Universidad Nacional de Mar del Plata.
Facultad de Ingeniería.
Departamento de Ingeniería informática.
Asignatura: Teoría de la Información.

Trabajo Practico Integrador

Informe

Grupo n°3

Integrantes:

- Cacace, Camila (16403)
- Gentili, David (12940)
- Luna, Lautaro (16175)

Repositorio: https://github.com/lunalauti/teoria-de-la-informacion.git

Fecha de entrega: 20/10/2022

Índice

Resumen

En este primer trabajo integrador se analizará una fuente de información brindada por la cátedra, determinando el tipo de fuente con el que se está tratando, y en base a esta fuente se formarán tres alfabetos código, se clasificarán y se hará un análisis para determinar si cumplen las condiciones de ser compactos. Finalmente se cargará en un archivo la codificación de los símbolos de estos según Huffman.

Como objeto de estudio tenemos a la fuente ya mencionada y a su vez, este mismo archivo que se usó para el análisis de la fuente se utilizará reinterpretado como un conjunto de códigos con el fin de estudiar las propiedades de los códigos y la codificación de los mensajes.

Lo que se busca con este trabajo es analizar las fuentes para poder entender mejor las propiedades de los códigos y las características de las fuentes de información. Además se quiere indagar en mayor profundidad la codificación de fuentes de información.

Introducción

Antes de comenzar a desarrollar los algoritmos implementados para el trabajo integrador daremos una breve introducción de manera que se entienda con mayor facilidad el marco teórico con el que elaboramos el código.

Teoría de la información

La información que se transmite depende de qué tan desconocido es el hecho presentado, tiene que ver con el grado de incertidumbre del mensaje transmitido. El cálculo de dicha cantidad de información está ligado con la probabilidad que tiene de ser enviada, mientras mayor sea ésta, menor será la información que aporte.

Relacionado directamente con este concepto, se encuentra la entropía . Esta se refiere a la cantidad media de información por símbolo transmitido, que también puede ser interpretada como la incertidumbre del observador antes de conocer la salida de la fuente, que en esencia refieren a lo mismo.

Uno de los principales elementos del modelo de comunicación que modela esta propuesta teórica son las fuentes de información, es decir *cualquier origen de información susceptible de ser representado mediante una señal analógica y/o digital.* Estas pueden clasificarse bajo distintos criterios que describen el comportamiento de la misma y como se la deberá manipular. Características tales como: **rango de valores que puede generar**, refiriéndose a la capacidad de la fuente de generar valores continuos o finitos, o **relación entre sus símbolos**, es decir la dependencia estadística de los símbolos que la componen.

El análisis de la relación entre los símbolos que emite una fuente de información nos permite clasificar dos tipos específicos, aquellas donde no existe dependencia alguna entre la aparición de los símbolos, denominados **fuente de memoria nula**. En contraposición, se encuentran las **fuentes de Markov** a las cuales Norman Abramson define como*:* "En una fuente de Markov de orden m, la probabilidad de un símbolo cualquiera viene determinada por los m símbolos que lo preceden. En cualquier momento, por lo tanto, definiremos el estado de la fuente de Markov de orden m por los m símbolos precedentes." (Abramson, 1986, Pág. 36). A su vez dicho autor define la “Fuente Afín” las cuales, son fuentes de Markov de orden 1.

Al adentrarse en el estudio de las fuentes de Markov, más específicamente en las de orden 1, se pueden obtener dos clasificaciones, ergódicas y no ergódicas, que dependen del comportamiento que se puede observar al transcurrir un tiempo relativamente largo. Abramson define las fuentes Ergódicas como: *"*aquella que, observada durante un tiempo suficientemente largo, emite (con probabilidad 1) una secuencia típica de símbolos(...)".(Abramson, 1986, Pág. 37).

