Teoría de la Información

Trabajo Integrador N° 2:

Unidad IV desde Códigos de Huffman hasta Unidad V inclusive- Informe

Grupo 6

Integrantes:

Aguilera Marcos marcos.aguilera.7.13@gmail.com

Castorina Matías matiascastorina@gmail.com

Noseda Demian nosedademian@gmail.com

Índice

Resumen

En este informe vamos a tratar en primer lugar con la codificación para la compresión de datos mediante los métodos de Huffman, Shannon-Fano y RLC, para dicho trabajo se desarrolló un programa en java que aplica dichos métodos para comprimir archivos de texto.

En la segunda parte del informe trataremos con canales de transmisión de información, para los mismos se desarrolló un programa en java que, a partir de un canal de información dado y las probabilidades de las entradas calcula diferentes parámetros relevantes para interpretar un canal de información.

Primera parte: Codificación y compresión

Introducción

Al hablar de compresión de datos, se hace referencia a reducir el tamaño de un archivo. Esto permite principalmente ahorrar espacio al guardarlo y ahorrar tiempo al transmitirlo. Para lograr esto se emplea un programa capaz de comprimir y descomprimir el archivo de texto hecho en java, este programa solicita la dirección del archivo origen y permite seleccionar que tipo de compresión se realizara (Huffman, Shannon-Fano o RLC).

Al comprimir obtendremos un archivo que no solo contiene los datos codificados mediante el método correspondiente si no también un diccionario para la decodificación.

Al descomprimir el archivo se utiliza el diccionario guardado en el comprimido y a partir de la codificación se obtiene nuevamente el archivo de texto sin pérdida de información.

Los métodos utilizados para generar los algoritmos de Huffman y Shannon-Fano son de tipo recursivo, mientras que el método utilizado para codificar mediante RLC es de tipo iterativo.

Método de Huffman

El método Huffman se basa en crear un código compacto óptimo que codifique los símbolos del archivo en un alfabeto binario, utilizando {0,1}.

A partir de esto se da la compresión dado que en el alfabeto original cada símbolo utiliza 8 bits para representarse sin importar su probabilidad, y en la codificación de Huffman cada símbolo utiliza una cantidad de bits óptima obtenida a partir de su probabilidad de aparición.

El código de Huffman se obtiene a partir de un proceso recursivo que en cada paso agrupa los símbolos menos probables para formar un nuevo símbolo. Obteniendo así una denominada fuente reducida, sobre la cual se repite la fusión de los dos símbolos menos probables hasta llegar a una fuente reducida de sólo dos símbolos. Comenzado por esta última fuente reducida se construye el código compacto para cada fuente, asignando en la última un cero y un uno en los símbolos respectivamente. La fuente reducida anterior se codifica copiando las palabras código al símbolo de precedencia. Si el símbolo precede de la fusión de dos, la palabra se copia a los dos que se originaron, y para diferenciarlas añadiendo el símbolo 0 a una, y 1 a la otra. Así en la última iteración se obtiene un código compacto óptimo para la fuente original. La longitud media Lm del código Huffman asociado a la fuente S, sin memoria, extendida a orden n, está limitado: $H(S) \leq Ln /n < H(S)+1/n$.

Para realizar esta compresión en nuestro caso lo primero que se realiza es una lectura del archivo en formato UTF-8 y a partir del mismo se guarda una tabla con los símbolos y las probabilidades relativas.

A partir de la tabla anterior se genera una lista ordenada (en un vector), con los nodos que estarán posteriormente en el árbol de Huffman, los hijos de los nodos en esta etapa no contienen nada.

Luego se procede a formar el árbol de Huffman, para esto sumamos los dos nodos con las probabilidades menores y creamos un nuevo nodo unión de estos cuyos hijos serán los nodos que agrupamos en él. Este nuevo nodo se agrega a la lista en la posición que le corresponda y se procede de igual manera hasta que la lista contenga un solo nodo (la raíz del árbol).

Ahora que ya está formado el árbol lo recorremos de forma recursiva hasta llegar a las hojas manteniendo un String en el cual al desplazarnos a la izquierda se agrega un 0 y en su lugar al desplazarnos a la derecha se agrega un 1. Al llegar a una hoja se inserta el nodo con la codificación actual de la recursividad en una lista que será el diccionario de la codificación.

