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

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

Índice

Resumen

En el presente informe se expondrán temas relacionados a las fuentes de información, fuentes de memoria nula y fuentes de Markov. Se considerarán casos experimentales con el fin de analizar la variabilidad de los resultados en características como la entropía de una fuente, probabilidades de apariciones de un símbolo de una fuente, rendimiento y redundancia de un código, entre otros.

Para la comprensión de este informe se detallan a continuación los temas a tratar durante el informe:

- Fuentes de memoria nula: Cantidad de información y entropía.
- Fuente de Markov o de memoria no nula: Cantidad de información, cálculo del vector estacionario y entropía. Ecuaciones de estados de Chapman-Kolmogorov
- Códigos y codificación de fuentes de información
- Inecuación de Kraft y McMillan. Longitud media del código
- Códigos compactos. Redundancia y rendimiento de un código. Algoritmo de codificación de Huffman

Introducción

La utilización de las fuentes de información es hoy en día un proceso que se realiza en todo momento. En general, los sistemas informáticos transmiten información constantemente y está en su responsabilidad utilizar la mejor manera de realizar esa tarea, con el fin de

optimizar los tiempos de envío, eficiencia de almacenamiento o evitar la pérdida de la información.

Para el desarrollo del informe se trabajará con una muestra de 10 000 caracteres dadas por la cátedra. Esta representa un mensaje enviado por una fuente de información y 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 y 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. Se hicieron 2 códigos, uno para la primera parte y otro para la segunda, en el código de la segunda parte se reutiliza la codificación de la primera parte. A medida que se avanza con el trabajo, se irán presentando las ecuaciones pertinentes y desarrollando los conceptos teóricos-prácticos.

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

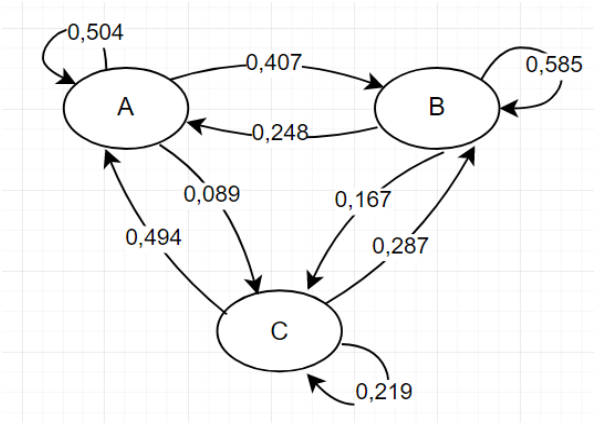
Observemos por ejemplo que si se emite el símbolo A, es más probable que el carácter siguiente a emitir sea nuevamente una A ya que su probabilidad es de 0,504 a comparación del símbolo B y C con 0,407 y 0,089 respectivamente.
Podemos afirmar entonces que estamos en frente de una fuente de memoria no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

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)

La matriz $M_{i/j}$ obtenida representa un estado temporal de una cadena de markov. 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 5 veces dando la matriz resultante:

	A	B	C
A	0.381138	0.381138	0.381138
B	0.473947	0.473947	0.473947
C	0.144914	0.144914	0.144914

En esta matriz final, los valores en cada fila se mantuvieron casi idénticos. Se logró una convergencia.

Luego, 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 es 0.381138, una letra B es 0.473947 , y una letra C es 0.144914.

