Trabajo Práctico Nº1

Nombre, apellido y legajo de los integrantes: Mateo Etchepare (15093), Gregorio Firmani (15051), Franco Sardi([DNI]40831214).
Emails de contacto correspondientes: mateoetchepare@gmail.com , gregoriofirmani@gmail.com, fraansardi@gmail.com .
Número de grupo: 11
Universidad: Universidad Nacional de Mar del Plata.
URL GitHub: https://github.com/Greg1704/Teoria-de-la-Informacion/

Índice

Resumen

La Teoría de la Información es una propuesta teórica originada con un artículo de Claude E. Shannon, y trata de investigar y medir la información, su almacenamiento y comunicación, de una forma matemática y rigurosa. En el núcleo de ésta teoría están los emisores y receptores de información.

En este trabajo, se analizará un archivo de texto, tomándolo como fuente de información y aportando datos como su entropía, qué tipo de fuente es y características propias de ser nula o no nula, es decir, analizando la independencia estadística de ocurrencia entre símbolos se establecerá una o la otra. Luego, para la segunda parte se considerará la cadena de caracteres del archivo como palabras de un código de distintas longitudes, y se analizarán dichos códigos desde el punto de vista de la Teoría de la Información, aspectos como el tipo de código, compacidad, compresión mediante Algoritmo de Huffman, etcétera.

Introducción

Para plantear la resolución del problema en cuestión, utilizamos el lenguaje de programación **Java**, ya que tiene la potencia necesaria para resolver cálculos matemáticos complejos (como se detallará más adelante). Se irá recolectando cada símbolo del archivo y se procesará de forma de obtener los números necesarios para analizar y clasificar la fuente de información.

Usando un archivo de texto provisto por la cátedra, se tomó como fuente de información una cadena de 10.000 caracteres, que consta de 3 símbolos diferentes. En la primera parte nos planteamos ver si ésta fuente de información esa de memoria nula o no nula, que implica ver si la aparición de un símbolo influye en las chances de que luego aparezca o no un símbolo en particular, y para esto se necesita el cálculo de las probabilidades condicionales. Una vez establecida que la fuente es de memoria nula, pasamos a generar la extensión de orden 20, que implica generar una fuente de memoria nula donde cada símbolo de ésta se corresponde con una secuencia determinada de 20 símbolos de la fuente inicial.

Más adelante, en la segunda parte, se tomó la cadena de caracteres representando palabras de un código de longitud fija, de 3, 5 o 7 caracteres de largo. A partir de ésto, se calculó la cantidad de información que aporta cada palabra (que depende de su probabilidad de aparición) y la entropía de la fuente a partir de la probabilidad de ocurrencia de cada palabra y su cantidad de información. Luego establecimos qué tipo de códigos son en base a sus características (si son de bloque, singulares o no singulares, etcétera). Después, calculamos la Inecuación de Kraft, MacMillan y la longitud media de los códigos, y en base a los resultados establecimos características sobre dichos códigos. Igualmente con el cálculo del rendimiento y redundancia de cada código, que nos aportan información sobre qué tan útiles son para transmitir información. Por último, reconstruimos el archivo codificándolo a un alfabeto binario, así comprimiéndolo y sacando conclusiones en base a la diferencia en el tamaño.

Desarrollo

Primera Parte

inciso A: Probabilidades condicionales

Para comenzar con el problema, se escribió un código que permita **leer** el archivo y **procesarlo** de manera que se acumule la ocurrencia absoluta de cada símbolo en un **vector** y en una **matriz** la ocurrencia de cada símbolo S_i luego de la ocurrencia de cada símbolo S_j, siendo **i** las filas y **j** las columnas.. Luego, con esto, se calculó la **matriz de probabilidades condicionales**, la cuál contiene la probabilidad de que cada símbolo S_i ocurra luego de cada símbolo S_j con la fórmula siguiente:

FUNCIÓN EN PSEUDOCÓDIGO PARA OBTENER **MATRIZ DE PASAJE**
PARA i = 0 HASTA i = N CON PASO 1 HACER (N: TAMAÑO DE MATRIZ): PARA j = 0 HASTA j = N CON PASO 1 HACER (N: TAMAÑO DE MATRIZ): MATRIZ DE PASAJE [I][J] = MATRIZ DE OCURRENCIAS[I][J] / OCURRENCIA DEL SÍMBOLO J

Siendo N, en este caso, N = 3.