Para comprimir el archivo en primer lugar se guarda el diccionario para utilizarlo en la descompresión del archivo, luego a partir del diccionario se realiza una nueva lectura del archivo original y se guarda la representación en bits correspondiente a cada símbolo ordenado en una estructura de bits. Al terminar la lectura del archivo origina se guarda la estructura de bits en el comprimido y termina la compresión.

Para descomprimir el archivo leemos en primer lugar el diccionario de datos del comprimido y luego la estructura de bits. A partir de la estructura de bits se van consumiendo los mismos cotejando con el diccionario si la secuencia representa algún símbolo, de ser así se escribe dicho símbolo en un archivo de texto nuevo, se limpia la secuencia anterior de bits y se continúa leyendo bits hasta descomprimir el archivo completamente.

Método de Shannon-Fano

El método de Shannon-Fano se base en crear un código compacto subóptimo que codifique los símbolos del archivo en un alfabeto binario {0,1}.

A partir de esto se da la compresión dado que en el alfabeto original cada símbolo utiliza 8 bits para representarse sin importar su probabilidad, y en esta codificación cada símbolo utiliza una cantidad de bits subóptima a partir de su probabilidad de aparición.

El código Shannon-Fano solo alcanza una cota de $L \leq H(S) + 2$ (donde L es la longitud media del código y H(S) la entropía), por lo tanto, se dice que es subóptimo. Para obtenerlo se ordenan los símbolos según su probabilidad en forma decreciente y se dividen los símbolos en dos subconjuntos lo más equiprobables posibles, a los que se le asigna un bit 0 o 1 respectivamente. Este procedimiento se repite para todos los subconjuntos hasta que todos los subconjuntos tengan un elemento.

Para realizar esta compresión al igual que en Huffman lo primero que realizamos es una lectura del archivo en formato UTF-8 y a partir del mismo se guarda una tabla con los símbolos y las probabilidades relativas.

Luego al igual que en Huffman creamos un vector ordenado de mayor a menor probabilidad con los nodos que representan a cada símbolo y su probabilidad.

A partir del vector ordenado anterior se procede a formar el diccionario. Para ello se utiliza una función recursiva en la que en cada iteración se divide el vector a la mitad de acuerdo a la probabilidad de aparición de los símbolos y a la mitad izquierda se le agrega un 1 en su representación binaria, mientras que a la mitad derecha se le agrega un 0. Esta división en mitades se repite hasta que ya no se puedan dividir (la subdivisión solo contiene un nodo) y en ese momento se toma el símbolo del nodo y su representación para guardarla en el diccionario.

La compresión y descompresión del archivo se realiza de igual manera que la descripta anteriormente en Huffman dado que lo único necesario para las mismas es el diccionario que funciona igual en ambos métodos.

RLC

El método de RLC codifica identificando secuencias de datos con el mismo valor consecutivo que son almacenadas como un único valor más su número de apariciones. Este tipo de compresión, al igual que Huffman y Shannon-Fano, es una compresión sin pérdidas.

Para comprimir utilizando RLC primero se abre el archivo deseado en formato UTF-8 al igual que en los anteriores. Luego se lee símbolo a símbolo para ir contando sus apariciones consecutivas y se los guarda en el archivo comprimido como una dupla “Símbolo”,” Apariciones”. En nuestro caso para mejorar el caso de una única aparición consecutiva evitamos guardar la cantidad de apariciones dejando implícito que se trata de una sola, cada par está separado por un espacio. De esta manera se repite la codificación hasta consumir todos los símbolos del archivo de texto y guardarlos en el archivo de texto comprimido.

Este método no requiere de guardar ningún tipo de diccionario en el comprimido, y tampoco de recorrer más de una vez el archivo de texto original.

Para descomprimir se lee en primer lugar el símbolo y luego se continúa leyendo hasta encontrar un espacio. Se emiten tantas repeticiones del símbolo como las que hayan sido leídas del comprimido y si la lectura no tenía un numero de repeticiones se toma implícitamente como una única repetición. Se continúa de la misma manera con el siguiente par hasta concluir la descompresión.

Conclusiones de la compresión

Como se puede apreciar en las tablas 1.1 y 1.2 anexadas en el apéndice, al comprimir los archivos de texto mdp-español.txt y mdp-frances.txt lo primero que notamos es que RLC en lugar de reducir su tamaño, lo aumenta. Esto tiene sentido dado que RLC es un método de compresión útil cuando se trata de fuentes de información que repiten secuencias de un mismo símbolo y en el caso de las palabras en nuestros archivos de texto no es así, por lo que para representar cada símbolo está utilizando más lugar que en el archivo original.

