Teoría de la Información y Codificación

Práctica 2: Creación código con distribución de probabilidad asociada

José A. Montenegro Montes

26 de septiembre de 2014

1. Enunciado

La práctica muestra los conceptos teóricos desarrollados en el tema 3.

A la hora de crear una codificación tenemos que tener en cuenta las propiedades que vimos en el tema anterior. La fuente emisora de los símbolos tiene asociada una distribución de probabilidad que la caracteriza. Dicha distribución de probabilidad debe ser tomada en cuenta a la hora de crear los códigos que realizaremos en esta práctica.

En la práctica además calcularemos un indicador, la entropía, que nos permitirá determina el límite inferior y superior de la longitud del código que estamos buscando.

Para el correcto funcionamiento de la práctica, el alumno deberá implementar los siguientes métodos:

- *entropía*. Calcula el valor de la entropía de una distribución dada (página 23 del tema 3).
- *longitudMedia*. Calcula la longitud media de un código (página 23 del tema 3).
- *reglaShannonFano*. Obtiene los parámetros n_i para establecer un código posteriormente con el árbol binario (página 39 del tema 3).

2. Conclusiones

Con esta práctica observamos como crear códigos adaptados a una fuente generadora. La propiedad que caracteriza a la fuente es la distribución de probabilidad asociada a la emisión de los símbolos. Por tanto, es necesario tener en consideración la distribución de probabilidad a la hora de crear la codificación.

El cálculo de la entropía nos ofrece información sobre el límite inferior en la codificación que podemos obtener. Mediante la regla de Shannon-Fano podemos establecer una codificación "aceptable" pero no óptima. Para obtener la codificación óptima es necesario aplicar las dos reglas de Huffman.

La salida del código, una vez implementados los métodos correctamente, será la siguiente:

```
Ejercicio 5. Probabilidades: [0.4, 0.2, 0.2, 0.1, 0.1]

Shannon Fano Codigos
--------------------

Code obtained with 0 1 2 2 0  parameters:
 00  010  011  1000  1001

 Huffman Codigos
--------------------

Optimal Code obtained with [1, 1, 1, 2, 0, 0, 0, 0, 0, 0] parameters:
[0, 10, 110, 1110, 1111]

****** Calculo Valores ***********

Entropia: 2.1219280948873624
Longitud Media: 2.8
Longitud Optima: 2.2
Verificar h<lo<lm<h+1: true
```

3. Código

Clase Fuente

```java
/**
 *
 */
package Practica2;

import java.util.Arrays;

import Practica1.BinaryTreeCod;
import Practica1.TreeException;


```

```java

/**
 * Practica 2. TIC
 *
 * @author Jose A Montenegro
 * @version 1.0 8/10/2013
 *
 */
public class Fuente {

        private static final int AddLevel = 3; //ToDo: Obtener valor teorico.
        double []probabilidades;
        int lengProbabilidades=0;
        BinaryTreeCod code;
    //Arbol para calcular codigo con los parametros de shannon-fano

        //Valores para shannon-Fanno
        boolean codeSetting=false;
        boolean parametersValid=false;
        int[] parametros;
        int parametroslong;

        HuffmanCode huffmanCodification=null;

        /**********
         * Constructor de la clase.
         * Inicialmente ordeno las probabilidades para estar
         * en orden decreciente.
         * Ojo puede ser un problema al codificar. Para este
         *  ejercicio no supone ningun problema
         *
         * @param distribution Array de probabilidades
         */
        public Fuente( double [] distribution) {

                //Ordenan de forma decreciente las probabilidades.
                // ToDo: Sustituir por un algoritmo unico.

                probabilidades = reverseOrder(distribution);

                lengProbabilidades = probabilidades.length;

                parametroslong= (int)logBase2(lengProbabilidades)+AddLevel;
                parametros = new int[parametroslong];
        }