Las ocurrencias absolutas de cada suceso fueron:

A = 0,266
B = 0,387
C = 0,346

Para las probabilidades condicionales, se obtuvo la matriz. La matriz resultante, entonces, fue la siguiente:

	A	B	C
A	0,271	0,262	0,266
B	0,380	0,389	0,391
C	0,348	0,348	0,342

Con los resultados obtenidos, analizamos cada fila y sus respectivas probabilidades, de modo que si en alguna de ellas se encuentra que la diferencia entre el valor mayor y el menor es muy chica

entonces consideramos que la fuente es de **memoria nula,** ya que no hay cambio significativo en las probabilidades de que salga ese símbolo dados los distintos símbolos anteriores posibles. De otra forma, la fuente sería considerada una **fuente de Markov**. En éste caso, tomamos que si la diferencia entre la mayor probabilidad y la menor probabilidad, es mayor a 0.03 entonces es lo suficientemente significativa para ser considerada una fuente de Markov, caso contrario es una fuente de memoria nula. La tabla correspondiente a las diferencias entre cada fila está en **(1)** en la sección del Apéndice.

inciso B: Fuente de memoria nula y extensión orden 20

En el segmento de análisis para una fuente de **memoria nula**, se diseñó un algoritmo que genera la extensión de orden 20, eso es, todos los códigos posibles que sean de longitud 20. Entonces, el total de casos que se pueden obtener es 3^{20} (fórmula de **variación con repetición)**. Los resultados fueron los siguientes:

Usando la fórmula de la entropía de una fuente:

$$H(S) = \sum_{S} P(S_i) log_r(\frac{1}{P(S_i)})$$

Entropía inicial de la fuente = 0,989
Entropía orden 20 = 19,787

Como se puede observar, la entropía de orden 20 es 20 veces la entropía inicial, lo cuál satisface la fórmula vista en la teoría. Esto indica que la cantidad media de información suministrada por símbolo depende directamente del orden de la fuente.

$$H(S^n) = n * H(S)$$

inciso B: Pseudocódigo extensión orden 20

FUNCIÓN EN PSEUDOCÓDIGO PARA OBTENER LA ENTROPÍA DE ORDEN 20:
Definición variables: -input: Variable local del tipo String que cuenta con todos los caracteres posibles, en este caso “ABC”. -orden: Variable local del tipo Integer que lleva la cuenta del largo restante para que se pueda calcular la entropía de la cadena. -auxentrop: Variable local del tipo Double que contiene la multiplicación de las probabilidades de aparición de cada símbolo, la cual luego se usa para calcular la

entropía de la fuente. -VProb: Variable local del tipo ArrayList que contiene las probabilidades de aparición de los símbolos que se encuentran en input. -orden20entrop: Variable global del tipo Double que contiene la entropía total de orden 20.

Funcion void calculoEntropiaOrden20(input,orden,auxentrop,VProb[])
 	
SI orden = 0 ENTONCES
  Cálculo de la entropía en la variable auxentrop
  Sumo auxentrop en la variable global orden20entrop
SINO
  PARA i HASTA input.length CON PASO 1 HACER
  auxentrop = auxentrop * (probabilidad de alguno de los char de input)
  calculoEntropiaOrden20(input,orden-1,auxentrop,VProb[])
  auxentrop = auxentrop / (probabilidad de alguno de los char de input)

Segunda Parte

inciso A: Cantidad de información y entropía

Para calcular la cantidad de información de cada palabra del código, obtuvimos primero su cantidad de ocurrencias y con eso calculamos la probabilidad de ocurrencia del símbolo, usando la fórmula simple de
$Cantidad\ de\ ocurrencias$ / (10000 / $tamaño\ de\ cada\ símbolo$ (3, 5 *o* 7)).

El denominador representa la cantidad de símbolos que entran en el archivo. Dada la probabilidad de ocurrencia de un suceso, calculamos la información que transmite con la fórmula:

$$I(E) = log_r(\frac{1}{P(S_i)})$$

En este caso usando logaritmo de base 3 ya que el alfabeto código tiene 3 caracteres.

Para calcular la entropía de cada fuente, se hace la sumatoria de la probabilidad de cada símbolo por su cantidad de información, usando la fórmula ya mencionada anteriormente:

$$H(S) = \sum_S P(S_i) * log_r(\frac{1}{P(S_i)})$$

Se obtuvo:

Entropía de fuente de símbolos de longitud 3 = 2,965
Entropía de fuente de símbolos de longitud 5 = 4,900
Entropía de fuente de símbolos de longitud 7 = 6,186

inciso B: Tipo de código

Para responder al inciso b, determinamos que el tipo de código es instantáneo, ya que la consigna establece una longitud fija de las palabras, por lo que esto ya fija su condición de **código bloque**. Luego, nosotros almacenamos las ocurrencias de cada palabra, siendo la cantidad de palabras distintas posibles: $\#símbolosDelAlfabetoCódigo^{longitudPedida}$ $(variación\ con\ repetición)$. Por lo que esto afirma su condición de **código no singular**. Al cumplir estas dos condiciones simultáneamente, es un **código unívocamente decodificable**. Por último, quedaba chequear si se trataba de un código instantáneo. Al ser todos las palabras código de **misma longitud**, **todas diferentes** unas de las otras, es **imposible** que una sea el prefijo de la otra.

inciso C: Inecuación de Kraft, MacMillan, y longitud media del código

En el inciso c, la inecuación de Kraft nos dió, para los diferentes casos, los resultados indicados a continuación:

Primer caso (longitud = 3) = 0,999…
Segundo caso (longitud = 5) = 0,999…
Tercer caso (longitud = 7) = 0,456

Por lo que para los tres tipos de fuente planteados por la consigna, la inecuación de Kraft se cumple, ya que el resultado es menor o igual a 1. Debido a que la fórmula de MacMillan es la misma que la planteada por Kraft, los resultados son los mismos.

La inecuación de Kraft es sólo una medida cuantitativa basada en las longitudes de los códigos, pero **no** asegura que el código armado sea instantáneo. De todos modos, por lo explicado en el inciso anterior, el código es instantáneo.

La longitud media del código obtenido es la siguiente:

Primer caso (longitud = 3) = 3,0
Segundo caso (longitud = 5) = 5,0
Tercer caso (longitud = 7) = 6,999…

Los resultados obtenidos son totalmente lógicos, ya que las longitudes pedidas por la consigna son fijas: **NO** hay palabras código que sean de diferente longitud.

inciso C: Compacidad del código

Para analizar la compacidad del código, primero se debe analizar si es **unívoco**. Como se mencionó anteriormente, efectivamente lo es. Además, se debe verificar que la longitud media del código sea mínima, en este caso en particular, como las longitudes de los códigos son fijas, para que el código sea considerado compacto se debe dar que todas las combinaciones posibles, de longitud 3 sean equiprobables entre ellas, y lo mismo con las fuentes de longitud 5 y de longitud 7. Para corroborar si esto ocurre, se utilizó la siguiente igualdad que establece que la longitud de cada palabra código equivale al logaritmo en base "r" de la inversa de la probabilidad de emisión del símbolo fuente correspondiente:

$$l_i = log_r(\frac{1}{P_i})$$

Para el de longitud 3, los 27 casos posibles (3^3, $variación\ con\ repetición$) dan como resultado probabilidades diferentes entre ellas. Por lo cuál este código **no** puede ser considerado **compacto**.

Para el de longitud 5, para los 243 casos posibles (3^5, $variación\ con\ repetición$) ocurre lo mismo que con los códigos de longitud 3, dan probabilidades diferentes entre ellos, por lo que **tampoco** puede ser considerado **compacto**.

Para el de longitud 7, hay una gran cantidad de casos diferentes posibles:
(3^7, $variación\ con\ repetición$). Esto hace que no ocurran uno o varios de los casos posibles, ya que hay 2187 casos posibles y en un archivo de 10000 caracteres entran 1428 códigos de longitud 7. Por lo que todo esto hace **imposible** que el código sea compacto, ya que hay ciertos códigos que no aparecen, por lo que éstos tienen probabilidad 0, y los que aparecen, tienen una probabilidad >0, es decir, **no** son equiprobables.

inciso D: Rendimiento y redundancia del código

Para explicar el concepto de rendimiento, se debe tener en cuenta la entropía de la fuente. Cuanto más información aporte una fuente, su rendimiento será mayor. Los resultados a analizar fueron:

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

Rendimiento de fuente de símbolos de longitud 3 = 0,988
Rendimiento de fuente de símbolos de longitud 5 = 0,980
Rendimiento de fuente de símbolos de longitud 7 = 0,883

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

Redundancia de fuente de símbolos de longitud 3 = 0,011

Redundancia de fuente de símbolos de longitud 5 = 0,019
Redundancia de fuente de símbolos de longitud 7 = 0,116

Como se puede observar, el rendimiento de la fuente de símbolos de longitud 3 y de longitud 5, el rendimiento es 0,98, muy cercano a 1, mientras que en la fuente de símbolos de longitud 7, se reduce a 0,88. Esto se debe a que, en la secuencia de 10.000 caracteres brindada por el archivo, no caben el total de símbolos diferentes que se pueden generar con una longitud de 7 caracteres, por lo que hay símbolos que **NO** aparecen, mientras que los que sí se encuentran en el archivo tienen una o múltiples apariciones. Para contrarrestar que no quepan todos los símbolos posibles, lo deseado sería que **TODOS** (no la mayoría) de los símbolos aparezcan por lo menos **UNA** vez, así se obtiene mayor cantidad de símbolos diferentes, y que **NO** se repita ningún símbolo, y si lo hacen, lo menos posible.