Esta característica de presentar una secuencia típica de símbolos pasado un tiempo largo de su transmisión, concluye en una estabilización de las probabilidades de aparición de estas secuencias que permite plantear lo que denominaremos “vector estacionario” que es presentado por Abramson de esta manera: "Existe una distribución de probabilidades única para un conjunto de estados de una fuente de Markov ergódica, y los estados en cualquier secuencia suficientemente larga, se presentarán (con probabilidad 1) de acuerdo con esa distribución. Esta distribución única recibe el nombre de distribución estacionaria del proceso ergódico de Markov;" (Abramson, 1986, Pág. 39). Su cálculo sería el siguiente:

$$V^* = M \cdot V^* \quad Luego$$
$$(M - I) \cdot V^* = 0$$

$$Además \sum Vi^* = 1$$

Propiedades de los códigos

Profundizando un poco más en el tema de las fuentes, se pueden analizar los tipos de códigos que estas transmiten. Definiendo un código como “la correspondencia de todas las secuencias posibles de símbolos de S a secuencias de símbolos de algún otro alfabeto X”.

Estos códigos se pueden clasificar en código bloque o no bloque. Un código bloque es aquel que asigna a cada uno de sus símbolos alfabeto, lo que sería S en la definición anterior, una secuencia fija de símbolos del alfabeto código X que permanece en el tiempo.

En el caso de que tratemos con un **código bloque**, el análisis se puede seguir extendiendo, determinándolo como **singular** o **no singular**, definiendo un código singular como aquel que presenta palabras código repetidas, es decir que a una palabra código le pueden corresponder distintos símbolos de la fuente. Por ende, un código no singular tiene todas sus palabras distintas.

Si al continuar el estudio obtenemos un código no singular, se puede analizar una características más, lo que denominamos **código unívocamente decodificable**. En este caso estamos refiriéndonos a lo que sería la interpretación del código y cuán ambigua puede ser. Si tratamos efectivamente con un código unívoco , quiere decir que dos secuencias cualesquiera de símbolos de la fuente de igual longitud, o también de distinta longitud, dan lugar a secuencias de símbolos códigos distintos. Esto quiere decir que, como se mencionó anteriormente, el código no da lugar a ambigüedades a la hora de su decodificación.

Finalmente, el último análisis que podemos realizar con respecto a los códigos es si se trata de un código instantáneo o no instantáneo. La definición que nos brinda Abramson acerca de esta característica es la siguiente: “Un código unívocamente decodificable se denomina instantáneo cuando es posible decodificar las palabras de una secuencia sin precisar el conocimiento de los símbolos que las suceden.” (Abramson, 1986, Pág. 66).

Ampliando un poco su descripción, podemos agregar que “la condición necesaria y suficiente para que un código sea instantáneo es que ninguna palabra del código coincida con el prefijo de otra.”

La determinación de esta última propiedad se determina a través de dos inecuaciones. Inecuación de Kraft y McMillan,

$$Inecuación\ de\ Kraft\colon \sum_{i=1}^{q} r^{-1} \leq 1$$

que determinan la condición necesaria y suficiente para la existencia de un código instantáneo.

Problemas de la codificación

La longitud media de un código se define de la misma forma tanto para las fuentes de memoria nulas como para las de Markov. Se trata de un cálculo sencillo que utiliza la probabilidad de los símbolos y su longitud:

$$Longitud\ media\colon\ L\ =\ \sum_{i=1}^{q} p_i . l_i$$

Un problema fundamental de la codificación de las fuentes de información es la búsqueda de códigos compactos. Con esto nos referimos a un código cuya longitud media es igual o menor que la longitud media de todos los códigos unívocos que pueden aplicarse a la misma fuente y el mismo alfabeto.

Así, para contribuir con la optimización de los códigos se puede aplicar el algoritmo de Huffman, o bien se puede utilizar el algoritmo de Shannon-Fano obteniendo códigos subóptimos.