Luego si pasamos a tratar exclusivamente con Huffman y Shannon-fano, podemos ver como Huffman en ambos casos da una mayor tasa de compresión que Shannon-Fano. Esto se debe a que el código de Huffman es óptimo mientras que el de Shannon-Fano es subóptimo. A partir de dicha idea podemos ver como también esto representa que Shannon-Fano tenga una mayor redundancia por lo que nos aporta menos cantidad de información por byte en la compresión.

También podemos apreciar en los resultados que la tasa de compresión es relativamente baja, no alcanza siquiera 1,1 en ningún caso (teniendo en cuenta Shannon-Fano y Huffman exclusivamente). Concluimos que esto se debe a nuestra forma de persistir el diccionario en el archivo comprimido, dado que al remover el diccionario la tasa de compresión aumenta hasta llegar a 1,75 aproximadamente. Por ello podemos concluir que estas compresiones aumentarán su eficiencia cuanto más grande sea el archivo de texto dado que el tamaño del diccionario se mantendrá constante disminuyendo así el porcentaje que representa del archivo comprimido final.

En cuanto a los idiomas podemos concluir que el francés tiene una mayor cantidad de símbolos para representar lo mismo dado que el archivo original tiene un mayor tamaño y además esto se ve reflejado en el tamaño del diccionario que es mayor en el francés. Pero aparte de esto podemos ver que ambos idiomas se comprimen a una tasa similar y no tienen grandes diferencias de tamaño.

Segunda parte: Canales de comunicación

Introducción

Llamamos canal de información al medio por el que se transmite la información desde la fuente de información al destino. La señal se codifica antes de ingresar al canal y se decodifica a la salida. Puede existir ruido que perturba la transmisión (distorsiones).

Estos canales de información vienen determinados por un alfabeto de entrada A = {ai}, i = 1, 2, ..., r; un alfabeto de salida B = {bj}, j = 1, 2, ..., s; y un conjunto de probabilidades condicionales P (bj/ai).

En nuestro caso a partir de la matriz de probabilidades condicionales P, y las probabilidades a priori P(ai), mediante un programa codificado en lenguaje java realizaremos los distintos tipos de cálculos útiles para interpretar un canal de comunicación.

Equivocación

H(A/B) recibe el nombre de Equivocación de A con respecto a B a través del canal. Este parámetro nos da una medida de la información que queda en A después de observar B, así como la perdida de información sobre A causada por el canal y la cantidad de información de A que no deja pasar el canal.

La fórmula que define el cálculo de esta propiedad es la siguiente:

$$H(A/B)=\sum_{B} P(b)\,H(A/b)=\sum_{A,B} P(a,b)\log\left(\frac{1}{P(a/b)}\right)$$

Para realizar esta cuenta en nuestro programa a partir de las probabilidades a priori y la matriz del canal debemos realizar cálculos intermedios.

En primer lugar, debemos obtener el termino P(b) que representa las probabilidades de la salida. Esta probabilidad la obtendremos de forma teórica a partir de la siguiente formula:

$$P(bj) = \sum_{i=1}^{r} P(ai)\,Pij \qquad \forall\, j = 1..s$$

Se implementó un código en java representado por el siguiente seudocódigo para calcularlo:

```
m = cantidad de entradas
n = cantidad de salidas
canal = matriz del canal
probabilidadPriori = vector con las probabilidades a priori
probabilidadSalida = nuevo vector con las probabilidades de salidas
j = 0
mientras que (j < n) hacer
{
      probabilidadSalida[j] = 0
      i = 0
      mantras que (i < m) hacer
      {
            probabilidadSalida[j] = probabilidadPriori[i] * canal[i][j]
            i = i + 1
      }
      j = j + 1
}
```

De esta forma obtenemos un vector con las probabilidades de las salidas, a partir del mismo pasamos al siguiente termino necesario para calcular la equivocación H(A/b) llamado entropía a posteriori.

Esta entropía se calcula a partir de la siguiente formula:

$$H(A/bj) = \sum_{A} P(a/bj) \log \frac{1}{P(a/bj)}$$

Pero nuevamente para calcular este término se requiere de un cálculo dado que no poseemos la probabilidad a posteriori P(a/bj), por lo que la calcularemos a partir de la siguiente formula:

$$P(ai/bj) = \frac{p(bj/ai)\,P(ai)}{P(bj)} = \frac{P(bj/ai)\,P(ai)}{\sum_{k=1}^{r} P(bj/ak)\,P(ak)}$$

Como vemos también utiliza las probabilidades de salida P(bj) calculadas anteriormente, junto con las probabilidades de entrada y la matriz de canal.

