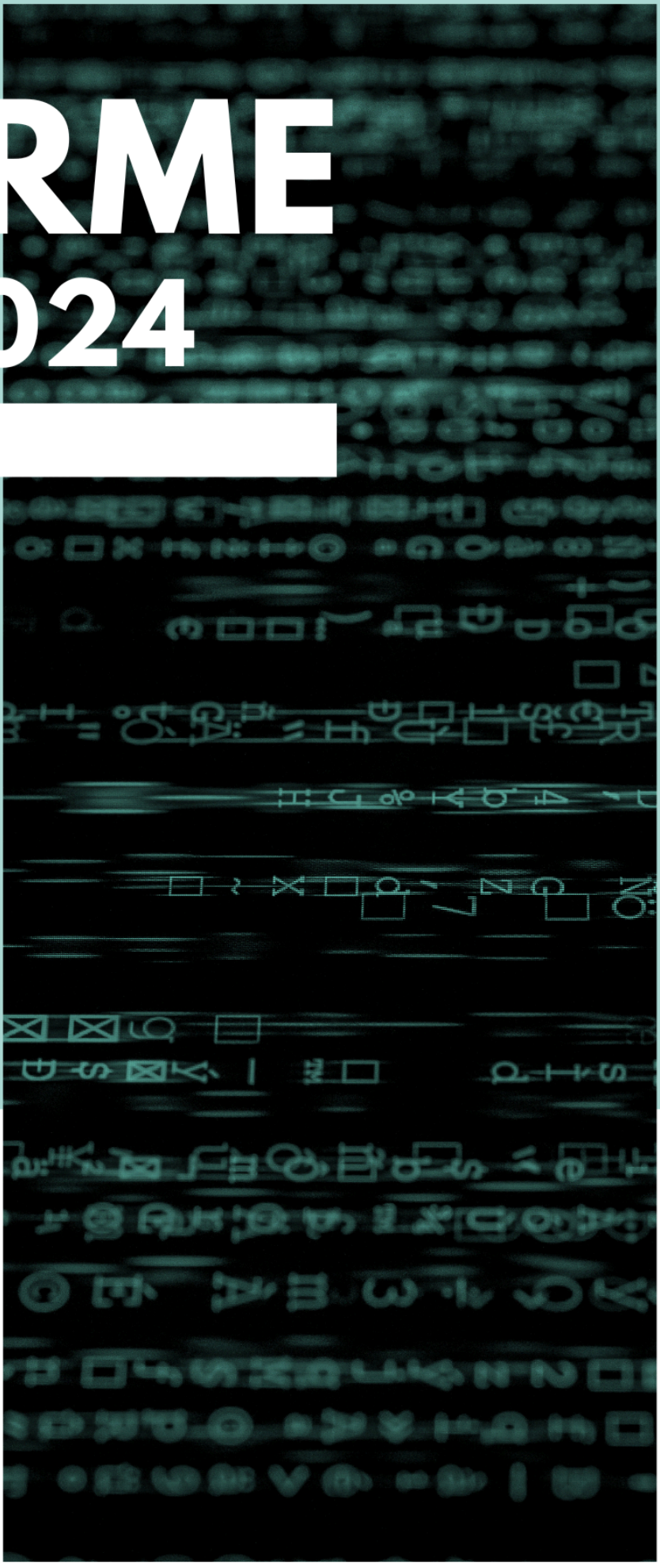
INFORME
T.P.E. 2024

ENTREGA: 21/06



GRUPO #10

FERRERI DELFINA
DELFIFERRERI23@GMAIL.COM

FERRERO ADRIEL RAM
ADRIELNECOCHEA@HOTMAIL.COM

GIRARDI EMILIANA
EGIRARDI@ALUMNOS.EXA.UNICEN.EDU.AR

TEORÍA DE LA INFORMACIÓN - UNCPBA FCEX

INTRODUCCIÓN

El presente trabajo aplica los contenidos impartidos en la cátedra de *Teoría de la Información 2024*. La consigna se basa en aplicar técnicas de análisis, codificación y muestreo computacional sobre tres datasets que contienen las temperaturas promedio diarias de las ciudades de Buenos Aires, Bogotá y Vancouver durante un mismo lapso de tiempo.