```

```
58
59
60
61          /**
62           * Calcula la entropia
63           * @return        Entropia de la distribucion de probabilidad.
64           */
65          public double entropia() {
66                  double h = 0;
67
68          // Tu codigo aqui
69
70                  return h;
71          }
72
73          /**
74           * Calcula la longitud media de un codigo.
75           *
76           * Nota: El parametro n1 esta en la posicion cero del array.
77           * @param   Codigo
78           * @return        Longitud media.
79           */
80          public double longitudMedia() {
81                  double l = 0.0;
82                  int j=0;
83                  int temp=parametros[0];
84
85                  if (parametersValid){
86
87                          // Tu codigo aqui
88
89                  }
90
91                  return l;
92          }
93
94          /**
95           * Devuelve los parametros ni para un codigo segun la regla de Shannon-Fano.
96           *
97           * @return        Parametros de un codigo.
98           */
99          public int [] reglaDeShannonFano() {
100
101                 int y;
102
103         // Tu codigo aqui
```

```
104
105             parametersValid=true;
106
107             return parametros;
108         }
109
110         /************
111          * Calcula los codigos mediante arbol binario de practica 1 y
112          * los parametros de Shannon-Fano
113          *
114          * @return
115          * @throws TreeException
116          */
117         public boolean setCodeShannonFano() throws TreeException{
118
119             int [] parameters=reglaDeShannonFano();
120             int leng=parameters.length;
121
122             code = new BinaryTreeCod(leng);
123             codeSetting=code.buildCodeParameters(parameters);
124
125             return codeSetting;
126         }
127
128         /***************
129          * Calcula el codigo optimo mediante la clase Huffman
130          *
131          * @return
132          */
133
134         public void generarCodigoOptimo(){
135             huffmanCodification = new HuffmanCode(probabilidades);
136         }
137
138         /*********
139          * Obtengo la longitud optima del codigo huffman.
140          * Si no esta calculado codigo optimo devuelve cero.
141          *
142          * @return longitudOptima
143          */
144         public double longitudOptima(){
145
146             if(huffmanCodification==null) return 0.0;
147
148             return huffmanCodification.longitudOptima();
149         }
```

```
150
151         /********
152          * Imprime el codigo optimo calculado
153          */
154
155         public void printCodeOptima(){
156
157          if(huffmanCodification!=null) {
158                 int huffmanParameters[];
159                 huffmanParameters = huffmanCodification.getParametrosOptima();
160                 System.out.println("\nOptimal Code obtained with "+
161                 Arrays.toString(huffmanParameters)+" parameters:");
162                 System.out.println(huffmanCodification.getCodeOptima().toString());
163                 }
164         }
165
166         /********
167          * Verifica desigualdades de la pagina 54
168          *
169          * @param entropia
170          * @param lm
171          * @param lo
172          * @return verdadero si cumple las condiciones
173          */
174
175
176         public boolean Verificar (double entropia,double lm, double lo){
177                 if (entropia<lo & lo< lm & lm <(entropia+1)) return true;
178                 else return false;
179         }
180
181         /*******
182          * Calculo logartimo base 2 mediante cambio de base
183          * @param num
184          * @return
185          */
186         private  double logBase2(double num) {
187                 return logCambioBase(num,2);
188         }
189
190         private  double logCambioBase(double num, int b) {
191                 return Math.log(num)/Math.log(b);
192         }
193
194         /**********
195          * Ordena de forma descendente las probabilidades
```

```
196          * ToDo: Mejorar algoritmo
197          * @param distribution
198          * @return
199          */
200         private double [] reverseOrder(double distribution []){
201 
202                 Arrays.sort(distribution);
203 
204                 double temp;
205 
206                 for(int i = 0; i < distribution.length / 2; i++)                {
207                     temp = distribution[i];
208                     distribution[i] = distribution[distribution.length - 1 - i];
209                     distribution[distribution.length - 1 - i] = temp;
210                 }
211                 return distribution;
212         }
213 
214 
215 
216         public static void main(String[] args) {
217 
218                 double [] probabilidadesEjemplo5={0.4,0.2,0.2,0.1,0.1};
219 
220 
221                 Fuente fuente = new Fuente (probabilidadesEjemplo5);
222                 int [] parameters=fuente.reglaDeShannonFano();
223 
224                 // Creo codigo con los parametros que calculamos con ShannonFano mediante a
225 
226                 BinaryTreeCod code = new BinaryTreeCod(parameters.length);
227                 boolean result=false;
228                 try {
229                         result = code.buildCodeParameters(parameters);
230 
231                 } catch (TreeException e) {
232                         // TODO Auto-generated catch block
233                         e.printStackTrace();
234                 }
235 
236 
237                 //Calculo codigo optimo con huffman
238 
239                 fuente.generarCodigoOptimo();
240 
241 
```

```
                double entropia,lm,lo;
                entropia        = fuente.entropia();
                lm                          = fuente.longitudMedia();
                lo                         = fuente.longitudOptima();


    System.out.println("Ejercicio 5. Probabilidades: "+Arrays.toString
     (probabilidadesEjemplo5));

    System.out.println("\nShannon Fano Codigos\n--------------------");
        if (result) code.printCodes(parameters);

        System.out.println("\n\n Huffman Codigos\n--------------------");
        fuente.printCodeOptima();

                System.out.println("\n****** Calculo Valores **********\n");
                System.out.println("Entropia: "+entropia);
                System.out.println("Longitud Media: "+lm);
            System.out.println("Longitud Optima: "+lo );

    System.out.println("Verificar h<lo<lm<h+1: "+
                fuente.Verificar(entropia, lm, lo));


        }

}
```