Finalmente, definimos la rendimiento y redundancia de la siguiente forma:

$$Rendimiento:\ \eta\ =\ \frac{H_r(S)}{L}$$

$$Redundancia:\ 1\ -\ \eta\ =\ \frac{L - H_r(S)}{L}$$

Desarrollo

Análisis de fuentes de información (Parte 1)

Recolección de datos

Para poder determinar si la fuente discreta tiene o no memoria debimos analizar la relación entre sus símbolos, para ello elaboramos una **matriz con las probabilidades condicionales**, construida en base a la implementación de los siguientes algoritmos: la fuente lee todo el archivo con los símbolos y a medida que lo lee va incrementando en la matriz el elemento de la **fila del símbolo actual** y la **columna del símbolo previo**, quedando la posición (actual,previo). Además en cada lectura incrementa el contador de incidencias del símbolo correspondiente. Finalmente, dividimos cada columna por el total de apariciones de su símbolo.

```
void matrizTransicion(){
    char act, prev
    int i,j
    double matriz= getMatriz()
    Archivo= Abrir("Fuente.txt")
    prev= Leer(Archivo)
    Mientras(!FinArchivo(Archivo)){
        act=Leer(Archivo)
        matriz[act][prev]= matriz[act][prev]+1
    }
    Para j=0 Hasta TamañoDeMatriz{
        total=0
        Para i=0 Hasta TamañoDeMatriz
            total= total+ matriz[i][j]
        Para i=0 Hasta TamañoDeMatriz
            matriz[i][j]= matriz[i][j]/total
    }
}
```

Definición del tipo de fuente

Teniendo ésta **matriz de transición** o de **probabilidades condicionales** , la cual nos informa la *probabilidad de que ocurra el símbolo i siendo j el actual*, evaluamos si los símbolos son **estadísticamente independientes** o si **dependen de los símbolos anteriormente emitidos**. Para ello desarrollamos una función teniendo en cuenta que, para ser estadísticamente independientes los símbolos entre sí, la probabilidad del símbolo de cada fila debe ser igual en cada columna ya que de esta forma sería *igual de probable que ocurra dicho símbolo sin importar los* emitidos *previamente.* Esta función booleana lo que hace es justamente recorrer cada fila comparando las probabilidades de dicha fila en todas las columnas con cierto margen de tolerancia (en este caso una diferencia de 0.01), en el caso de no cumplirse en alguna fila, ya determina que nos encontramos frente a una **fuente de memoria no nula**, como ocurrió en nuestro caso.

A partir del resultado obtenido en este cálculo, debemos seguir analizando esta fuente considerando que se trata de una fuente de memoria no nula. Por ende, una de las características que se puede averiguar en fuentes de este tipo es si son **ergódicas o no ergódicas**. Para determinar esta condición, lo que hicimos fue basarnos en las condiciones necesarias para la existencia del **vector estacionario**, las cuales nos permiten afirmar que si existe tal vector entonces la fuente es ergódica.

Cálculo del vector estacionario

Lo que representa este vector es justamente el comportamiento a largo plazo que aparece por estar tratando con una fuente periódica, donde se comienza a repetir un patrón. Al no depender este estado de las condiciones iniciales, podemos obtenerlo a partir de las probabilidades condicionales. Para este cálculo planteamos un sistema de ecuaciones con la diferencia entre la matriz de transición y la matriz identidad, igualada a cero (A) y la sumatoria de las variables igualadas a 1 (B). Esto lo implementamos cargando una matriz con la ecuación A, y reemplazando arbitrariamente una fila por la ecuación B, extendiendo la matriz y finalmente aplicando el método de Gauss-Jordan. Así nos quedan en las variables los valores del vector estacionario.

```
void calculoVecEstacionario(){
    double matriz= getMatriz()
    double[TamañoDeMatriz] extension= {0,0,1}
    int i
    int j=2 //valor arbitrario
    double[TamañoDeMatriz] VEstacionario

    Para i=0 Hasta TamañoDeMatriz
        matriz[i][i]= matriz[i][i] - 1

    Para i=0 Hasta TamañoDeMatriz
        matriz[j][i]=1

    VEstacionario= GaussJordan(matriz,extension)
}
```