Se implementó un código en java representado por el siguiente seudocódigo para calcularlo:

```
m = cantidad de entradas
n = cantidad de salidas
canal = matriz del canal
probabilidadPriori = vector con las probabilidades a priori
probabilidadSalida = vector con las probabilidades de salidas
probabilidadPosteriori = nueva matriz con las probabilidades a posteriori
j = 0
mientras que (j < n) hacer
{
      i = 0
      Mientras que (i < m) hacer
      {
            probabilidadPosteriori[i][j] = canal[i][j] * probabilidadPriori[i] /
            probabilidadSalida[j]
            i = i + 1
      }
      j = j + 1
}
```

Ahora que ya tenemos la probabilidad a posteriori podemos pasar a calcular la entropía a posteriori para ello utilizamos un código en java similar al siguiente seudocódigo:

```
m = cantidad de entradas
n = cantidad de salidas
entropiaPosteriori = nuevo vector de entropías a posteriori
probabilidadPriori = vector con las probabilidades a priori
probabilidadSalida = vector con las probabilidades de salidas
probabilidadPosteriori = matriz con las probabilidades a posteriori
j = 0
mientras que (j < n) hacer
{
      entropiaPosteriori[j] = 0;
      i = 0
      Mientras que (i < m) hacer
      {
            si (probabilidadPosteriori[i][j] = 0)
            {
                  cantInfoPosteriori = 0;
            } si no
            {
                  cantInfoPosteriori = (-1) * log(probabilidadPosteriori[i][j])
                  / log (2)
            }
```

```
                entropiaPosteriori[j]      =      entropiaPosteriori[j]      +
                probabilidadPosteriori[i][j] * cantInfoPosteriori
            i = i + 1
        }
        j = j + 1
}
```

Ahora que tenemos todos los términos podemos pasar al seudocódigo de la función real implementada en java para calcular la equivocación H(A/B):

```
Equivocación = 0;
n = cantidad de salidas
entropiaPosteriori = vector de entropías a posteriori
probabilidadSalida = vector con las probabilidades de salidas
j = 0
mientras que (j < n) hacer
{
      equivocación     =     equivocación     +     probabilidadSalida[j]     *
      entropiaPosteriori[j];
}
```

Información mutua

Es la cantidad de información que se obtiene de A gracias al conocimiento de B.
Nos dice la incertidumbre sobre la entrada del canal que se resuelve observando la salida del mismo. También representa la cantidad de información de A que atraviesa el canal.

Se denomina Información mutua (de A y B), o información mutua del canal a la propiedad calculada en la siguiente ecuación:

$$I(A,B)=\sum_{A,B} P(a,b)\log\left(\frac{P(a,b)}{P(a)P(b)}\right)$$

Si se analiza algebraicamente podemos llegar a que la información mutua está representada por la siguiente diferencia:

$$I(A,B)= H(A)-H(A/B)$$

A partir de esta ecuación resulta muy simple calcular la información mutua en el programa que generamos dado que tanto el termino H(A) (entropía a priori) y H(A/B) (equivocación) fueron calculados anteriormente, por ello solo realizamos una resta y obtenemos la información mutua.

Conclusiones de los cálculos sobre canales de comunicación

Como se puede apreciar en las tablas 2.1, 2.2 y 2.3 anexadas en el apéndice, a la hora de comparar las equivocaciones entre los canales encontramos resultados muy diferentes. Pero comparar las equivocaciones entre si no nos da una idea de que canal es o no mejor que el otro por lo que realizamos una comparación mejor teniendo en cuenta tanto la equivocación como la información mutua, para a partir de estos datos analizar que canal de comunicación cumple mejor su función.

Si tenemos en cuenta que la suma entre la equivocación y la información del canal da como resultado la entropía a priori, podemos, comparando estos dos parámetros y viendo cual es mayor en cada caso tener una idea de si un canal nos aporta más información de la entrada conociendo la salida o si en su lugar es mayor la cantidad de información de la entrada que se pierde en el canal.

