Capítulo 4. Implementación del Sistema

En este capítulo se explicarán, algunas de las principales partes del código desarrollado; entre éstas se encuentran el proceso para encontrar los términos multipalabra y la función para encontrar el valor de la información mutua.

El algoritmo para encontrar los términos multipalabra es el siguiente:

a) Inicializar el vector archivo y la tabla hash vocabulario

b) Copiar el vector del archivo original al vector del archivo.

   El archivo original contiene todo el texto plano de la página Web, así como también los signos de puntuación que aparecen en ella.

c) Encontrar el vocabulario del archivo.

   El vocabulario tiene dos columnas, en la primera se encuentran los términos del documento y en la segunda se coloca la frecuencia de ocurrencia de cada término.

d) Encontrar el vocabulario por párrafos.

   Lo que hace este método es encontrar el vocabulario, tomando como unidad de término a una pareja de palabras; es importante aclarar que no se eliminaron los signos de puntuación del archivo, debido a que estos nos proporcionaran información al momento de hacer las parejas de palabras. A continuación se presenta el código para encontrar el vocabulario por párrafos.

```
// Vectores:
// archivo: contiene el texto plano de la página Web
// signos: contiene los signos de puntuación.

// Tabla hash
// vocParr: guarda el bigrama y su frecuencia.
// multipalabra: guarda los términos multipalabra.

Integer valor;
Object val;
String token1="",token2;
boolean existe;
int i=0;

if (archivo.size()>0)
   token1=(String)archivo.elementAt(0);
while((i+1)<archivo.size()){
   if(!(signos.contains(token1))){
      if ((i+1)<archivo.size()){
         i++;
         token2=(String)archivo.elementAt(i);
         if(!(signos.contains(token2))){
            // si no existe en la tabla vocParr agregar (con 1)
            existe=vocParrafos.containsKey(token1+" "+token2);
            if (!existe){
               val=vocParrafos.put(token1+" "+token2, new Integer(1));
            }
            else{
               valor=(Integer)vocParrafos.get(token1+" "+token2);
               val=vocParrafos.remove(token1+" "+token2);
               val=vocParrafos.put(token1+" "+token2,new Integer(1+valor.intValue()));
            }
            token1=token2;    // se trabaja con la siguiente pareja de términos
         }
         else{        // token2 es un signo de puntuación, por lo tanto
         // esta pareja no se toma en cuenta y se trabaja con la
                                    // siguiente pareja de términos
            if ((i+1)<archivo.size()){
               i++;
               token1=(String)archivo.elementAt(i);
            }
            }
         }//if ((i+1)<archivo.size())
      }
      else{         // token1 es un signo de puntuación; por lo tanto, se
                                    // busca otra pareja de términos
         if ((i+1)<archivo.size()){
```

```
            i++;
            token1=(String)archivo.elementAt(i);
          }
       }
}// while
```

e) Crear el archivo de salida, el cual tendrá como primera columna la pareja de palabras; mientras que la segunda columna tendrá la medida de la información mutua de esta pareja de palabras.

f) Revisar el valor de la información mutua de cada pareja de palabras; con el objetivo de reconocer a las parejas que forman un término multipalabra. Una pareja de palabras (x,y) es considerada como término multipalabra [Salazar *et al*, 2006]:

$$\text{si } IM(x,y) > 5 \quad \text{y} \quad fr(x) > 3 \quad \text{y} \quad fr(y) > 3$$

En el caso de ser un término multipalabra, agregarlo a la tabla de términos multipalabra.

```
double MI=5;
double MIrestriccion=3;
Object val;
for (Enumeration claves = vocParrafos.keys() ; claves.hasMoreElements() ;) {
  String c1=(String)claves.nextElement();  // c1 tiene a la pareja de terminus, los
                                           // cuales están separados por un espacio
                                           // en  blanco
  Integer frecClave, frecClave1, frecClave2;

  int pos=c1.indexOf(' ');
  String clave1=c1.substring(0,pos);    // clave1 contiene al primer término de
                                           // la pareja, mientras que clave2 guarda
                                           // al segundo término
  String clave2=c1.substring(pos+1,c1.length());

  frecClave=(Integer)vocParrafos.get(c1);
  frecClave1=(Integer)vocabulario.get(clave1);
  frecClave2=(Integer)vocabulario.get(clave2);

  if((funMI(frecClave, frecClave1, frecClave2)>MI)
                &(frecClave1.intValue()>MIrestriccion)
                &(frecClave2.intValue()>MIrestriccion) ){
                // es multipalabra, agregar a la tabla multipalabra
```

```
    val=multipalabra.put(c1,new Integer(1));
  }
 }//for
```

g) Si se encontraron términos multipalabra, se creará un nuevo archivo el cual se guardará en el vector archivoEti, este archivo contiene todos los términos y signos de puntuación del archivo original, la diferencia que tiene es que los términos multipalabra se unirán por medio del carácter guión bajo (‘_’) y de esta forma, aunque sean 2 o más palabras, éstas serán tomadas como un solo término. Después se inicializa el vector del vocabulario, el vector del vocabulario por párrafos y la tabla hash de los términos multipalabra.

