Trabajo Práctico N°1

Autores: Alejandro Olave, Mikel ; Giacri, Tobías ; Nievas, Nahuel Isaias

Carrera: Ingeniería Informática

Denominación de la materia: Teoría de la información

Profesores a cargo: Massa, Stella Maris; Spinelli, Adolfo Tomás

Emails: nievas.nahuel.1998@gmail.com mikelajandroolave@gmail.com

tobiasgiacri@gmail.com

Índice

Resumen

El siguiente trabajo tiene como objetivo el análisis de una fuente de 10 000 caracteres con el fin de definirla y estudiarla. Esta fuente consiste de un alfabeto fuente {A,B,C}.

Introducción

Para el desarrollo del informe se trabajará con una fuente de 10 000 caracteres dadas por la cátedra. Ésta estará compuesta por el conjunto de símbolos B= {A,B,C} . El objetivo será analizar la distribución de probabilidades de ocurrencia de cada símbolo, como también definir el tipo de fuente con el que estamos tratando (Fuente nula o no nula).

Luego de resolver esto, se pide que se consideren del archivo cadenas de tres,cinco y siete caracteres. Identificar en cada caso el código base, su frecuencia, su cantidad de información, su entropía, su tipo del código. Además, determinar el rendimiento y redundancia de cada código. Y por último codificar los símbolos de los códigos según Huffman o Shanon-Fano y reconstruir el archivo.

Para la resolución de esta problemática se han utilizado algoritmos desarrollados en el lenguaje de programación C.

Desarrollo

Primera Parte

Determinación del tipo de fuente de información

Para poder determinar si la fuente dada es de memoria nula o no nula analizamos uno a uno los caracteres del archivo de texto dado y en base a él recabar información de las frecuencias condicionales en la muestra. Para ello consideramos que el archivo es una muestra lo suficientemente grande como para poder realizar un análisis estadístico confiable.
Decidimos plantear la solución con un algoritmo que determine las probabilidades de ocurrencia analizando todos los pares de caracteres adyacentes y contando cuántas veces ocurre cada uno de los siguientes casos: AA, AB, AC, BA, BB, BC, CA, CB, CC.

El valor de la cantidad de apariciones de cada caso obtenido fue almacenado en una matriz de la forma:

	A	B	C
A	AA	BA	CA
B	AB	BB	CB
C	AC	BC	CC

Donde las columnas representan el carácter anterior, y las filas el carácter posterior.

Luego de la ejecución del algoritmo la matriz resultante fue:

	A	B	C
A	1920	1175	716
B	1550	2773	416
C	341	791	317

A su vez creamos un vector de símbolos en el cual llevamos la cantidad de veces que cada símbolo aparecía en el archivo:

	A	B	C
Repeticiones	3811	4739	1449

Teniendo dichas cantidades en el vector y dividiéndolas por el total de ocurrencias de cada caracter, podemos obtener las probabilidades condicionales de cada par posible.

Luego, la matriz de probabilidad de ocurrencias condicionales $M_{i/j}$es:

	A	B	C
A	0,504	0,248	0,494
B	0,407	0,585	0,287
C	0,089	0,167	0,219

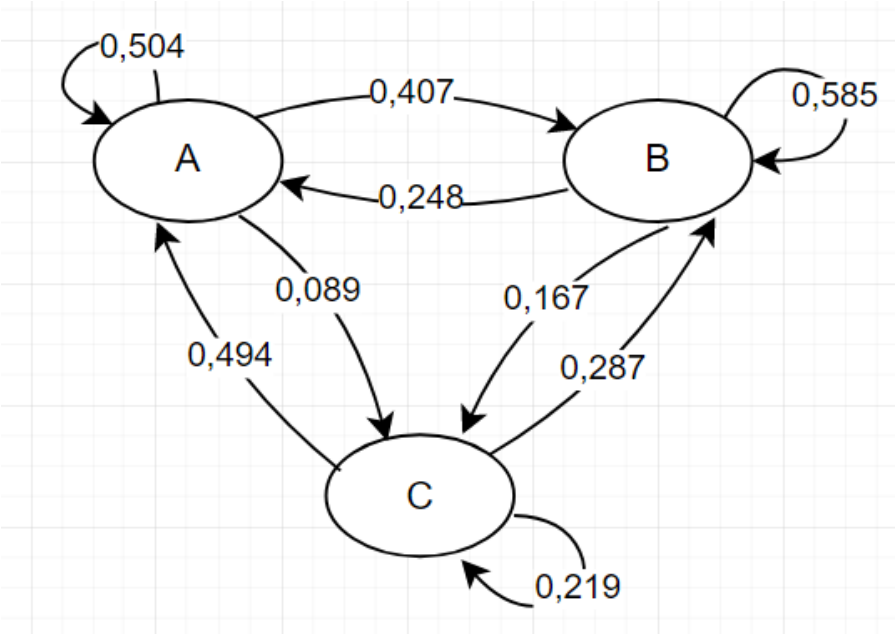
Observemos por ejemplo que si se emite el símbolo A, es más probable que el carácter siguiente a emitir sea nuevamente una A ya que su probabilidad es de 0,504 a comparación del símbolo B y C con 0,407 y 0,089 respectivamente.
Podemos afirmar entonces que estamos en frente de una fuente de memoria no nula de orden 1 donde los símbolos son estadísticamente dependientes de su carácter anterior emitido. Además, considerando que estamos tratando con una fuente de orden 1, estamos frente a una fuente markoviana.