Pasando a los casos puntuales vemos como en el canal 2 (tablas 2.1), la equivocación es mayor que la información mutua, esto mismo sucede con el canal 2 (tablas 2.2), pero si observamos el canal 3 (tablas 2.3) vemos como la información es bastante mayor que la información mutua. Teniendo en cuenta estos resultados podemos decir que el canal 3 es el que mejor representa la información respecto de la entrada, en otras palabras, es el canal en el que se obtiene la mayor cantidad de información respecto de A partiendo del conocimiento de b.

Ahora si queremos comparar los canales 1 y 2 tendremos que ver porcentualmente respecto de la entropía a priori (H(A)), cual tiene mayor información mutua y cual mayor equivocación.

Si realizamos el cociente I (A,B) / H(A) vemos como en el canal 1 obtenemos un resultado de 0,4, mientras que en el canal 2 el resultado es de 0,4347. A partir de estos dos resultados podemos concluir entonces que el canal 2 es ligeramente mejor que el primero y que el canal 1 es el que más ruido aporta a la salida por ser el de mayor equivocación respecto de la entropía a priori (H(A/B) / H(A)).

Conclusión

Como conclusión respecto a la primera parte del informe pudimos apreciar como el algoritmo de Huffman y Shannon-Fano se aplicaban mucho mejor que RLC a nuestro caso de compresión de un archivo de texto escrito en un idioma y sin repetición de secuencias de símbolos largas. Además, comparando estos dos vimos que Huffman es ligeramente mejor dado que elige de forma óptima los códigos para cada símbolo.

Respecto a la segunda parte del trabajo, a la hora de analizar los canales de información nos encontramos con que uno de los mismos era mucho mejor que los otros dos si tenemos en cuenta la cantidad de información que obtenemos de la entrada a partir la salida. Mientras tanto los otros dos canales de comunicación pudimos compararlos y ver como uno tenía un aporte de ruido ligeramente mayor que el otro siendo este el peor en cuanto a la cantidad de información que obtenemos de la entrada al conocer la salida.

Apéndice

Tablas de compresión de archivos

Tabla 1.1

mdp-español	Original	Huffman	Shannon-Fano	RLC
Tamaño	5044 bytes	4608 bytes	4626 bytes	9982 bytes
Tasa de compresión	-	1,0946	1,0904	0,6157
Tamaño sin diccionario	-	2873 bytes	2905 bytes	-
Tasa de compresión sin diccionario	-	1,7557	1,7363	-
Rendimiento	-	0,9912	0,9803	-
Redundancia	-	0,0088	0,0197	-

Tabla 1.2

mdp-frances	Original	Huffman	Shannon-Fano	RLC
Tamaño	5237 bytes	4816 bytes	4836 bytes	10213 bytes
Tasa de compresión	-	1,0874	1,0829	0,6393
Tamaño sin diccionario	-	2961 bytes	2985 bytes	-
Tasa de compresión sin diccionario	-	1,7687	1,7544	-
Rendimiento	-	0,9944	0,9861	-
Redundancia	-	0,0056	0,0138	-

Tablas de canales de comunicación

Canal 1

Tablas 2.1

probabilidades a priori	
Ai	P(Ai)
A1	0,6
A2	0,4

Matriz del canal					
	B1	B2	B3	B4	B5
A1	0,5	0	0	0,4	0,1
A2	0,5	0	0,4	0	0,1

Equivocación	0,58257
Inf. Mutua	0,38838

H(A)	0,97095

	B1	B2	B3	B4	B5
H(A/b)	0,970951	0	0	0	0,970951

Canal 2

Tablas 2.2

probabilidades a priori	
Ai	P(Ai)
A1	0,25
A2	0,25
A3	0,25
A4	0,15
A5	0,1

Matriz del canal					
	B1	B2	B3	B4	B5
A1	0,1	0,2	0,2	0,4	0,1
A2	0,5	0,4	0	0	0,1
A3	0	0	0	0,4	0,6
A4	0	0	0	1	0
A5	0	0	0	0	1

Equivocación	1,26782
Inf. Mutua	0.97492

H(A)	2,24274

	B1	B2	B3	B4	B5
H(A/b)	0,650022	0,918296	0	1,556657	1,625815

Canal 3

Tablas 2.3

probabilidades a priori	
Ai	P(Ai)
A1	0,7
A2	0,1
A3	0,1
A4	0,1

Matriz del canal			
	B1	B2	B3
A1	1	0	0
A2	0	1	0
A3	0	0	1
A4	0	0,5	0,5

Equivocación	0,27549
Inf. Mutua	1,08129

H(A)	1,35678

	B1	B2	B3
H(A/b)	0	0,918296	0,918296