Las observaciones iniciales sobre los datasets indican que los tres tienen el mismo tamaño (6940 entradas, es decir, la temperatura diaria tomada durante 19 años) y se componen de secuencias de números enteros que representan la temperatura promedio asociada a ese día en °C. Las ciudades de las que provienen las muestras tienen la particularidad de encontrarse en latitudes muy distantes entre sí, con Buenos Aires en extremo sur, Bogotá cercano al Ecuador y Vancouver en el extremo norte. Creemos que esto será relevante para la interpretación de los resultados.

Los objetivos del trabajo apuntan a que podamos implementar en máquina las herramientas adquiridas en la materia con el fin de procesar, comprender y concluir de forma asertiva sobre datos reales en función del contexto dado. Adicionalmente nos propusimos hacer énfasis en seguir buenas prácticas del código para lograr eficiencia y claridad, minimizando la duplicación de código y el recorrido innecesario de estructuras, entre otras cosas.

Notas:

- Para resolver el enunciado optamos por escribir código Python en Google Colab.
- Los datasets, junto con este informe y una copia del entorno, fueron almacenados en un repositorio de Github cuyo link se adjunta a continuación: https://github.com/equisdel/weather-datasets.git.
- Hacemos referencia a las librerías: Request (para recuperar los datasets de Github y disponer los datos en listas de enteros), Numpy (para agilizar cálculos que involucran matrices), Math y Matplotlib (para creación de gráficos).
- Utilizamos listas para almacenar los datos de las fuentes.
- Utilizamos diccionarios para representar distribuciones de probabilidad. Por ejemplo para los símbolos S1, S2 y S3 una posible distribución válida sería {‘S1’: 0.2, ‘S2’: 0.75, ‘S3’: 0.05}. Este formato también es utilizado para expresar la codificación de Huffman. Para el ejemplo la codificación queda de de esta forma: {‘S1’: ‘01’, ‘S2’: ‘1’, ‘S3’: ‘00’}.

DESARROLLO Y ANÁLISIS

Medidas de tendencia central

Cálculo de la media aritmética.
Definición: *Valor promedio de temperatura que se esperaría observar en cada ciudad si se repitiera el experimento aleatorio una gran cantidad de días.*
Pseudocódigo:

```
getMedia(data):
   suma = 0
    for (i in data) suma += i
   return suma / len(data)
```

Cálculo del desvío estándar.
Definición: *Medida de dispersión que indica cuán dispersos están los valores de temperatura de cada ciudad respecto a la media.*
(!) Hace uso del cálculo de la media.
Pseudocódigo:

```
getDesvio(data):
   return (getMedia(data ^2) - getMedia() ^2) ^1/2
```

RESULTADOS	S1 - BUENOS AIRES	S2 - BOGOTÁ	S3 - VANCOUVER
Media aritmética	16.5	12.9	10.1
Desvío Estándar	6.0	1.1	5.6
Temp. máxima	31	19	28
Temp. mínima	1	8	-9

Interpretación de los resultados

- La ciudad donde más calor hace en promedio es Buenos Aires (16.5°C).
- La ciudad donde más frío hace en promedio es Vancouver (10°C).
- La ciudad de Bogotá presenta un promedio de temperatura diaria de 12.9°C.
- La ciudad con mayor desvío estándar es Buenos Aires, es decir, es la ciudad donde la temperatura suele variar más con respecto de la temperatura promedio. La ciudad de Vancouver posee un desvío similar.
- La ciudad con menor desvío estándar es Bogotá, donde las temperaturas son más estables o más fáciles de predecir.
- Sospechamos que la razón por la cuál los desvíos resultaron tan altos en Buenos Aires y en Vancouver mientras que en Bogotá obtuvimos un desvío bajo se relaciona con la ubicación de las ciudades con respecto a la línea del Ecuador. En las primeras dos, al encontrarse respectivamente en el extremo sur y en el extremo norte, las estaciones impactan con más fuerza sobre las temperaturas. En Bogotá, por otro lado, no se da esta situación dado que las variaciones en la cantidad de luz solar que recibe durante el año son mínimas.