A continuación se muestra el pseudocódigo que calcula lo explicado anteriormente:

```
Read(caracter);
while (No es fin de archivo);{
   Busco (caracter en vector);
   VectorCaracteres[posicionCaracterActual]++;
   MatrizCombinaciones[posicionCaracterActual][posicionCaracterAnterior]++;
   posCaracterAnterior=posCaracterActual;
   Read(caracter);
}
```

Ergodicidad

Ahora que sabemos que nuestra fuente es Markoviana, podemos analizar la ergodicidad de ésta. Para ello comencemos por representar la matriz de probabilidad del proceso estocástico en forma de grafo:



Notemos que es posible acceder a cualquier estado desde cualquier estado posible, por ende la fuente es ergódica.

Obtención del vector estacionario

Al ser nuestra fuente Markoviana-ergódica podemos obtener su vector estacionario. Nos basaremos en las ecuaciones de forma matricial de Chapman-Kolmogorov que nos dicen que dado un estado temporal de probabilidad en un tiempo t y un estado temporal de probabilidad en un tiempo q entonces:
P(t+q)=p(t)*p(q)

Aplicando lo anterior a nuestro caso, desarrollamos un algoritmo que multiplica nuestra matriz $M_{i/j}$ por sí misma tantas veces como sea necesaria hasta obtener una convergencia en valores casi constantes.

El algoritmo utilizado posee una tolerancia t (en este caso t=0.00001) con el fin de que la multiplicación de la matriz $M_{i/j}$ en el algoritmo finaliza cuando el cambio para cada uno de los valores de la matriz a la matriz siguiente es menor o igual a la tolerancia:

$$\forall\ i,j\ < |B| :\quad {M_{i,j}}^{n+1} - {M_{i,j}}^{n} <= 0,00001$$

El algoritmo realizó el proceso de multiplicación X Veces dando la matriz resultante:

Ergo, nuestro vector estacionario es:

	A	B	C
Probabilidad	0.381138	0.473947	0.144914

De este vector podemos interpretar que la probabilidad de que en el archivo se encuentre una letra A 0.381138, una letra B 0.473947 , y una letra C 0.144914.

Como sabíamos que la matriz sea cuadrada por , se la multiplicación matricial se pensó de la siguiente manera:

```
do{
      MatrizAnterior=MatrizActual;
      MatrizActual=MatrizNueva;
      for (i = 0; i < cantidad; i++)
         for (j = 0; j < cantidad; j++){
            for (k = 0; k < cantidad; k++)
               suma += MatrizAnterior[i][k] * MatrizActual[k][j];
            MatrizNueva[i][j] = suma;
            suma = 0;
         }
   }
   while (checkTolerancia(MatrizNueva,MatrizActual,cantidad,tolerancia));
```

La entropía de la fuente es: 1.3823. Esta se calculó a partir de la siguiente cuenta:

$$H1 = \sum_i (pi\ *) \sum_j p\,j/i\ log\frac{1}{p\,j/i} = 1.3823$$

Que es la entropía de una fuente markoviana de orden 1. En la que pi* son las probabilidades conseguidas en el vector estacionario. y pj/i son las probabilidades condicionales.