Cálculo de la entropía

Con el vector estacionario pudimos determinar la entropía de nuestra fuente. Es decir, *la cantidad media de información que aporta cada símbolo*. Para esto implementamos la fórmula vista en la teoría para este tipo de fuentes considerando la matriz de transición con las probabilidades condicionales y el vector estacionario obtenido en nuestro último cálculo. Desarrollamos una función que recorriendo la matriz va sumando, para cada columna, la información de cada uno de sus elementos (probabilidades) y al finalizar el recorrido se le multiplica a este sumador, el elemento del vector que corresponde con la columna recorrida, almacenando en la variable final de entropía.

```
String getEntropia(double VEstacionario[],int n){
    int i,j
    double matriz= getMatriz()
    double aux, H=0;
    Para i=0 Hasta n {
        aux=0
        Para j=0 Hasta n
            aux= aux + matriz[i][j]* logbase2(1/matriz[i][j])
        H=H+aux*VEstacionario[i]
    }
    return "Entropia de la fuente" + H
}
```

Codificación y propiedades de los códigos (Parte 2)

```
    int contador=1
    string palabra=""
    Archivo= Abrir("Fuente.txt")
    Mientras(!FinArchivo){
        simbolo= Leer(Archivo)
        palabra= palabra+simbolo
        Si (contador==longitud){
            contador=1
            cargarPalabra(palabra)
            palabra=""
        }Sino contador++
    }
```

Recolección de datos

Para poder realizar el análisis de la segunda parte del trabajo, tomamos nuevamente el archivo con los símbolos y, agregando tres nuevos atributos a nuestra clase Fuente, almacenamos en cada uno de ellos las cadenas de 3, 5 y 7 que se forman agrupando los símbolos según corresponda, obteniendo así las palabras código, y sus respectivas frecuencias.

Con esto determinado pudimos realizar diferentes análisis.

Análisis de tipos de códigos

Al analizar teóricamente nuestro código, podemos observar que se trata de un **código en bloque** ya que a cada uno de los símbolos del alfabeto fuente, que, a pesar de que nosotros no conocemos, sabemos que se le asigna una secuencia fija de símbolos del alfabeto código (que es el que nosotros tenemos). A partir de conocer esta característica de nuestros códigos, podemos continuar con su estudio. Otra propiedad que podemos observar es que tratamos con **códigos no singulares**, y esto se debe a que todas sus palabras son distintas. Al ser un código bloque no singular y como todas las palabras código tienen la misma longitud podemos decir que se trata de un **código unívocamente decodificable**, ya que con el mismo tamaño de las cadenas se hace imposible el conseguir formar una de las palabras a partir de la combinación de otras. Y como ninguna palabra del código coincide con el prefijo de otra podemos afirmar que se cumple la condición necesaria y suficiente para ser un código instantáneo.

Cálculo de cantidad de información y entropía

Inicialmente lo que hicimos fue transformar las frecuencias de cada palabra en probabilidades, dividiendo la frecuencia por la cantidad total de palabras, para poder realizar los cálculos siguientes.

Para informar la cantidad de información de las palabras de cada alfabeto código lo que hicimos fue aplicar logaritmo (en base 3 en nuestro caso) de la probabilidad a cada palabra del mismo, tal como se explicó previamente.

Basándonos en la definición ya dada de entropía, para calcular la cantidad media de información de cada conjunto de palabras código lo que hicimos fue recorrer cada alfabeto código y acumular la suma de la cantidad de información de cada palabra multiplicado por su probabilidad.

Análisis de compacticidad del código

Continuando con el análisis analítico, para determinar si se cumple la condición necesaria y suficiente para la existencia de un código instantáneo, es decir, *si es posible decodificar las palabras de una secuencia sin necesitar conocer los símbolos que la suceden*, aplicamos la inecuación de Kraft. Para implementar este cálculo, recorrimos nuevamente las palabras código de cada tamaño y en un acumulador sumamos la cantidad de símbolos que conforman las palabras código (3 en nuestro caso) elevado a la opuesta de la longitud de cada una de las palabras. Si el resultado es menor o igual a 1 se puede decir que se cumple la condición, caso contrario no se podrá encontrar un código instantáneo con esas características.

```
boolean cumpleKraft(String Alfabeto[], int cantSimbolos){
    int i, suma=0
    boolean cumple=false
    String palabra=""
    Para cada palabra de Alfabeto
        suma= suma + cantSimbolos^-longitud(palabra)
    Si (suma<1)
        cumple=true
    return cumple
}
```