Clase HuffmanCode Java.

```
package Practica2;


import java.util.*;


/**
 * Practica 2. TIC
 *
 * @author Jose A Montenegro
 * @version 1.0 8/10/2013
 *
 * Algoritmo basado en la version http://rosettacode.org/wiki/Huffman_coding#Java
 *
```

```
 */

public class HuffmanCode {
        double[] probabilidades;
        HuffmanTree tree;
         List <Values> CodeList;


        public         HuffmanCode (double[] probabilidadesP){
                probabilidades=probabilidadesP;

                 tree= regla1(probabilidades);
                 CodeList=new ArrayList<Values>();
             CodeList=regla2(tree, new StringBuffer(),CodeList);

        }

/*****************
 *
 * @param probabilidades
 * @return
 */
 public  HuffmanTree regla1(double[] probabilidades) {

           PriorityQueue<HuffmanTree> trees = new PriorityQueue<HuffmanTree>();

    for (int i = 0; i < probabilidades.length; i++)
        trees.add(new HuffmanLeaf(probabilidades[i]));

        assert trees.size() > 0;

        while (trees.size() > 1) {

                        //Optiene los dos elementos menores
                        //Estan ordenados
                        HuffmanTree a = trees.poll();
                        HuffmanTree b = trees.poll();
                        //Crea un nodo nuevo con los dos anteriores
                        trees.offer(new HuffmanNode(a, b));
                    }
                        return trees.poll();
     }
/*********
 *
 * @param tree
 * @param prefix
```

```
61  * @param code
62  * @return
63  */
64
65     public   List <Values>  regla2
66         (HuffmanTree tree, StringBuffer prefix,List <Values> code) {
67
68                         assert tree != null;
69                         Values value;
70
71                 if (tree instanceof HuffmanLeaf) {
72                          HuffmanLeaf leaf = (HuffmanLeaf)tree;
73                              value=new Values(prefix.toString(),leaf.frequency);
74                              code.add(value);
75
76                 } else if (tree instanceof HuffmanNode) {
77                               HuffmanNode node = (HuffmanNode)tree;
78
79                               prefix.append('0');
80                               code= regla2(node.left, prefix,code);
81                               prefix.deleteCharAt(prefix.length()-1);
82
83                               prefix.append('1');
84                               code= regla2(node.right, prefix,code);
85                           prefix.deleteCharAt(prefix.length()-1);
86                   }
87                 return code;
88         }
89 /**********
90  * Obtiene los codigos
91  * @return
92  */
93         public List <Values> getCode(){
94                 return CodeList;
95         }
96
97 /*********
98  * Calcula la longitud optima
99  * @return
100 */
101        public double longitudOptima() {
102           double l=0.0;
103
104                Iterator<Values> itr = CodeList.iterator();
105
106                while(itr.hasNext()) {
```

```
107                         Values element = (Values) itr.next();
108                         l+=element.frecuencia*element.size;
109                 }
110                 return l;
111         }
112 /************
113  *         Obtiene los parametros optimos
114  * @return
115  */
116         public int [] getParametrosOptima() {
117                         int parameters [] =new int [10];
118                         Iterator<Values> itr = CodeList.iterator();
119 
120                         while(itr.hasNext()) {
121                                 Values element = (Values) itr.next();
122                                 parameters[element.size-1]++;
123                          }
124                          return parameters;
125                 }
126 /**********
127  * Obtiene los codigos de huffman
128  * @return
129  */
130         public List<String>  getCodeOptima() {
131                         List<String> code = new ArrayList<String>();
132                         Iterator<Values> itr = CodeList.iterator();
133 
134                         while(itr.hasNext()) {
135                                 Values element = (Values) itr.next();
136                                 code.add(element.code);
137                          }
138                          return code;
139                 }
140 
141 /********
142  * Clases Auxiliares para construir el arbol Huffman
143  *
144  */
145 
146 class HuffmanTree implements Comparable<HuffmanTree> {
147     public double frequency; // the frequency of this tree
148     public HuffmanTree(double freq) { frequency = freq; }
149 
150     // compares on the frequency
151     public int compareTo(HuffmanTree tree) {
152             return (int) (frequency*100 - tree.frequency*100);
```

```
153     }
154
155     public String toString(){
156             return frequency+" ";
157     }
158 }
159
160 class HuffmanLeaf extends HuffmanTree {
161     public HuffmanLeaf(double freq) {
162             super(freq);
163       }
164 }
165
166 class HuffmanNode extends HuffmanTree {
167     public final HuffmanTree left, right; // subtrees
168
169     public HuffmanNode(HuffmanTree l, HuffmanTree r) {
170         super(l.frequency + r.frequency);
171         left = l;
172         right = r;
173 }
174
175 }
176
177 /*******
178  * Clase que devuelve los codigos con sus probabilidades
179  * y longitud, utilizadas para calcular los parametros.
180  *
181  * @author joseamontenegromontes
182  *
183  */
184 class Values {
185         double frecuencia;
186         String code;
187         int size;
188
189          public Values(String codeA, double frecuenciaA) {
190                         frecuencia=frecuenciaA;
191                         code=codeA;
192                         size=codeA.length();
193             }
194
195          public String toString (){
196                  return frecuencia + " "+code;
197          }
198 }
```

```
199
200 }
```