Segunda Parte

Para la continuación del trabajo se presentan 3 casos que se enumeran a continuación:

1. Alfabeto código de cadenas de 3 caracteres:
2. Alfabeto código de cadenas de 5 caracteres:
3. Alfabeto código de cadenas de 7 caracteres:

Cálculo de la entropía y de la información

En la entropía para memorias nulas, se utiliza la siguiente ecuación:

$$H1 = \sum_j p\,j/i\ log\frac{1}{p\,j/i}$$

que a diferencia de las de memoria no nula, en esta entropía no interviene el vector estacionario.

La entropía del caso 1 en el archivo es de 2.659790 y su cantidad de información acumulada de 90.888138 .

La entropía del caso 2 en el archivo es de 2.963295 y su cantidad de información acumulada de 774.949402.
La entropía del caso 3 en el archivo es de 3.167323 y su cantidad de información acumulada de 2338.431641.

Identificación de los códigos obtenidos

Como no se especifica el alfabeto fuente y tampoco la correspondencia entre las palabras de este y el alfabeto código no podemos determinar si un código es singular o no singular. En los hechos prácticos, asumimos que van a ser no singulares.

Partiendo de esta precondición, al ser palabras distintas entre sí pero de longitudes constantes (En nuestro caso 3,5 y 7) no existe la posibilidad de que una palabra sea prefijo de otra. Luego podemos afirmar que siempre va a ser unívocamente decodificable e instantáneo.

Cálculo de las inecuaciones de Kraft y McMillan, la longitud media del código y si son compactos

La inecuación de Kraft y McMillan:

$$\sum_{i=1}^{q} r^{-l_i} \leq 1$$

Se tendrá un r = 3 para los 3 casos.
Como los 3 casos presentados presentan alfabetos de longitudes constantes, se simplifican las operaciones:

- En el caso número 1, se presenta un $l_i = 3 \; \forall \; i$ y un $q = 27$.

$$\sum_{i=1}^{27} 3^{-3} = 1$$

- En el caso número 2, se presenta un $l_i = 5 \; \forall \; i$ y un $q = 243$.

$$\sum_{i=1}^{243} 3^{-5} = 1$$

- En el caso número 3, se presenta un $l_i = 7 \; \forall \; i$ y un 2187.

$$\sum_{i=1}^{2187} 3^{-7} = 1$$

Para los 3 casos, tanto la inecuación de Kraft como la de McMillan se cumplen.

Para los 3 casos, la inecuación de Kraft se cumple, esto es condición necesaria y suficiente para afirmar que existe un código instantáneo…

- En el primer caso, de longitud 3 para todas las palabras.
- En el segundo caso, de longitud 5 para todas las palabras.
- En el tercer caso, de longitud 7 para todas las palabras.

Como cada caso presenta longitudes de palabras constantes, la longitud media es para cada caso:

1. $L = 3$
2. $L = 5$
3. $L = 7$

No es compacto porque el que tiene mayor probabilidad (0.16) y no cumple con la condición L⩾Hr (S) L:1.66

Cálculo del rendimiento y redundancia de cada código

Como la ecuación del rendimiento del código enuncia: $\eta = \frac{Hr\,(S)}{L}$ y la ecuación de redundancia enuncia: $1 - \eta = \frac{L - Hr\,(S)}{L}$ se calcularon para los 3 casos dados.

Para el caso 1 el rendimiento es: 88.65 % y la redundancia es: 11.34 %
Para el caso 2 el rendimiento es: 59.27 % y la redundancia es: 40.73 %
Para el caso 3 el rendimiento es: 45.25 % y la redundancia es: 54.75 %

Según los datos obtenidos, lo que se puede observar es que a mayor cantidad de caracteres, el rendimiento disminuye. Esto está relacionado con la entropía de cada código conseguido, ya que esta es la cantidad de información media y a medida que la cantidad de caracteres para formar una palabra aumenta, como en el rendimiento se divide entropía por longitud media, su cociente va a ser menor porque su dividendo va a ir aumentando. También tiene importancia que el alfabeto está compuesto por pocos caracteres, lo que provoca que con códigos de menos caracteres, se pueda aprovechar mejor y transmitir mayor cantidad de información.

Codificación de los códigos a partir de Huffman

Hemos decidido hacer la codificación a partir de Huffman. Resultando en 3 archivos, uno para cada longitud, estos poseen determinadas características.

