TRABAJO PRÁCTICO ESPECIAL

Teoría de la información - 2021

Integrantes:

[249843 - Coria, Santiago]

santiagocoria@live.com.ar

[248530 - Santos, Matias Andrés]

matias_a_santos@hotmail.com

[248506 - Catauro Nardella, Eliel Arturo]

eliel.catauro@gmail.com

Indice

Introducción

En este informe se desarrolla como trabajar con señales y a partir de ellas poder crear fuentes que nos brindan una comprensión de ellas más sencilla. Además de esto se hará un análisis de los valores de la fuente a partir de indicadores calculados de estos valores.

Luego por otra parte se desarrollarán métodos de compresión de datos para así reducir el espacio que se necesita para representar la emisión dada.

Por último se trabajará sobre cómo implementar un canal a partir de las señales dadas y el análisis del mismo.

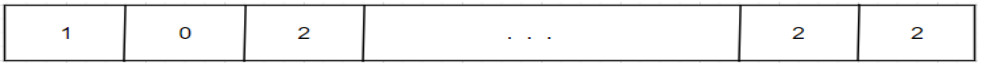
Desarrollo y Análisis.

Fuentes de información y muestreo computacional.

A

Para resolver este problema, se implementó una solución la cual consta de un previo procesamiento de los datos, ya que estos mismos como estan dados al querer hacer una matriz de pasaje sería muy grande y no tendría mucha lógica ya que por ejemplo si emite un valor de 370 el siguiente símbolo emitido no va a ser de 1980, ya que el crecimiento de la criptomoneda se podría decir que es “lineal”.

Entonces basados en esto se decide procesar los datos donde la emisión del símbolo 0 representa el estado que la moneda bajo, 1 que la moneda se mantuvo y 2 que la moneda subió.



(Imagen 1: Arreglo representativo de cómo serían las emisiones)

Una vez procesado los datos se implementó una solución basada en el siguiente pseudocódigo:

```
//Recorro el arreglo de emisiones
    //Sumo en la matriz de qué símbolo a que símbolo transiciona
//Recorro las columnas de las matriz para saber cuantas transiciones se hicieron desde ese símbolo
//Divido a esa columna por la cantidad de sus transiciones
//Retorno la matriz
```

(Pseudocodigo 1: Calcular la matriz de pasaje)

Una vez implementado el algoritmo podremos obtener los siguientes resultados:

Matriz pasaje ETH			
	Bajo	Mantuvo	Subió
Bajo	0.337	0.213	0.373
Mantuvo	0.187	0.480	0.260
Subió	0.474	0.305	0.365

Matriz pasaje BTC			
	Bajo	Mantuvo	Subió
Bajo	0.407	0.428	0.461
Mantuvo	0.011	0.095	0.025
Subió	0.581	0.476	0.512

A partir de estos resultados podremos obtener información sobre cómo se comportan nuestras emisiones ya que por ejemplo podremos saber que cuando el bitcoin baja de precio es altamente probable que el precio siga bajando o pase todo lo contrario y suba, además que es poco probable que una vez que bajo se mantenga el precio ya que dicha probabilidad es de 0.011.

B

En este problema se trabaja específicamente con los valores de cotización dados, y para resolverlo se recorre los valores de cada crypto sumando la multiplicación de la crypto en un tiempo actual por la crypto un tiempo actual más tau.

Una vez calculado esto al finalizar se va a dividir por la cantidad de emisiones menos tau, luego se actualiza el tau y se vuelve a calcular la autocorrelación para el nuevo valor de tau. Para implementar la solución se basó en este pseudocodigo:

```
//Mientras que aux_tau < tau
    //recorro i = 0 hasta valores_crypto.length
        //controlo que valores[i+aux_tau] no se pase de rango
            calculo_autocorrelacion = valores[i] * valores[i+aux_tau]
    calculo_autocorrelacion = calculo_autocorrelacion / (valores.length - aux_tau)
    //Guardo el valor calculado en un vector autocorrelacion
//Retorno el vector
```

(Pseudocodigo 2: Cálculo de la autocorrelación)

Cuando se corre el algoritmo con los datos provistos arroja los siguientes resultados:

Autocorrelación BTC		Autocorrelación ETH	
tau=1	valor = 5.216E7	tau=1	valor = 1443499.89
tau=2	valor = 5.444E7	tau=2	valor = 1442684.86
tau=3	valor =8.259E7	tau=3	valor = 1441829.58
...	...	...	...
tau=48	valor = 9.303E7	tau=48	valor = 1420067.96
tau=49	valor = 9.562E7	tau=49	valor = 1419661.93
tau=50	valor = 9.834E7	tau=50	valor = 1419286.52

Se puede observar que la moneda BTC y ETH su autocorrelación para los diferentes valores de tau siempre van a ser proporcionales.