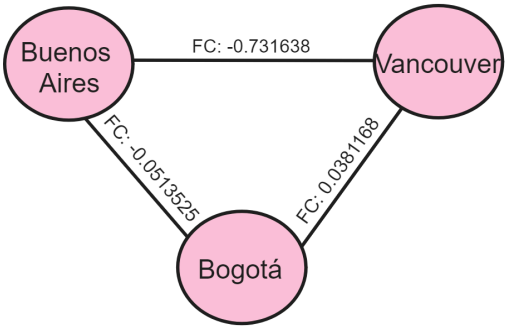
Factor de correlación cruzada

Definición: *Medida estadística utilizada para determinar la relación entre dos fuentes. La magnitud y el signo del resultado de la correlación cruzada indican la fuerza y la dirección de la relación entre ellas. Un valor alto (positivo o negativo) sugiere una fuerte relación, mientras que un valor cercano a cero indica poca o ninguna relación.*

Pseudocódigo:

```
correlacionCruzada(X, Y):
    suma = 0
    for i=0 to i=len(x)
      suma = suma + [(X[i] - <X>)(Y[i] - <Y>) / σ(X)σ(Y)]
    return suma/len(x)
```

RESULTADOS



	BsAs	Bog	Vanc
BsAs	1	-0.0513525	-0.731638
Bog	-0.0513525	1	0.0381168
Vanc	-0.731638	0.0381168	1

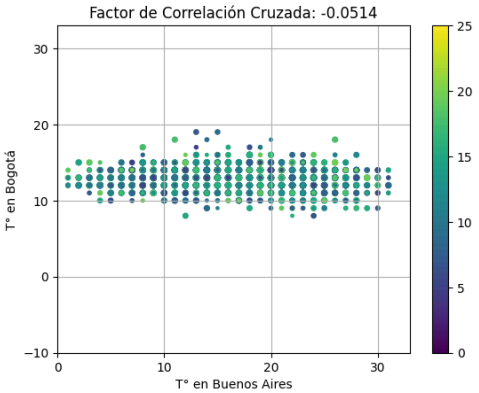
1. Buenos Aires - Bogotá:
Correlación Baja con signo (-)
El gráfico de dispersión se asemeja a una línea horizontal.

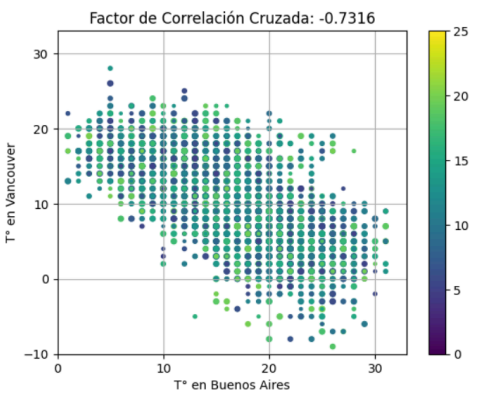
2. Buenos Aires - Vancouver:
Correlación Moderada-Alta con signo (-)
El gráfico de dispersión se asemeja a una línea diagonal hacia abajo.

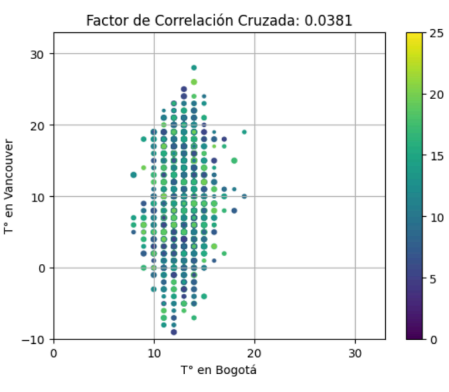
3. Bogotá - Vancouver:
Correlación Baja con signo (+)
El gráfico de dispersión se asemeja a una línea vertical.

1, 3: No hay una correlación significativa entre los datos.
2: Existe una correlación fuerte entre los datos. La tendencia es que los días en que la temperatura en Buenos Aires es más elevada, la temperatura en Vancouver suele ser más baja y viceversa. Esto tiene sentido, ya que las estaciones se invierten en los hemisferios norte y sur.

Diagramas de Dispersión:







Modelado de las fuentes markovianas

Pseudocódigo: obtención de la matriz de transición

La matriz de transición de estados contiene las probabilidades de pasar desde cualquier símbolo i a cualquier otro símbolo j.

Para calcularla nuestro método:

1. Inicializa una matriz de 3x3 con todos ceros, asociando los símbolos de a pares. Para eso proponemos una función que mapea los símbolos presentes a índices de la matriz. Para Buenos Aires y Vancouver el índice resultante será { 'A': 0, 'M': 1 , 'B': 2 }, mientras que para Bogotá, por no tener presente altas temperaturas, el índice generado será { 'M': 0, 'B': 1 }.

