Guión de prácticas nº 2:

Entropía y modelado de los datos

Rubén Dugo Martín

Compresión y Codificación de Datos

Curso 2007-2008

Universidad de Granada

1. **Calcular la entropía de primer orden de varios ficheros de ejemplo (varias imágenes, textos, binarios) estimando las probabilidades a partir de los datos en el fichero y suponiendo que las letras del alfabeto son independientes e idénticamente distribuidas. ¿Qué significan los números que obtenemos con relación a su posible compresión? ¿Por qué diferentes ficheros, aún del mismo tipo (por ejemplo, texto) nos dan valores de la entropía de primer orden diferentes?**

   Escoger una imagen binaria y convertirla a formato gray (ver guión 1) para que las cabeceras no nos alteren las estadísticas de la imagen. Al convertir la imagen binaria a formato gray lo que hacemos es generar un fichero en que cada bit (que representa un píxel en el imagen binaria) se convierte en un byte con el valor 255 para el uno y el valor 0 para el cero y se eliminan las cabeceras. ¡Cuidado! El fichero resultante puede tener más de 4Mb. Esta conversión nos permite trabajar con los programas que tenemos para calcular la entropía. Computar su entropía de primer orden.

En esta tabla se puede observar los distintos ficheros que he escogido junto al cálculo de su entropía:

ENTROPÍA DE PRIMER ORDEN PARA DIFERENTES FICHEROS	
Imágenes en escala de grises (pgm)	
Fichero	**Entropía**
camera.pgm	7.01
goldhill.pgm	7.47
Imágenes binarias (pbm)	
Fichero	**Entropía**
ptt1.pbm	0.70
ptt2.pbm	0.59
ptt3.pbm	1.17
Audio (wav)	
Fichero	**Entropía**
lacool.16bits.wav	7.61

Texto plano y texto binario (txt)	
Fichero	**Entropía**
Cinco semanas en globo - Julio Verne.txt	4.47
texto10000.txt	1.00
Imágenes binarias convertidas a gray 8 bits	
Fichero	**Entropía**
ptt1.pbm	0.23
ptt2.pbm	0.26
ptt3.pbm	0.41

La relación que existe entre los valores de entropía obtenidos y la posible compresión de los datos es que dichos datos se podrían comprimir con aproximadamente *n* bits por símbolo (o letra del alfabeto), siendo *n* el valor de la entropía.

La razón de que ficheros del mismo tipo nos den diferentes valores de entropía (aunque aproximados) es que la frecuencia de aparición de cada uno de los símbolos depende del formato pero también del contenido del fichero, esto entra en contraste con el enunciado del ejercicio en el que suponemos que *las letras del alfabeto son independientes e idénticamente distribuidas* .

Algo que me extrañó de los valores de entropía obtenidos en el ejercicio es que dichos valores en las imágenes binarias (pbm) eran anormalmente bajos, investigué porque podría ocurrir esto, hice una copia del fichero *entfile.c* y lo modifiqué de tal forma que imprimiera en consola el histograma de frecuencias, entonces pude observar que ocurría lo siguiente; en esos ficheros el byte *0* se repite muy por encima de los demás valores, podemos suponer que esto es una peculiaridad de este formato o del contenido de dichos ficheros, esto era la causa que la entropía de los ficheros sea mínima.

2. **Escoger una imagen en escala de grises de las usadas en el punto anterior. Calcular la entropía de segundo orden de esa imagen. Suponer que las letras del alfabeto son independientes e idénticamente distribuidas.**
   ¿Qué significa el valor obtenido? Según ese valor, ¿podemos comprimir más o menos que tomando los símbolos de uno en uno?

   Repetir el proceso para la imagen binaria escogida en el apartado anterior.

ENTROPÍA DE SEGUNDO ORDEN PARA DIFERENTES FICHEROS	
Imágenes en escala de grises (pgm)	
Fichero	**Entropía**
camera.pgm	11.12
Imágenes binarias (pbm)	
Fichero	**Entropía**
ptt1.pbm	1.12

Para la imagen en escala de grises (pgm), el valor de la entropía de segundo grado indica que podemos comprimir su alfabeto con una media de *11'2* bits por letra, como el alfabeto está codificado con *16* bits por letra, podemos afirmar que tendrá una razón de compresión del *1'44:1* siempre que codifiquemos pares de bytes, sin embargo no ocurre lo mismo si codificáramos byte a byte, en este caso necesitaríamos una media de *7'01* bits por símbolo, quedando la razón de compresión *2'28:1*.

En este caso comprimiendo byte a byte la compresión es notablemente mayor.

Para la imagen binaria (pbm) lograríamos una razón de compresión de *14'28:1* comprimiendo por pares en contraste con el método byte a byte, con el que lograríamos una razón de compresión compresión de *11'43:1*, algo menor, a diferencia del caso anterior.