Todo esto hace que la entropía no sea igual a la longitud media del código, pero de todas formas, es **casi** igual.

En cambio, la redundancia, es lo opuesto al rendimiento. Cuánto menos información aporte una fuente, su redundancia será mayor. Por lo que la redundancia de la fuente de símbolos de 7 caracteres, es mayor a la de las fuentes de símbolos de 3 y de 5 caracteres. Esto es lógico, ya que rendimiento y redundancia tienen que sumar 1.

inciso E: Algoritmo de Huffman

En el inciso e), decidimos aplicar el Algoritmo de Huffman. Éste algoritmo está diseñado para codificar los símbolos de una fuente con alfabeto r-ario a un alfabeto binario. El propósito principal de la codificación a alfabeto binario es comprimir el archivo original.

El archivo brindado por la cátedra tiene un tamaño de ≈ 10 kB. En tanto que los archivos obtenidos por el algoritmo de Huffman dieron como resultado un tamaño de:
Fuente de 3 caracteres: ≈ 3 kB
Fuente de 5 caracteres: ≈ 2 kB
Fuente de 7 caracteres: ≈ 1 kB

Por lo que para el primer caso, se obtuvo una compresión de un 70%, para el segundo caso, una compresión de un 80%, y para el último caso, un 90%.

Estos resultados son los encontrados si no ingresamos en el mismo binario la tabla con las traducciones de binario a alfabeto código, si agregamos las tablas en los archivos binarios, los pesos no cambian de manera notable. Para lograr ingresar la tabla sin comprometer la compresión del algoritmo transformamos las palabras código del alfabeto r-ario en un Integer que equivale a la suma de los caracteres de la palabra código en valor ASCII.

A continuación, adjuntamos un pseudocódigo del Algoritmo de Huffman utilizado en el código fuente:

PSEUDOCÓDIGO DE FUNCIÓN PRINCIPAL MÉTODO HUFFMAN

```
Función void MetodoHuffman(map){ //Funcion recursiva
   //inicialización de variables locales
   if(map tiene más de 2 elementos){
      Búsqueda de los 2 elementos de menor probabilidad
      Remuevo ambos elementos del map
      Agregó un nuevo elemento con la probabilidad de los dos eliminados sumadas
      MetodoHuffman(map); //Llamado recursivo
      Se recuperan los 2 elementos  anteriores
      Se reemplaza sus identificadores por el identificador padre más 0 o 1.
      Se remueve el elemento padre
      Se agrega nuevamente los 2 elementos que se encontraban anteriormente
   }else{
      Reemplazo el identificador del primer elemento restante por 0.
      Reemplazo el identificador del segundo elemento restante por 1.
   }
}
```

Apéndice

(1):

Símbolo	Probabilidad condicional más alta	Probabilidad condicional más baja	Diferencia
A	0,271	0,262	0,009
B	0,391	0,380	0,011
C	0,348	0,342	0,006

Conclusión

Para finalizar, podemos mencionar los aspectos más importantes de los códigos analizados:

- Se determinó que la primera fuente era una fuente de memoria nula. Es decir, emite símbolos de manera aleatoria, por lo que cada símbolo tiene la misma probabilidad de ocurrencia sin importar el símbolo anterior. Luego, se calculó la entropía de orden 20 y se pudo verificar que el resultado obtenido es la entropía de la fuente multiplicada por 20.

- Al ser códigos instantáneos, implica que sean unívocos, por lo que se analizó la compacidad de los códigos y se obtuvo que tanto los para los de longitud 3, 5 y 7 **no** eran compactos.

- Se analizaron propiedades importantes como el rendimiento y la redundancia de los códigos. Estos son indicadores simples pero muy importantes, ya que lo deseado es que el rendimiento sea el máximo posible (mayor aporte de información), y en los tres casos se pudo observar que el rendimiento es mucho mayor a la redundancia, llegando a valores cercanos a **1** (es decir, el 100%).

- Al separar los códigos en longitudes diferentes, se constató que eran códigos instantáneos, por lo que se pudo aplicar el Algoritmo de Huffman para traducirlo a lenguaje binario y así comprimir el tamaño de cada archivo respecto del archivo original.

- Todo lo anterior mencionado, siempre fue justificado con cada propiedad cuantificada, como por ejemplo, la entropía, la longitud media del código, etcétera.