2. Recorre la fuente categorizada actualizando en cada momento el conteo de las ocurrencias de cada símbolo j según cuál fue el símbolo anterior i.

3. Al final divide cada conteo por la sumatoria de su columna, que representa la cantidad de ocurrencias del símbolo i por sí solo.

Siguiendo estos pasos obtenemos como resultado la matriz de transición de estados para la fuente dada.

```
getMatrizDeTransicion(fuente):
  index = getIndex(fuente)  // {A: 0, M: 1, B: 2}
  matriz = matriz(3x3)
  for i in len(fuente)-1:
    matriz[index[fuente[i+1]], index[fuente[i]] += 1
  matriz = matriz / sumaColumnas
  return matriz
```

↙	BUENOS AIRES			↙	BOGOTÁ			↙	VANCOUVER		
	A	M	B		A	M	B		A	M	B
A	0.818371	0.123094	0.0	A	-	-	-	A	0.59585	0.02338	0.0
M	0.181214	0.79023	0.30947	M	-	0.99681	0.84	M	0.404145	0.90764	0.06740
B	0.0	0.086674	0.69052	B	-	0.00303	0.16	B	0.0	0.06896	0.93229

Probabilidades marginales

- Precisamos la distribución de probabilidades marginales para cuantificar la entropía de las fuentes.
- Volver a recorrer la fuente no es una buena opción (ineficiente).
- No contamos con la matriz conjunta. Contamos solo con la matriz condicional, que por sí sola no basta para inferir las probabilidades marginales.
- Decidimos calcular la distribución a partir de un conteo de símbolos que implementamos en el método de categorización. Luego el conteo de cada símbolo se divide por la cantidad de datos totales para obtener su probabilidad.

P: Cont/6940	**A**		**M**		**B**	
S1	Cont: 2406	P: **0.34668**	Cont: 3542	P: **0.510374**	Cont: 992	P: **0.142939**
S2	Cont: 0	P: **0.0**	Cont: 6915	P: **0.99639**	Cont: 25	P: **0.00360**
S3	Cont: 193	P: **0.02780**	Cont: 3335	P: **0.48054**	Cont: 3412	P: **0.491642**

Entropía de las fuentes

Entropía sin memoria

Definición: *Promedio de preguntas que deben hacerse como mínimo para identificar un símbolo de la fuente sin memoria.*

$$H = \ -\sum pi * log2(pi)$$

```
getEntropiaSinMem(distProb):
  suma = 0
  for i in distProb:
    pi = distProb[i]
    if p > 0:
      suma += pi * log2(pi)
  return suma*(-1)
```

Entropía con memoria

Definición: *Promedio de preguntas que deben hacerse como mínimo para identificar un símbolo de la fuente con memoria dado que conozco el previo.*

$$Hcond = \ -\sum pi * hi$$

hi: entropía de cada columna de matriz de transición

```
getEntropiaConMem(distProb, mTrans):
suma = 0
for i in distProb:
   distCond = getDistribucionCondicional(i, mTrans)
   suma += distProb[i] *getEntropiaSinMem(distCond)
return suma
```

RESULTADOS	S1 - Buenos Aires	S2 - Bogotá	S3 - Vancouver
Entropía sin memoria	1.4262557016176838	0.034427079101625305	1.1553851439352645
Entropía con memoria	0.8473691887617005	0.03215758002947199	0.45210269269218595

Interpretación de los resultados

La **entropía sin memoria** no considera la dependencia entre los símbolos consecutivos.

- Buenos Aires y Vancouver tienen valores de entropía sin memoria relativamente altos, lo que sugiere que la distribución de las temperaturas es bastante impredecible y por ende se requiere un mayor número de preguntas binarias para identificar cada símbolo.
- Bogotá tiene un valor de entropía sin memoria muy bajo, lo que indica que hay poca incertidumbre en predecir la temperatura de un día cualquiera. Esto es porque Bogotá no presenta temperaturas Altas y tiene una probabilidad mínima de presentar Bajas.
- En conclusión: Buenos Aires y Vancouver son más aleatorias comparadas con Bogotá, que tiene un patrón de temperaturas muy predecible.

La **entropía con memoria** considera la dependencia entre los símbolos consecutivos, por lo que debería haber una mejora en la predicción de un símbolo si se conoce el símbolo anterior.