3. **Utilizar el programa diferencia.c que calcula la diferencia de cada píxel de una imagen con el anterior y aplicarlo a la imagen en escala de grises escogida en el punto anterior. Notad que esto es una aplicación del modelo de Markov de primer orden descrito en el tema 2 de teoría. Calcular la entropía de primer orden sobre la imagen de diferencias. ¿Es esta entropía menor que las obtenidas en los dos puntos anteriores? ¿Por qué?**

He escogido el fichero *camera.pgm*, la entropía del fichero es *7'01*, sin embargo la entropía del fichero generado con el programa *diferencia* es *3'54*, algo menor, esto puede ser debido a que en una imagen suele ser frecuente que bytes consecutivos tengan valores similares, de esta forma las diferencias obtenidas tendrán valores bastante parecidos haciendo que la entropía sea menor.

4. **Escribir un programa que obtenga palabras de cuatro letras generando aleatoriamente (siguiendo una distribución de probabilidad) letras del abecedario.**

Para la realización de este ejercicio he implementado una biblioteca con una serie de funciones relacionadas con los diferentes apartados del ejercicio, el fichero *alpha.h* es el fichero de cabecera y de especificación de dicha biblioteca y los ficheros *alphaodds.c* y *alphagen.c* contienen las implementaciones de las funciones, para ver una breve descripción de las mismas consulte el fichero *alpha.h*.

Para utilizar la biblioteca debemos enlazar con el fichero *libalpha.a*.

Con la finalidad de comprobar el correcto funcionamiento de la biblioteca he generado diferentes diccionarios de palabras con varias longitudes utilizando las funciones de la misma, si dichos diccionarios generados contienen un número significativo de palabras es intuitivo pensar que las probabilidades se asemejarán bastante a la del fichero origen.

Utilizando hojas de cálculo he podido comprobar que esto ocurre por lo que podemos concluir que la biblioteca desarrollada funciona correctamente.

He incluido una batería de tests en el fichero *testalpha.c*.

El programa que hace uso de la biblioteca descrita es el *ej4*, el cual genera los ficheros correspondientes a cada uno de los apartados del ejercicio, los parámetros necesarios para la ejecución del programa son:

./ej4 <diccionario español> <diccionario inglés>

De forma que si situáramos los diccionarios en el mismo directorio que el programa podríamos ejecutarlo de la forma:

./ej4 PalabrasDRAE.4letras.txt 4letters.word.txt

NOTA: Podemos despreocuparnos totalmente de la compilación y enlace de los programas ya que incluyo un fichero *Makefile* que genera todo el contenido.

1) **Formar 100 palabras suponiendo que todas las letras tienen igual probabilidad de aparición y ver cuántas tienen sentido.**

Tras la ejecución del programa obtenemos el fichero *ej4.1.txt*.

Ejecutando *aspell -l es list < ej4.1.txt > ej4.1-mal.txt* obtengo en el fichero *ej4.1-mal.txt* que contiene las palabras generadas incorrectas.

Mediante *echo $[`wc -l < ej4.1.txt`-`wc -l < ej4.1-mal.txt`]* (en bash) obtengo el número de palabras correctas.

Alternativamente puedo listar las palabras correctas utilizando *diff ej4.1.txt ej4.1-mal.txt* .

Finalmente he obtenido una única palabra correcta.

2) **Calcular la probabilidad de cada letra del abecedario usando el fichero PalabrasDRAE.4letras.txt que contiene todas las palabras del Diccionario de la Real Academia Española (DRAE) de cuatro letras. Formar 100 palabras usando las probabilidades calculadas anteriormente. ¿Cuántas tienen sentido? Lo mismo para inglés.**

Tras la ejecución del programa obtenemos los ficheros *ej4.2-es.txt* y *ej4.2-en.txt*, español e inglés respectivamente.

Utilizando los mismos comandos y programas que el ejercicio anterior obtengo un total de 3 palabras correctas para español y 5 para inglés (cambiando *aspell -l en* , claro).

3) **Calcular la probabilidad de cada pareja de letras del abecedario usando el fichero PalabrasDRAE.4letras.txt que contiene todas las palabras del Diccionario de la Real Academia Española (DRAE) de cuatro letras. Formar 100 palabras de cuatro letras usando esas probabilidades. La primera letra de la palabra se generará a partir de las probabilidades generadas en el apartado 2. ¿Cuántas palabras tienen sentido?**

Tras la ejecución del programa obtenemos el fichero *ej4.3.txt.*
De la misma forma que los ejercicios anteriores obtengo 6 palabras correctas.

4) **Realizar una comparación de los resultados obtenidos anteriormente.**

Podemos observar claramente que construyendo palabras mediante parejas de letras teniendo en cuenta su frecuencia de aparición (apartado 3) hay mucha más probabilidad de obtener palabras correctas que generando aleatoriamente cada una de las letras (apartado 1) o bien seleccionando letra a letra mediante su frecuencia de aparición individual (apartado 2).

Podemos concluir que en la construcción de palabras del idioma español (con el único que hemos experimentado) es bastante frecuente encontrar pares de letras que se repiten en bastantes palabras (sílabas).