C

Para realizar el cálculo de la correlación cruzada se plantea recorrer los arreglos que contienen los valores de las cotización y ir llevando en una variable el valor de multiplicar la cotización de una crypto en un tiempo t y de la otra crypto en un tiempo t más el incremento, además vamos a tener dos variables que van a ir llevando el cuadrado de cada uno de los valores a multiplicar. Por último lo que se hace es hacer la división y retornar el valor.

Los valores que retorna el algoritmo después de procesarlo son los siguientes:

Correlación cruzada BTC		Correlación cruzada ETH	
Incremento = 0	valor = 0.993	Incremento = 0	valor = 0.993
Incremento = 50	valor = 0.984	Incremento = 50	valor = 0.986
Incremento = 100	valor = 0.980	Incremento = 100	valor = 0.980
Incremento = 150	valor = 0.982	Incremento = 150	valor = 0.979
Incremento = 200	valor = 0.980	Incremento = 200	valor = 0.976

Estos valores dan todos cercanos a 1 lo que quiere decir es que están altamente correlacionados entre sí y esto tiene sentido ya que la cotización de la moneda tiende a mantenerse en valores cercanos al anterior, salvo en casos excepcionales donde cierta moneda tiene un pico, pero no son sucesos de todos los días.

D

El análisis de los resultados fue provisto en cada inciso explicado previamente.

Codificación y compresión.

A

Para calcular la distribución de probabilidades de cotización de cada moneda se planteó resolverlo a partir del siguiente pseudocódigo:

```
//Recorro el arreglo de valores hasta obtener todos los simbolos y su cantidad de apariciones
    //El simbolo no pertenece a mi lista
        //creo un par (simbolo emitido, 1)
        //Lo añado a lista
    //El simbolo pertenece a mi lista
        //traigo el par y le actualizo la cantidad de apariciones + 1

//Recorro mi nueva lista hasta no tener mas elementos
    //Divido cada elemento por la cantidad de emisiones
```

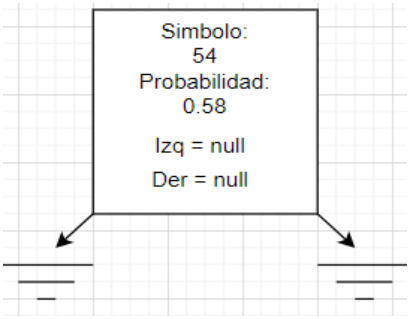
(Pseudocódigo 3: Cálculo de la distribución de probabilidades)

Cabe destacar que para la resolución de este ejercicio se plantea usar una Clase Par para almacenar el símbolo y la distribución de probabilidades de él y con estos cálculos que se realizaron se van a guardar así se podrán utilizar en el inciso B para calcular el código de Huffman Semi-Estático.

B

La solución de este ejercicio consta de varios métodos los cuales van a trabajar en conjunto para poder codificar el mensaje.

Primero se planteó en la estructura que va a contener al árbol generado entonces se decidió crear una clase llamada HuffmanNodo la cual va a guardar la información de la siguiente manera:



(Imagen 2: Estructura de la clase HuffmanNodo)

Una vez implementada esta estructura, procederemos a generar el árbol en el cual para hacerlo nos basamos en el siguiente pseudocódigo:

```
colaPrioridad<HuffmanNodo> q //cola de prioridades la cual va a tener todos los nodos con su probabilidad en orden ascendente

while (q.size() > 1) // recorro hasta que solo quede la raiz
    HuffmanNodo nodo1 = //Agarro el primer nodo de menor probabilidad y lo saco
    HuffmanNodo nodo2 = //Agarro el nuevo primer nodo de menor probabilidad y lo saco
    HuffmanNodo nuevo_nodo = //sumo las probabilidades de nodo1 y nodo2, el simbolo sera -1 ya que es un nodo intermedio
    //Actualizo la raiz y agrego el nuvo nodo a la cola

//retorno la raiz
```

(Pseudocódigo 4: Generar árbol de Huffman)

Ahora que ya se obtuvo el árbol, se procederá a recorrerlo de manera recursiva para obtener el código de cada símbolo, para hacer esto se siguió el siguiente pseudocódigo:

```
void conseguirCodigo(arbol , codigo){
    if(arbol.simbolo > -1)
        //Significa que es hoja asi que busco que simbolo es y guardo el codigo generado
    else {
        if(arbol.izq != null)
            //llamo a conseguirCodigo con arbol.getIzq() y a codigo lo concateno con 0
        if(arbol.der != null)
            //llamo a conseguirCodigo con arbol.getDer() y a codigo lo concateno con 1
    }
}
```

(Pseudocódigo 5: generación del código binario para cada símbolo)

Cómo se tomó la decisión de poner -1 para nodos intermedios y al saber que la cotización de una moneda nunca puede ser negativa, sabemos que cuando el símbolo sea mayor a -1 se va a estar en presencia de una hoja entonces vamos a guardar el valor del código generado para ese símbolo.