- Como es de esperar, estos valores para las tres fuentes disminuyen. Esto indica que hay mayor predictibilidad cuando se consideran las secuencias.
- Buenos Aires y Vancouver reducen considerablemente sus entropías, aunque nuevamente muestran valores relativamente más altos en comparación con Bogotá.
- La disminución de la entropía con memoria en Bogotá es mínima, introducir memoria no genera una mejora significativa con respecto al valor obtenido a partir de las marginales.
- En conclusión: la consideración de la memoria reduce la entropía en todas las ciudades, destacando la importancia de las relaciones temporales en la predictibilidad de las temperaturas.

Codificación de Huffman

Definición: *La codificación de Huffman es un algoritmo de compresión sin pérdida que asigna códigos binarios más cortos a los caracteres más frecuentes y más largos a los menos frecuentes. Se considera la mejor porque minimiza la longitud promedio del código, logrando una compresión eficiente.*

Pseudocódigo:

- **p_dist:** diccionario donde la clave es el símbolo y el valor es su probabilidad.
- **p_queue:** lista que se arma con p_dist empleando un formato donde cada elemento tiene dos partes: **[0:*probabilidad*, 1:[*lista de símbolos*]]**. Al inicio la lista de cada elemento consta de un único símbolo. Para ejemplificar, la distribución {**'S1': 0.2**,**'S2': 0.75**,**'S3': 0.05**} se traduce a [**[0.2**,**['S1']]**,**[0.75**,**['S2']]**,**[0.05**,**['S3']]**]. La función que efectúa la conversión es p_distList().
- **code:** diccionario donde la clave es el símbolo y el valor es su codificación.

```
HuffmanCoding(p_dist):
    code = {k: '' for k in p_dist.keys()}
    p_queue = p_distList(p_dist)
    n = len(p_queue)            # cantidad de símbolos presentes en la distribución

    for i in range(n-1):        # n-1 iteraciones:
      # 1. Ordena la lista y selecciona los dos elementos menos probables de la misma
      p_queue.sort()                          # orden por probabilidad: menor a mayor.
      1st_least_likely = p_queue.pop()        # elemento de menor probabilidad
      2nd_least_likely = p_queue.pop()        # segundo elemento de menor probabilidad

      # 2. Actualiza la codificación para los símbolos de los dos elementos menos probables
      for i in 1st_least_likely[1]:
          # Inserta 0 al inicio del code de los símbolos involucrados
          code[i] = '0' + code[i]
      for i in 2nd_least_likely[1]:
          # Inserta 1 al inicio del code de los símbolos involucrados
          code[i] = '1' + code[i]

      # 3. Combina ambos elementos en uno y reinserta el nuevo elemento en la lista
      disjoint_prob = 1st_least_likely[0] + 2nd_least_likely[0]
      merged_symbols = 1st_least_likely[1] + 2nd_least_likely[1]
      p_queue.append([disjoint_prob, merged_symbols])

      return code
```

NOTAS

- El código funciona para cualquier distribución, por ende la extensión a orden 2 (o más) puede hacerse sin problemas con tan solo pasar la distribución previamente calculada.
- EXTENSIÓN A ORDEN 2: utilizamos un método que genera la distribución extendida uniendo cada par de símbolos i,j en un solo símbolo con probabilidad p[i]*p[j/i] (asumimos que es con memoria). Retornamos el resultado en un diccionario.

RESULTADOS

1er teorema de Shannon: $H_1 \leq l < H_1 + 1$

2do teorema de Shannon: $\frac{H_1}{n} + (1 - \frac{1}{n}) * H_{cond} < \frac{H_1}{n} + (1 - \frac{1}{n}) * H_{cond} + \frac{1}{n}$

S1	Distribución	Code	Longitud promedio	Shannon $H_1 = 1.42625;\ H_{cond} = 0.84736$
Orden 1 (n = 3)	'A': 0.34668 'M': 0.51037 'B': 0.14293	'A': 01 'M': 1 'B': 00	2 · 0.34668 + 1 · 0.51037 + 2 · 0.14293 **1.48963**	1er T. de Shannon $1.42625 \leq$ **1.48963** < 2.42625
Orden 2 (n = 6)	'AA': 0.28371 'AM': 0.06282 'MA': 0.06282 'MM': 0.40331 'MB': 0.04423 'BM': 0.04423 'BB': 0.09870	'AA': 10 'AM': 1101 'MA': 1100 'MM': 0 'MB': 11100 'BM': 11101 'BB': 1111	2 · 0.28371 + 4 · 0.06282 + 4 · 0.06282 + 1 · 0.40331 + 5 · 0.04423 + 5 · 0.04423 + 4 · 0.09870 **2.31052**	2do T. de Shannon $1.13681 \leq$ **1.449** < 1.63681