La multiplicación matricial se realizó 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));
```

Entropía

A continuación se calculará el valor de la entropía de la fuente markoviana de orden 1 a partir de la siguiente fórmula

$$H1 = \sum_i (p_i{}^*) \sum_j p_{j/i} \; log_r \left(\frac{1}{p_{j/i}}\right)$$

Donde pi* son las probabilidades conseguidas en el vector estacionario. y pj/i son las probabilidades condicionales de la matriz $M_{i/j}$.

$$H1 = 0.8721$$

Segunda Parte

En esta segunda parte 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:

Estos casos se aplicarán sobre la muestra de 10000 caracteres.

Aclaración: En el archivo de texto con la muestra no se encuentran todas las combinaciones posibles para 5 y 7 caracteres. Por lo tanto se considera como precondición que las combinaciones faltantes que no aparecen en el archivo de texto no pueden ser generadas por la fuente de información, sino que solo puede generar las que se observan en el archivo.

Cálculo de la entropía y de la cantidad de información

Anteriormente llegamos a la conclusión de que estábamos tratando con un mensaje emitido por una fuente de memoria no nula. Sin embargo, para fines prácticos, se considerará que las fuentes que siguen a continuación serán de memoria nula.

En el cálculo de la entropía para fuentes de información de memorias nula, se utiliza la siguiente ecuación:

$$H1 = \sum_i p_i \, log_r \left(\frac{1}{p_i}\right)$$

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

Caso	Entropía obtenida
1	2.659790
2	4.341156
3	5.610097

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 inecuación de Kraft y McMillan es la siguiente:

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

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 $q = 27$.

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

- En el caso número 2, se presenta un $l_i = 5 \; \forall \; i$ y un $q = 210$.

$$\sum_{i=1}^{210} 3^{-5} = 0.864197 \leq 1$$

- En el caso número 3, se presenta un $l_i = 7 \; \forall \; i$ y un q = 670.

$$\sum_{i=1}^{670} 3^{-7} = 0.306355 \leq 1$$

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.

La longitud media del código

La longitud media de un código está dada por la fórmula

$$\sum_{i=1}^{q} p_i l_i$$

En esta situación por precondición sabemos que cada caso presenta longitudes de palabras constantes. Entonces la longitud media es para cada caso:

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

Compactación de código:

La condición suficiente para afirmar que un código es compacto es

$$H_r = L$$

Realizando sustituciones en la fórmula, se llega a que para que un código sea compacto, debe cumplir que:

$$l_i = \quad log_r\left(\frac{1}{P_i}\right)(Se\ aproxima\ al\ primer\ valor\ entero\ mayor)$$

Caso 1: $\forall\ i\ /\ i\ \geq 1\ \wedge\quad i\ \leq 27$

Caso 2: $\forall\ i\ /\ i\ \geq 1\ \wedge\quad i\ \leq 210$

Caso 3: $\forall\ i\ /\ i\ \geq 1\ \wedge\quad i\ \leq 670$

Nótese que se realiza una aproximación por función techo en el lado derecho de la ecuación. Esto se debe a que las longitudes de palabras son valores enteros, mientras que el logaritmo nos puede dar un valor decimal. Además, debido a que un código es compacto si y sólo si no existe otro código con longitud media menor a él, se puede asumir que el valor mínimo entero posible para esa palabra es la aproximación por función techo.
ACLARACIÓN: Existe la posibilidad de que esta aproximación provoque errores debido a problemas con el redondeo, pero a fin de facilitar el cálculo, utilizamos esta premisa.

Los resultados obtenidos determinan que para todos los casos al menos una palabra no cumple la condición de compactación, por lo tanto no se puede afirmar que el código sea compacto.

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: 86.82 % y la redundancia es: 13.18 %
Para el caso 3 el rendimiento es: 80.14 % y la redundancia es: 19.86 %

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 ésta 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.

Pseudocódigo utilizado para la obtención de los códigos de Huffman:

```
while (no haya 1 solo elemento){
      buscoMinimos(mínimo 1,mínimo 2);
            nuevoElemento(árbol,Frecuencia mínimo 1+Frecuencia Mínimo 2,puntero mínimo 1,puntero mínimo 2);
      insertarNuevoElementoOrdenadaAscendente(NuevoElemento);
  }
```

Caso 1:

Código	Código de Huffman	Código	Código de Huffman	Código	Código de Huffman
BBB	001	ABA	01001	ACA	100001
AAA	110	CAA	01101	CBA	0000000
ABB	111	CAB	10001	BAC	0000001
AAB	0001	ABC	10010	ACB	0110000
BBA	0101	BCB	000001	CBC	0110001
BAA	0111	CBB	000010	CCC	1001100
BAB	1010	AAC	000011	CCB	1001101
BBC	1011	CCA	011001	ACC	1001110
BCA	01000	BCC	100000	CAC	1001111

El caso 2 y caso 3 no serán mostrados en el trabajo práctico por la cantidad de códigos que tiene cada uno, 210 y 670 respectivamente.

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:

(32 bits) Longitud palabras del bloque	(32 bits) Cantidad de palabras del alfabeto=N	
(M * 8 bits, donde M será la longitud de palabra) Palabra 1	(32 bits) Longitud en bits de código de huffman	(32 bits) Código de huffman 1
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

(32 bits) Tamaño en bits del body		
Body compuesto por el archivo codificado en Huffman en binario.		

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.

Caso	Header	Body	Total
1	0.305 KB	2 KB	2.03 KB
2	2.738 KB	1.63 KB	4.36 KB
3	10.062 KB	1.25 KB	11.3 KB

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. 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.

Conclusiones

Para finalizar, durante todo este trabajo se abordaron 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 permitió concluir que estamos frente a una fuente no nula, ya que las probabilidades condicionales no eran equiprobables. A partir de esto, se verificó si era ergódica o no y se calculó su estado estacionario.

En la segunda parte, a partir del mismo archivo, se presentaban tres casos, en los que había que analizar para cada uno su entropía, que tipo de código era, su compactación, su rendimiento y redundancia. Por último se pedía crear 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 y cuando la aparición de las palabras no son equiprobables.