.8636	0.2435
Particion 2	86.3636	2.5	45.6818	0.2462
Particion 3	83.63636	2.5	43.0681	0.272
Particion 4	84.5454	7.5	43.5227	0.276
Particion 5	85.45454	5	46.4772	0.231
Media	86.5454	6.5	46.5227	0.2636

4.3. Busqueda local

Los datos estadisticos obtenidos para colposcopy con esta clasificación se pueden ver en la Tabla 7

Tabla 7: Los datos obtenidos para texture en Busqueda local

	Tasa clasificacion	Tasa reducción	Tasa agregación	Tiempo
Particion 1	82.72727	22.5	52.61363	344.471
Particion 2	81.8181	27.5	54.6590	367.177
Particion 3	74.5454	47.5	61.0227	441.333
Particion 4	85.4545	37.5	61.4772	405.743
Particion 5	88.1818	55	71.59090	296.004
Media	82.5454	38.0	60.27274	370.9456

Los datos estadisticos obtenidos para ionosphere con esta clasificación se pueden ver en la Tabla 8

Tabla 8: Los datos obtenidos para ionosphere en Busqueda local

	Tasa clasificación	Tasa reducción	Tasa agregación	Tiempo
Particion 1	83.0985	44.1176	63.6081	242.917
Particion 2	82.85714	47.05882	64.9579	356.888
Particion 3	78.5714	35.2941	56.9327	135.301
Particion 4	81.42857	44.117647	62.7731	199.422
Particion 5	85.71428	54.62184	54.62184	635.08
Media	82.3340	2.9411	60.57876	313.9216

Los datos estadisticos obtenidos para texture con esta clasificación se pueden ver en la Tabla 9

Tabla 9: Los datos obtenidos para colposcopy en Busqueda local

	Tasa clasificacion	Tasa reducción	Tasa agregación	Tiempo
Particion 1	67.7966	40.322	54.0595	865.957
Particion 2	75.4385	41.9354	58.6870	292.145
Particion 3	64.9122	41.9354	53.4288	265.035
Particion 4	61.4035	48.3870	54.8953	390.866
Particion 5	64.403	51.6129	58.2625	556.41
Media	66.8926	44.8387	55.865	474.0826

4.4. Resumen

Viendo los datos podemos observar que el que mejor se comporta posiblemente sea el de 1-nn ya que obtiene mas tasa de acierto, y eso se puede a la forma en la que se encuentran nuestros datos. Ya que es posible que las clases están muy separadas entre si y con el vecino mas cercano sea suficiente para predecir la clase a la que pertenece.Ademas al ser un algoritmo lineal es muy rapido de realizar.

Por otro lado tenemos Relief y Busqueda local. Relief es cierto que es mucho mas rápido, pero no obstante los resultado de la búsqueda local son mejores. Esto se debe a que solo se muta si ha mejorado el valor de la tasa de agregación, obtenido así mejores resultado. Ademas con la mutación aleatoria, no cae en ningún mínimo local.

Referencias

[1] SciPy, *Biblioteca open source de herramientas y algoritmos matemáticos para Python*

[2] scikit-learn, *Biblioteca para aprendizaje de máquina de software libre para el lenguaje de programación Python*