S2	Distribución	Codificación obtenida	Longitud promedio	Shannon $H_1 = 0.03442;$ $H_{cond} = 0.03215$
Orden 1 (n = 2)	'M': 0.99639 'B': 0.00360	'M': 1 'B': 0	**1.00000**	1er T. de Shannon $0.03442 \leq$ **1.0** < 1.03442
Orden 2 (n = 4)	'MM': 0.99322 'MB': 0.00302 'BM': 0.00302 'BB': 0.00057	'MM': 1 'MB': 011 'BM': 00 'BB': 010	**1.01009**	2do T. de Shannon $0.03329 \leq$ **0.50539** < 0.53329

S3	Distribución	Codificación obtenida	Longitud promedio	Shannon $H_1 = 1.15538;\ H_{cond} = 0.45210$
Orden 1 (n = 3)	'A': 0.02780 'M': 0.48054 'B': 0.49164	'A': 10 'M': 11 'B': 0	**1.50836**	1er T. de Shannon $1.15538 \leq$ **1.50836** < 2.15538
Orden 2 (n = 6)	'AA': 0.01657 'AM': 0.01123 'MA': 0.01123 'MM': 0.43616 'MB': 0.03314 'BM': 0.03314 'BB': 0.45835	'AA': 1000 'AM': 10011 'MA': 10010 'MM': 11 'MB': 1010 'BM': 11101 'BB': 1111	**1.77450**	2do T. de Shannon $0.80374 \leq$ **1.2048** < 1.30374

OBSERVACIONES SOBRE LOS RESULTADOS

- Se verifica que todas las codificaciones obtenidas son válidas: cumplen la regla del prefijo

y las longitudes son acordes a las probabilidades de cada símbolo.
- En todos los casos se verifica el 1er Teorema de Shannon, es decir, se confirma que el código que retorna Huffman es casi tan eficiente como es teóricamente posible.
- Es observable mediante los gráficos de gradiente que la extensión a orden 2, aunque sea sin memoria, siempre mejora la calidad de la codificación. Esto es porque en los tres casos la extensión disminuyó la longitud promedio por símbolo a un valor más cercano a la entropía (óptimo teórico).
- Las mejoras al extender las fuentes variaron para cada conjunto de datos:
  - Para S1 y S2 la mejora en la longitud promedio fue mínima por razones diferentes.
  - En el caso de S1 fue porque para orden 1 se tenía de por sí una codificación bastante buena (muy cercana a la cota inferior).
  - S2, por el contrario, presenta una codificación de orden 1 que podría considerarse mala (muy cercana a la cota superior) y difícil de mejorar debido a la naturaleza de los datos. Esto indica que para notar una mejora debería extenderse el orden a un número considerable.
  - En el caso de S3, la mejora al extender el orden de la fuente fue notable.

TASAS DE COMPRESIÓN

- Se consideran tres tamaños para cada fuente (en bits): Tamaño original - T_{ORIG} -, Tamaño del archivo categorizado comprimido con orden 1 - $T_{COMP.Or1}$ -, Tamaño del archivo categorizado comprimido con memoria y orden 2 - $T_{COMP.Or2}$ -.
- Para estimar los tamaños asumimos que se emplea ASCII (8 bits por caracter/entero).

	S1		**S2**		**S3**	
$T_{ORIG} / T_{COMP.Or1}$	$\frac{103104\ b}{10338\ b}$	9.97:1	$\frac{110840\ b}{6940\ b}$	15.97:1	$\frac{84632\ b}{10468\ b}$	8.08:1
$T_{ORIG} / T_{COMP.Or2}$	$\frac{103104\ b}{7996\ b}$	12.89:1	$\frac{110840\ b}{3502\ b}$	31.65:1	$\frac{84632\ b}{6117\ b}$	13.84:1

Análisis del canal de información

MATRICES DEL CANAL

- La matriz **conjunta** - o matriz del canal - se obtuvo de manera similar a la matriz de transición de estados (conteo de ocurrencias de a pares S1-S4 en un mismo tiempo t dividido la cantidad total de datos).
- La matriz **condicional** se obtuvo a partir de la conjunta (dividiendo el valor en cada celda por la sumatoria de su columna).