Una vez generado el código de cada símbolo solo queda recorrer el arreglo de cotización de los valores e ir concatenando el código.

Ya calculado todo lo relacionado a cómo representar el símbolo se procede a recorrer las emisiones de la moneda e ir concatenando los códigos correspondientes a la representación de ese valor.

Por último para decodificar el mensaje encriptado lo que haremos es seguir el siguiente pseudocódigo.

```
String decodificarHuffmanCodigo(String mensaje_codificado){
    for(i = 0; i <mensaje_codificado.length ; i++ ){
        if(mensaje_codificado.charAt(i) == 0)
            //Me muevo a izquierda en el arbol
            codigo_aux += 0
        else
            //Me muevo a derecha en el arbol
            codigo_aux += 1
        if(arbol.esNull())
            mensaje_decodificado += simbolos_codificados.getSimbolo(codigo_aux)
            //Reinicio valores para calcular el proximo simbolo
    }
    return mensaje_decodificado;
}
```

(Pseudocódigo 6: Cálculo de la decodificación del mensaje)

C

Para realizar el Run Length Coding se implementa una clase Par para poder almacenar el símbolo que aparece y cuantas repeticiones del símbolo consecutivamente hay. El algoritmo que se implementó para realizar la solución está basado en el siguiente pseudocódigo:

```
RLC(){
    ArrayList<Par> rlc;
    while (i < emision.fin - 1){
        int j = i
        par aux = (emision[i],1)
        while (emision[i] == emision[i+1]) {
            aux.setSegundoDato(+1)
            i++
        }
        rlc.add(aux)
        i++
    }
    return rlc
}
```

(Pseudocódigo 7: Cálculo de la decodificación del mensaje)

Para codificar se va a recorrer el arreglo de emisiones y se irá contando las repeticiones que aparecen del símbolo, luego estos datos se guardaran en un par y se guardarán en la solución.

Por último el decodificador solo deberá de recorrer esta lista y ir concatenando la cantidad de símbolos que le corresponde, el algoritmo que se implementó para hacerlo se basó en este pseudocodigo:

```
String decodificarRLC(){
    for(Par p: codificacion_rlc)
        for(i = 0; i < p.getCantRepeticiones(); i++)
            mensaje_decodificado+= p.getSimbolo;
    return mensaje_decodificado;
}
```

(Pseudocódigo 8: Cálculo de la decodificación del mensaje)

D

Los resultados de la tasa de compresión aplicando los algoritmos fueron estos:

Tasa de compresión con Huffman		Tasa de compresión con RLC	
ETH	BTC	ETH	BTC
0.5254	0.5213	2.5301	2.5536

Como podemos observar, gracias a la alta cantidad de símbolos distintos en los archivos, la compresión de Huffman no resulta para nada eficiente, arrojando un tamaño de archivo 50% más largo que el original. Por otra parte, la tasa de compresión RLC, pese a tener una gran cantidad de símbolos, no se ve afectada por el mismo problema, lo cual permite reducir el tamaño de los archivos de manera considerable, ya que, como explicamos anteriormente, las monedas tienden a mantenerse en el mismo precio entre t y t+1.

E

Se podría realizar un RLC con pérdida el cual básicamente es darle una tasa de pérdida y ver si el siguiente valor de cotización se mueve entre la tasa de pérdida +/- el valor de cotización.

Si se aplicara este método en los datos a comprimir se podría llegar a tener una tasa de compresión mayor a la obtenida en el inciso anterior a coste de perder ciertos valores de la cotización, cabe que la variación de la tasa va a depender con el umbral de pérdida que tomemos.

```
RLC_conPerdida(umbral){
    ArrayList<Par> rlc;
    while (i < emision.fin()){
        int j = i;
        par aux = new par(emision[i],1);
        while(umbral-emision[i] < emision[j] < umbral+emision[i]){
            aux.setSegundoDato(+1);
            j++
        }
    i = j;
    rlc.add(aux);
    }
    return rlc;
}
```

(Pseudocódigo 9: Cálculo de la codificación del mensaje con pérdida)

F

Cuando se evalúa el rendimiento del código de Huffman dará un rendimiento muy malo como se puede observar en la tabla, ya que los símbolos emitidos al no repetirse tanto llevará a que todos tengan una probabilidad casi idéntica por lo que hará que la longitud de cada código sea excesivamente larga.

