Trabajo Práctico:
Teoría de la Información

Grupo 8

Integrantes:

Herrero, Santiago
Chanes, Juan Manuel
Billordo, Mateo

Introducción

En el ámbito de la Teoría de la Información, se nos pidió el análisis de dos señales adquiridas por sensores de luminosidad ubicados en distintos telescopios distribuidos en diferentes localizaciones del país. Como parte de este estudio, nuestra tarea consiste en **implementar un enfoque computacional** que nos permita obtener la distribución de probabilidades de los valores de dichas señales, considerando cada una como una **fuente sin memoria**. Además, se nos solicita aplicar el algoritmo de Huffman para su eficiente codificación. El presente informe detalla el enfoque utilizado y los resultados obtenidos a través de esta implementación.

Además, realizamos un muestreo computacional para obtener el valor medio y la desviación de las señales. El valor medio nos proporciona una medida de la tendencia central de los datos, mientras que la desviación nos indica qué tan dispersos están los valores alrededor del valor medio. Estas medidas estadísticas nos ayudan a comprender mejor las características y propiedades de las señales que estamos analizando.

Desarrollo

Para comenzar, llevamos a cabo una implementación computacional basada en el método de Huffman. **Este algoritmo se utiliza para la codificación de información, asignando códigos más cortos a los valores de la señal que tienen mayor probabilidad y códigos más largos a los de menor ocurrencia**. De esta manera, se logra una eficiente representación de la señal.

Procedimos a calcular la distribución de probabilidades de los valores de cada señal. Esto implicó analizar la frecuencia de ocurrencia de cada valor y determinar su probabilidad correspondiente. Con esta información, generamos el árbol de Huffman el cual es óptimo y obtuvimos el conjunto de códigos asociados a los valores de la señal. Gracias a este método, **nos acercamos al valor de la entropía de la señal** lo cual nos permitió lograr una representación más compacta y eficiente de la información, maximizando la compresión sin pérdida de información.

Implementación

En primer lugar debemos calcular la distribución de probabilidades de la fuente. Como se considera que es una fuente sin memoria, simplemente debemos contar la cantidad de ocurrencias de cada símbolo y dividirla por la cantidad total de símbolos para obtener su probabilidad.

```
signal = {}
cantidad = 0
for simbolo ∈ archivo:
        cantidad++
        if simbolo ∈ lista      # si existe en el mapa sumamos 1
                lista[simbolo]++
        else                    # generamos un nodo nuevo con ocurrencias =1
                lista[simbolo] = 1
for simbolo ∈ signal:           # calculamos la probabilidad como ocurrencias/cantidad
        signal_prob[simbolo] = signal[simbolo]/cantidad
```

Este pseudo-código genera los códigos mediante el método de Huffman. Debemos construir un árbol colocando como hojas todos los símbolos de la fuente junto con sus probabilidades, luego crear un nodo que sea padre de los dos nodos con probabilidades más bajas, que tendrá como probabilidad propia la suma de las probabilidades de sus hijos. Se repite este último paso siempre tomando en cuenta todos los nodos que aún no tengan padre. Además, cuando se crea un nodo padre, se le asocia a su hijo derecho un 1 y a su hijo izquierdo un 0, para luego armar el código.

```
def generarHuffman(listaValores):
    lista=[]
    for val ∈ listaValores:  # generamos la lista de terminales
       lista.append(Nodo(val[2],val[0],None,None))
       #val[2] = probabilidad del símbolo
       #val[0] = símbolo
    while len(lista)>1:
       lista.sort(key=lambda x:x.prob) # esto ordena de menor a mayor
       nodo1=lista.pop(0)
       nodo2=lista.pop(0)
       # tomamos los 2 primeros que serían los de probabilidad más baja
       lista.append(Nodo(nodo1.prob+nodo2.prob,None,nodo2,nodo1))
```

Para extraer la codificación debemos recorrer todo el árbol desde la raíz hasta cada hoja (nodo del símbolo), guardando los valores binarios de cada nodo en el camino. La secuencia binaria almacenada es el código del símbolo que está almacenado en dicha hoja.

```
def generarCodigos(nodo,codigo):
    if nodo != None:
       if nodo.valor != None:
          self.codigos[nodo.valor] = codigo
          # agregamos los codigos a un diccionario una vez que llegamos a un terminal
       self.generarCodigos(nodo.nodoDer,codigo+"1")
       self.generarCodigos(nodo.nodoIzq,codigo+"0")
```

Pseudo del código implementado para calcular el desvío en el muestreo computacional.
Utilizamos un motor montecarlo para calcular por muestreo computacional la media y el desvío de las señales en base a la distribución de las mismas (que calculamos previamente). Consideramos un epsilon de *1/1000 y una cantidad mínima de iteraciones de 1000 para evitar una convergencia temprana.*

```
def getDesvio(acumulada):
    suma = 0
    muestras = 0
    desvioAct = 0
    desvioAnt = 1
    sumaCuadrado = 0
    while not converge(desvioAct,desvioAnt) or muestras<1000:
        val=rand(acumulada)
        suma=suma+val
        muestras+=1
        mediaAct = suma/muestras
        desvioAnt = desvioAct
        sumaCuadrado = sumaCuadrado+(val-mediaAct)**2
        desvioAct = math.sqrt(sumaCuadrado/muestras)
    return desvioAct

def getMedia(acumulada):
    suma = 0
    muestras = 0
    mediaAct = 0
    mediaAnt = 1
    while not converge(mediaAct,mediaAnt) or muestras<1000:
        val=rand(acumulada)
        suma=suma+val
        muestras+=1
        mediaAnt = mediaAct
        mediaAct = suma/muestras
    return mediaAct
```