Observación: La codificación resultante para cada palabra estará formado tal que :

"***α*** seguido de un bit con valor uno", donde ***α*** = bits con valor cero.

La longitud de ***α*** dependerá de qué tan frecuente sea la palabra en el texto con respecto a otras, a menor frecuencia, mayor cantidad de ceros. Esto se cumple para todas las palabras exceptuando la menos probable.
Para la palabra menos probable, su código será de igual longitud a la anteúltima, pero su bit menos significativo será el opuesto.

Ejemplo:
Si existen 23 palabras, la más probable tendra el codigo binario "1", la anteúltima menos probable tendrá 22 ceros "00000000000000000000" y la ultima tendra un código binario de igual longitud, pero el ultimo bit en 1 "00000000000000000001"

Palabra	Frecuencia	Código huffman
BBB	X	1 —> ***α***=''
ABA	X1	01 —> ***α***='0'
ABB	X2	001 —> ***α***='00'
…	…	….
CAC	X(N-1)	000000000000000000 —> ***α***=(N-1 CEROS)
CCC	X(N)	000000000000000001 —> ***α***=(N-1 CEROS)

Caso 1:
Nombre de archivo: "huffman-3.dat"

Palabra	Frecuencia	Código huffman
BBB	536	1
ABA	309	01
ABB	292	001
	…	….
CAC	22	0000000000000000000000000
CCC	20	0000000000000000000000001

Caso 2: “huffman-5.dat”

Palabra	Frecuencia	Código huffman
BBBBB	112	1
ABBBB	68	01
BBBBA	56	001
	…	….
CAC	22	(209 CEROS)
CCC	20	(208 CEROS)+’1’

Caso 3: “huffman-7.dat”

Palabra	Frecuencia	Código huffman
BBBBBBB	20	1
ABBBBBB	19	01
BBBBBBA	17	001
	…	….
BCCCCAA	1	(669 CEROS)
CABBBAB	1	(668 CEROS)+’1’

A partir de los datos obtenidos se generó un archivo en donde se codifica el archivo dado según el algoritmo de Huffman. Este archivo resultante está formado por un header y un body. En el header se presenta la información necesaria para la decodificación del archivo. Su estructura es como sigue:

Longitud palabras del bloque	Cantidad de palabras del alfabeto=N	
Palabra 1	Longitud del código de huffman 1	Código huffman 1
Palabra 2	Longitud del código de huffman 2	Código huffman 2
...	...	...
Palabra N	Longitud del código de huffman 3	Código huffman 3

Es necesario destacar que el tamaño del header es muy grande cuando el alfabeto tiene muchas palabras. Entonces, si lo que queremos es compactar un archivo pequeño, puede que lo que ahorramos en compactación lo perdamos en la escritura del header. Esto justifica

su uso solo para archivos extensos, y donde no haya equiprobabilidad entre las distintas palabras.

Una de las cosas que nos dimos cuenta durante su implementación, es que a los códigos más largos son los códigos que tienen menos probabilidades de aparecer, en contraparte los códigos más cortos son los que aparecen más seguido, marcando una clara compresión para cada archivo.

Conclusiones

Para finalizar, durante todo este trabajo se fueron tocando todos los temas vistos en clases hasta la fecha, y se fueron poniendo en práctica. Durante el transcurso de la primera parte se analizó si la fuente era nula o no nula La distribución de probabilidad condicional nos permite concluir que estamos frente a una fuente no nula, ya que las distribuciones no condicionales no eran equiprobables. A partir de esto, como resultó no nula, se verificó si era ergódica o no y se calculó su estado estacionario.
En la segunda parte el objetivo principal era poder crear, a partir de los datos dados y las condiciones dadas, una codificación que nos permita compactar el archivo. Al implementar el algoritmo de huffman logramos reducir los archivos en su tamaño incluyendo la cabecera con las instrucciones acerca de cómo leerlo. También se pudo concluir que la utilización de este método es útil para compactar un archivo cuando los códigos tienen pocos caracteres. A medida que crecían los caracteres, la compactación empezó a ser menor, debido a que la cantidad de palabras del alfabeto crecía, haciendo que el header tomará mayor relevancia en el tamaño del archivo. Esto lo pudimos deducir antes, ya que su rendimiento iba disminuyendo cuando la longitud de los caracteres aumentaba.

Repositorio

https://github.com/tobiasmdp/teoria-informacion