	ETH	BTC
Entropía	1.486	2.364
Longitud media	6.702	9.590
Rendimiento	0.22	0.24

Este rendimiento se podría mejorar mediante la técnica de interpolación polinómica.

Canales.

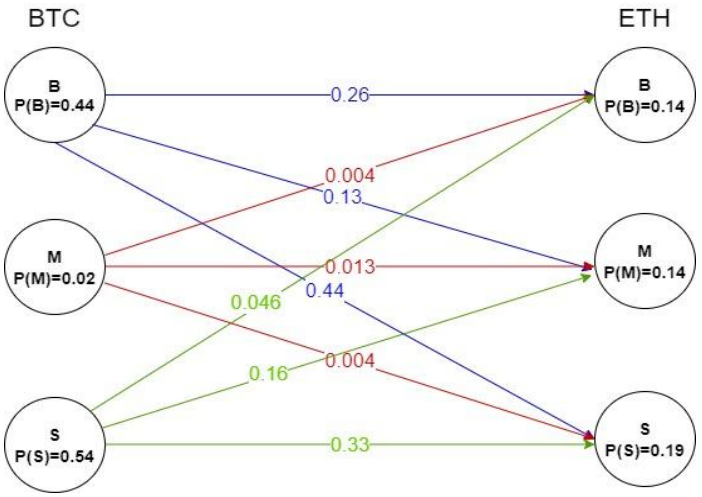
A

Para calcular el canal asociado se siguió el pseudocódigo:

```
void calculoProbabilidadTransicion(estado_btc,estado_eth){
    //Suma mas 1 a la probabilidad de que pase de un estado_btc a otro del eth
}

void calculoProbabilidadEmisionBTC(estado_btc){
    //Sumo 1 a la probabilidad que aparezca el estado pasado por parametro
}

void calculoProbabilidadEmisionETH(){
    //Calcula la probabilidad que se emita el simbolo eth
}

void calcularCanalAsociado(){
    for(int i = 1 ; i < emisiones.length; i++){
        estado_btc = Calculo el estado del btc si bajo, se mantuvo o subio con respecto al t anterior
        estado_eth = Calculo el estado del eth si bajo, se mantuvo o subio con respecto al t anterior
        calculoProbabilidadTransicion(estado_btc,estado_eth);
        calculoProbabilidadEmisionBTC(estado_btc);
    }
    calculoProbabilidadEmisionETH();
}
```

(Pseudocódigo 10: Cálculos del canal asociado)

Una vez aplicado el algoritmo nuestros resultados serán los siguientes para el canal:



(Imagen 3: Canal asociado calculado a partir del algoritmo)

B

El ruido y la pérdida del canal van a ser los siguientes:

	Bajó	Se mantuvo	Subió
Ruido	1.0871	0.1452	1.1583
Pérdida	0.6813	0.9837	0.4627

Para el ruido se puede observar que los valores de subió y bajo son mayores por lo que se agrega mayor información cuando se emita un símbolo lo que hará que sea más difícil distinguir el símbolo generado, mientras que cuando se emite el símbolo se mantuvo si bien cuenta con un poco de ruido este será más sencillo de distinguir.

Por otro lado la pérdida del canal será mayor en el símbolo se mantuvo, pero sin embargo también estará presente en los otros dos símbolos lo que generará incertidumbre al no saber de donde proviene el dato.

C

Como observamos en el inciso B, los valores de pérdida y ruido para ciertos símbolos no son lo suficientemente confiables para estimar el comportamiento de una moneda en base a la otra, por lo cual, el canal no nos resulta útil para estimar dicho comportamiento.

Conclusión

Gracias a este trabajo práctico logramos una mejor comprensión de cómo se analizan los datos desde un enfoque estadístico para fuentes y canales, pudiendo aplicar lo visto en la teoría.

Además al tener que aplicar algoritmos de compresión de datos se pudo entender mejor cómo funcionaban cada uno de ellos cerrando la idea de los algoritmos vistos en la teoría.

Con respecto a los resultados obtenidos podremos tener una estimación de que podría llegar a pasar a futuro con la criptomoneda pero no nos servirá para aplicarlo en la realidad ya que las monedas estudiadas en este trabajo práctico son conocidas por ser muy inestables.