Para calcular la entropía basta con recorrer la distribución de probabilidades, obtenidas anteriormente, sumando cada probabilidad multiplicada con su logaritmo en base 2. Para obtener la longitud media debemos recorrer la estructura que almacena nuestro código y sumar las longitudes de cada codificación, y luego dividirlo por la cantidad total de símbolos.

```
def entropia(signal):
    ent = 0
    for simbolo ∈ signal:
        ent = ent+math.log2(signal[simbolo])*signal[simbolo]
    return -ent
```

```
def longitudPromedio():
    prom = 0
    for simbolo ∈ codigos:
        prom = prom+codigos[simbolo]*len(codigos[simbolo])
    return prom
```

Resultados

La implementación del método de Huffman nos permitió obtener la distribución de probabilidades de los valores de cada señal y su respectiva codificación. Los resultados revelaron los siguientes aspectos:

- La **entropía de cada señal** fue calculada, proporcionando una medida de la cantidad de información promedio contenida en la señal. Este valor nos ayuda a comprender la complejidad y la cantidad mínima de bits requeridos para su codificación eficiente.
    - En la señal 1 la entropía dio 9.58
    - En la senala 2 la entropía dio 6.67

- La **longitud media de la codificación de Huffman** de cada señal se determinó, lo que nos indica la eficiencia promedio de la codificación obtenida. Cuanto menor sea esta longitud, mayor será la compresión lograda.
    - En la señal 1 la longitud media de la codificación de huffman dio 9.59
    - En la señal 2 la longitud media de la codificación de huffman dio 6.7

- Se evaluó la **longitud total en bits** de cada señal codificada con Huffman y se comparó con el tamaño del archivo original. Esto nos da una medida de la eficiencia de la compresión lograda, permitiéndonos evaluar el nivel de reducción.
- En la señal 1 la longitud total en bits nos dio 9593 bits (56.000 bits original)
    - Obtuvimos una mejora de 1-9593/56000 (83% porcentaje de compresión)
- En la señal 2 la longitud total en bits nos dio 6705 (48.000 bits original)
    - Obtuvimos una mejora de 1-6705 /48.000 (86% porcentaje de compresión)

- Se calculó el **rendimiento del código de Huffman obtenido**. Esta medida nos permite evaluar qué tan cerca estamos de la eficiencia teórica máxima en la representación de la información.
    - La señal 1 tiene una entropía de 9.587 y el Huffman construido tiene una longitud promedio de 9.592, valor muy cercano al de la entropía (rendimiento del código 99.9%)
    - La señal 2 tiene una entropía de 6.667 y el Huffman construido tiene una longitud promedio de 6.704, valor muy cercano al de la entropía (rendimiento del código 99.4%)

```
PS C:\Users\santy\OneDrive\Documentos\TeoriaTPE> & C:/Users/santy/AppD
ata/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/santy/OneDrive
/Documentos/TeoriaTPE/Huffman.py
Nombre archivo: signal1
Longitud promedio Huffman: 9.59299999999987
Entropia de la señal: 9.587063625675983
Longitud del archivo codificado en bits: 9593
Tamaño del archivo codificado en kb: 1.199125
se creo el archivo output.txt
```

```
PS C:\Users\santy\OneDrive\Documentos\TeoriaTPE> & C:/Users/santy/AppD
ata/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/santy/OneDrive
/Documentos/TeoriaTPE/Huffman.py
Nombre archivo: signal2
Longitud promedio Huffman: 6.704999999999993
Entropia de la señal: 6.672471717028501
Longitud del archivo codificado en bits: 6705
Tamaño del archivo codificado en kb: 0.838125
se creo el archivo output.txt
```