La longitud media de un código se define como la sumatoria de la probabilidad de cada palabra por su longitud. El pseudocódigo del código aplicado en nuestro trabajo es el siguiente:

```
double getLongitudMedia(){
    double suma=0
    Para cada palabra de Alfabeto
        suma= suma + longitud(palabra) * probabilidad(palabra)
    return suma
}
```

Determinación de rendimiento y redundancia

Para calcular tanto la redundancia como el rendimiento de un código, se necesita conocer su longitud media, nombrada anteriormente, y su entropía. Si queremos conocer el rendimiento o eficiencia, es decir que tanta información aporta nuestro código debemos realizar la división entre la entropía y la longitud media, como se puede observar en el pseudocódigo a continuación.

```
void getRendimiento(int longitudMedia, double entropia){
    float n;
    n = entropia / longitudMedia;
    Escribir "Rendimiento del codigo" n
}
```

(En el caso en que la longitud media sea igual a la entropía el rendimiento es máximo es decir que se aporta la mayor cantidad de información posible).

En el caso de la redundancia, esta significa lo contrario a lo visto anteriormente, mientras más redundante sea un código, menos información aporta. Por ende, su cálculo se puede plantear como 1 - el rendimiento, quedando entonces:

```
void getRedundancia(int longitudMedia, double entropia){
    float n;
    n = (longitudMedia - entropia) / longitudMedia;
    Escribir "Redundancia del codigo" n
}
```

Codificación y compresión de códigos

Para la obtención de un “código compacto” es necesario utilizar un algoritmo como pueden ser Shannon-Fano, o Huffman, para esto es necesario interpretar al conjunto de cadenas de orden n, tomadas previamente, como un “alfabeto fuente”, siendo cada una de estas un símbolo de dicho alfabeto. También por conveniencia en la implementación, se utiliza como alfabeto código el binario {0,1}.

Entre estos dos algoritmos, se optó por utilizar el método de Huffman. Para dicha implementación, se decidió ordenar todos los símbolos utilizando como criterio la probabilidad de aparición. Luego, se tomaron los dos más pequeños y se los retiraron de la estructura, enlazandolos a un nuevo “nodo” que se puede denominar “padre”, que posee

como valor de probabilidad, la suma de las probabilidades de los “hijos”. Este nuevo nodo se incluye en la estructura, adicionales a los hijos un prefijo a la palabra código, siendo ‘0’ a los hijos de la “rama izquierda”, y ‘1’ a los de la “rama derecha”, a su vez, estos nodos asignan el mismo sufijo a sus hijos, sin importar la rama. Este proceso continúa hasta que solo queda un nodo en la estructura, dicho nodo se puede interpretar como la “raíz” del árbol de Huffman, la cual tendrá como valor de probabilidad de aparición 1, y todos los nodos “hojas” de dicho árbol serán los símbolos del “alfabeto fuente”, cada uno con su respectiva palabra código.

Una vez se encuentran tabuladas las transformaciones entre el “alfabeto fuente” y el “alfabeto código”, se comienza el proceso de transformación del mensaje original, recorriendo el archivo de entrada, tomando las cadenas de la longitud correspondiente para cada caso, transformando cada cadena a su respectivo código y concatenado dicho código, con el fin de obtener el contenido original codificado.

Una vez que se posee dicho código, es relevante almacenarlo en un archivo para su posible transmisión, por ende, con el fin de poder decodificar el mensaje es imperioso almacenar la tabla que permita interpretar dicho mensaje, para poder almacenar la tabla y el mensaje a transmitir, por ende, es necesario definir una estructura que permita al receptor leer el archivo.