Finalmente, se tomará al archivo etiquetado como el archivo original, se encontrará el vocabulario de los términos multipalabra y de forma recursiva seguirá ejecutándose este proceso.

En caso de no encontrar términos multipalabra, revisar el archivo etiquetado para decidir cuáles son términos multipalabra.

Se quitaron los términos multipalabra que presentaban las siguientes características:

Términos multipalabra que empezaban o terminaban con palabras cerradas

Términos multipalabra de un solo caracter

h) Fin.

La primera versión de este algoritmo no tomaba en cuenta todos los signos de puntuación, únicamente consideraba el punto; por consiguiente, no todos los resultados que arrojaba eran correctos, se revisaron los patrones sintácticos que seguían cada uno de los resultados y se planteó como solución a este problema, eliminar los términos multipalabra que seguían determinados patrones sintácticos.

Para poder aplicar este filtro, se tenía que hacer uso de un etiquetador para encontrar la categoría gramatical de cada término, la desventaja que se generaba al aplicar este recurso era el incremento del tiempo y recordemos que el sistema se usará en tiempo real.

En el anexo B se muestra un documento que se usó como entrada a este módulo, las palabras subrayadas son los términos multipalabra que se encontraron usando una revisión manual; después se muestran los resultados que encontró el programa.

En la segunda versión de este módulo, se tomaron en cuenta los signos de puntuación y los resultados mejoraron ampliamente; por lo tanto, ya no se necesitó como recurso un etiquetador.

El código para calcular la información mutua es el siguiente:

```
public static double funMI(Integer frecClave,Integer frecClave1,Integer frecClave2){
  // el no. de divisiones entre 2 es el logaritmo base 2
  // N representa el tamaño del documento
  double r;

  r=(((N*frecClave.intValue())/(frecClave1.intValue()*frecClave2.intValue()))+1);
  r=Math.log(r)/Math.log(2);
  return r;
}
```

Uno de los problemas que se presentó al cambiar el formato del documento, de HTML a texto plano, fue el uso signos por parte de los creadores de las páginas Web. Para resolver este problema se aplicó un filtro, al documento, para eliminar estos signos.

A continuación se presenta el código para eliminar los signos usados en las páginas Web, los valores indican el código ASCII del carácter:

```
public static void cargaSignosHtml(){
	int j=160;
	int i;

	for(i=0;j<=192 ;i++){
		signosHtml[i]=j;
		j++;
	}
	for(j=194;j<=200 ;i++){
		signosHtml[i]=j;
		j++;
	}
	for(j=202;j<=204 ;i++){
		signosHtml[i]=j;
		j++;
	}
	for(j=206;j<=208 ;i++){
		signosHtml[i]=j;
		j++;
	}
	signosHtml[i]=210;
	i++;
	for(j=212;j<=217 ;i++){
		signosHtml[i]=j;
		j++;
	}
	for(j=219;j<=224 ;i++){
		signosHtml[i]=j;
		j++;
	}
	for(j=226;j<=227 ;i++){
		signosHtml[i]=j;
		j++;
	}
	for(j=229;j<=232 ;i++){
		signosHtml[i]=j;
		j++;
	}
	signosHtml[i]=234;
	i++;
	signosHtml[i]=236;
	i++;
	signosHtml[i]=238;
	i++;

	signosHtml[i]=240;
	i++;
```

```
        signosHtml[i]=242;
        i++;
        signosHtml[i]=244;
        i++;
        signosHtml[i]=245;
        i++;
        for(j=247;j<=249 ;i++){
              signosHtml[i]=j;
              j++;
        }
        signosHtml[i]=251;
        i++;
        for(j=253;j<=255 ;i++){
           signosHtml[i]=j;
           j++;
        }
        signosHtml[i]=34;           // "
        signosHtml[i+1]=38;         // &
        signosHtml[i+2]=93;         // ]
        signosHtml[i+3]=94;         // ^
        signosHtml[i+4]=124;        // |
        signosHtml[i+5]=33;         // !
        signosHtml[i+6]=63;         // ?
        signosHtml[i+7]=42;         // *
        signosHtml[i+8]=92;         // \
        signosHtml[i+9]=24;         // $
        signosHtml[i+10]=91;        // [
        signosHtml[i+11]=43;        // +
}
```