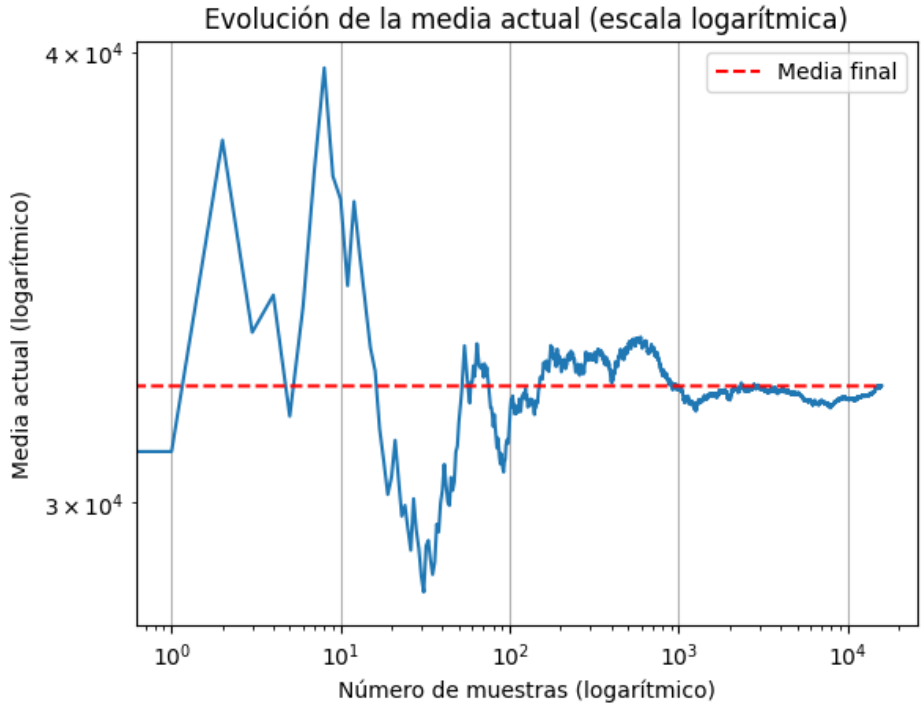
Una forma de **mejorar el rendimiento** de una fuente de información es **ampliando su orden a "n"**, lo que implica generar **n-tuplas de símbolos y calcular su probabilidad**. Según el teorema de Shannon, al hacer tender n hacia infinito, se puede alcanzar una mejora en la longitud media por símbolo, acercándose al valor de la entropía.

Además, utilizando la distribución de probabilidades obtenida para cada señal, realizamos una simulación computacional para obtener el valor de la media y el desvío en cada caso, y los comparamos. Esto nos brinda información sobre las características estadísticas de las señales y nos permite realizar análisis más detallados.

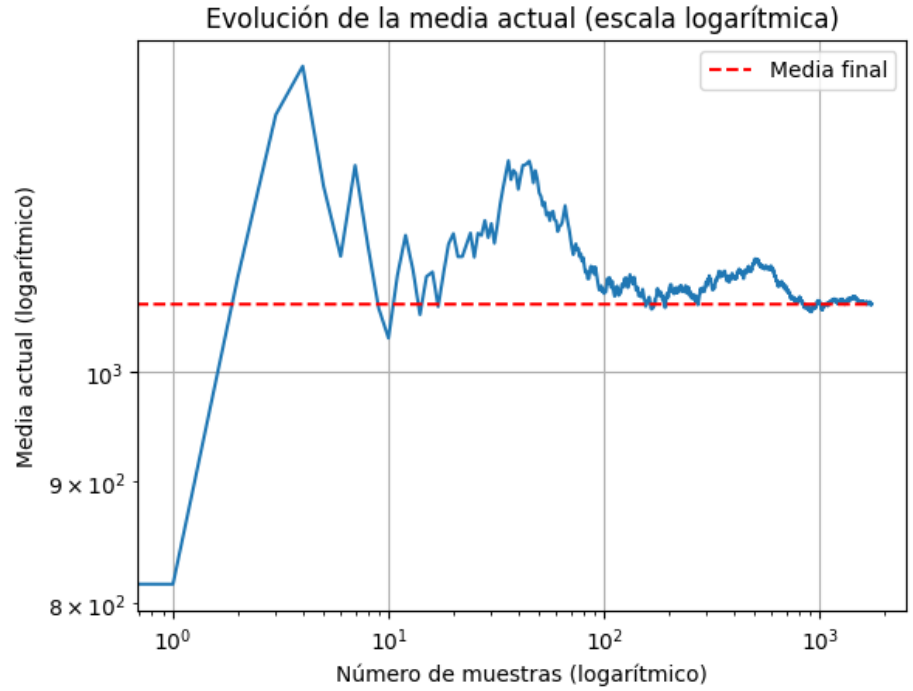
```
PS C:\Users\santy\OneDrive\Documentos\TeoriaTPE> & C:/Users/santy/AppD
ata/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/santy/OneDrive
/Documentos/TeoriaTPE/MuestreoComputacional.py
Nombre archivo: signal1
media:32732.698749320283
desvio:15628.221198830926
```

```
PS C:\Users\santy\OneDrive\Documentos\TeoriaTPE> & C:/Users/santy/AppD
ata/Local/Microsoft/WindowsApps/python3.10.exe c:/Users/santy/OneDrive
/Documentos/TeoriaTPE/MuestreoComputacional.py
Nombre archivo: signal2
media:1069.2395104895104
desvio:550.3674973029491
```

Señal 1



Señal 2



Conclusiones

En este informe, se presentó una implementación computacional del método de Huffman para el análisis de señales luminosas adquiridas por sensores en telescopios. Los resultados obtenidos demostraron la utilidad de este enfoque en la codificación eficiente de los valores de las señales, de la método de Huffman proporcionan herramientas poderosas para el estudio y análisis de señales.