Para esto se definió: las cadenas se almacenaron en un byte por cada símbolo del alfabeto código original, seguido de esto, se almaceno la palabra código correspondiente, utilizando un byte por cada bit o símbolo del alfabeto código, como divisor entre estos dos elementos se utilizó el byte ‘:’ (ASCII), entre cada par símbolo y palabra código, se utilizó el byte ‘;’ como divisor. Así también, para dividir la tabla del mensaje se utilizó el byte que representa el salto de línea ‘\n’. Posterior a la tabla, se utilizaron 4 bytes para especificar la cantidad de bits que corresponden al código, esto debido a que los códigos se leen/escriben de a un byte, por ende, existe la posibilidad de que la longitud del código, no sea divisible por 8 bits, teniendo que completar el mensaje con ‘0’s, al contar con la longitud en bits del código, es posible descartar los bits sobrantes y evitar así, arrastrar “basura” en la interpretación del mensaje.

Por último, respetando la estructura previamente definida, para lectoescritura, almacena la tabla en el archivo, se calcula la longitud en bits del código, y se almacena en el documento, y por último se comienza a dividir el mensaje en 8 bits, almacenando el byte que corresponde y en caso de existir un “resto”, se los completa con ‘0’s a la derecha.

Conclusiones

A partir de la recolección de datos para la creación de nuestra matriz de probabilidades, pudimos realizar un gran número de cálculos que nos brindaron información acerca de la fuente tratada.

Al calcular el vector estacionario y obtener resultados coherentes pudimos determinar que efectivamente se trataba de una fuente ergódica, lo que implica que no hay estados o clases absorbentes, es decir que todos los estados del proceso son alcanzables desde otro estado.

Respecto de la segunda parte, analizando los resultados obtenidos en las inecuaciones de Kraft, podemos afirmar que todos los alfabetos cumplen con la condición suficientes para la existencia de un código instantáneo, y a su vez, como todos tienen códigos unívocos entonces es una condición necesaria que se cumpla la inecuación de

McMillan (en el código no está distinguido entre inecuación de Kraft e inecuación de McMillan ya que ambas son idénticas).

Notamos que a medida que crece el orden de la fuente aumenta el valor de su entropía, es decir, aumenta la cantidad media de información por símbolo, siendo de $H_3(F)=2.54$ para un orden 3, $H_5(F)=4.10$ para un orden 5 y $H_7(F)=5.40$ para un orden 7. Además, calculando la longitud media verificamos nuevamente que se trata de un código compacto ya que se cumple con $H_r(S) \leq L$, siendo 3, 5 y 7 el valor de L correspondientemente.

Al analizar el rendimiento y la redundancia del código, podemos identificar una leve tendencia a disminuir el rendimiento, por lo tanto aumentar la redundancia, a medida que se incrementa la longitud de las cadenas, tomando la redundancia como valor, 0.85, 0.82 y 0.77 para las cadenas de orden 3, 5 y 7 respectivamente.

Al codificar los alfabetos por Huffman, notamos que se encuentra una relación entre la longitud de las cadenas, es decir el orden de la fuente, y la “calidad de compresión” de las cadenas, perdiendo efectividad a medida que aumenta la longitud media. Esto se puede ver claramente en el siguiente análisis de las compresiones hechas en el trabajo: el archivo original pesa 10KB y el archivo comprimido de orden 3 es de 2KB, resultando una compresión muy efectiva, sin embargo, al comparar el archivo original y el comprimido de orden 7 vemos que este último pesa 120KB, pesando mucho más que el original, y por ende, no siendo para nada efectivo.

Para finalizar con el análisis de los resultados obtenidos en la segunda parte del informe, tomando como referencia la tendencia que posee la relación entre la longitud de las cadenas con respecto al rendimiento, y la tendencia que presenta la tasa de compresión, en relación a las longitudes de las cadenas, podemos concluir que se obtienen resultados más eficientes, utilizando símbolos de menor longitud.

Referencias

Abramson, N. (1986). *Teoría de la Información y Codificación* (J. A. d. Miguel Menoyo, Trans.). Paraninfo.