MATRIZ CONJUNTA P(X,Y)		**X:** ENTRADA S1			**MATRIZ CONDICIONAL P(Y/X)** ↙		**X:** ENTRADA S1		
		A	M	B			A	M	B
Y: SALIDA S4	A	0.293761	0.031407	0.0	**Y:** SALIDA S4	A	0.847464	0.061547	0.0
	M	0.052874	0.437256	0.019881		M	0.152535	0.856860	0.139112

	B	0.0	0.041636	0.123037		B	0.0	0.081592	0.860887

CÁLCULO DE RUIDO E INFORMACIÓN MÚTUA

RUIDO - Pseudocódigo:

```
getRuido(pX, mCondicional):
   # getRuidoCols obtiene la
   # entropía de cada columna de la
   # matriz condicional
   ruidos = getRuidoCols(mCondicional)
   suma = 0
   for i in S1:
      suma += pX[i] * ruidos[i]
   return suma
```

INFO. MÚTUA - Pseudocódigo:

```
getInformacionMutua(ruido, pY):
  h = getEntropia(pY)
  return h - ruido
```

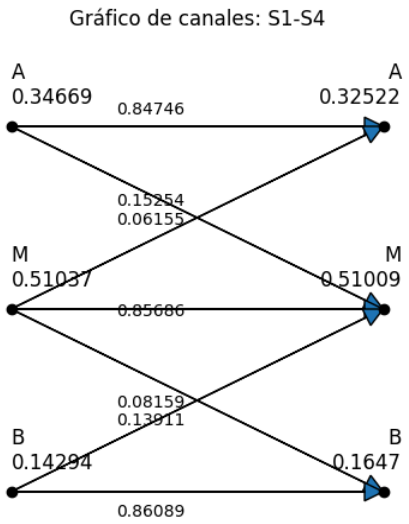
RESULTADOS

- **RUIDO:** **0.6711530178**
- **RUIDO MÍNIMO:** **0.0**
- **RUIDO MÁXIMO:** **1.5849625007**
- **INFO. MUTUA:** **0.7798190342**
- **INFO. MÍNIMA:** **0.0**
- **INFO. MÁXIMA:** **1.4262557016**

¿Cómo interpretamos los resultados?

- El **ruido** obtenido con respecto al máximo ruido posible (canal de la muerte, todos los vectores condicionales se setean en 1/3) indica que el canal posee un ruido considerable: un porcentaje importante de la entrada “se pierde” o no puede ser deducida a partir de la salida.

- Observamos a partir del valor obtenido de la **información mútua** que aproximadamente la mitad de la información se transmite correctamente.



- Para obtener el valor máximo de ruido y el valor mínimo de información mútua planteamos por separado al mejor y al peor canal posible para transmitir a S1. El peor canal posible es aquel que destruye toda la información ya que todos los símbolos de entrada se distribuyen con igual probabilidad entre todos los símbolos de salida. El mejor canal posible es aquel en el que no hay ningún tipo de ruido, por lo que el símbolo que entra siempre es igual al símbolo que sale.

Muestreo computacional del canal

Definición: *Implementando un motor de Montecarlo buscamos estimar la probabilidad de que entre dos apariciones consecutivas de un mismo símbolo j a la salida del canal, se obtengan hasta N símbolos distintos de j.*

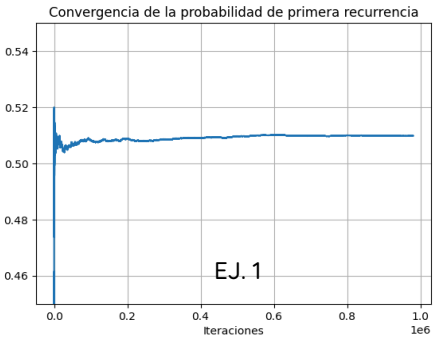
- Utilizamos las marginales de S1 para generar cada símbolo de entrada (sin memoria).
- Utilizamos la matriz condicional del canal para generar cada símbolo de salida en función del símbolo de entrada.

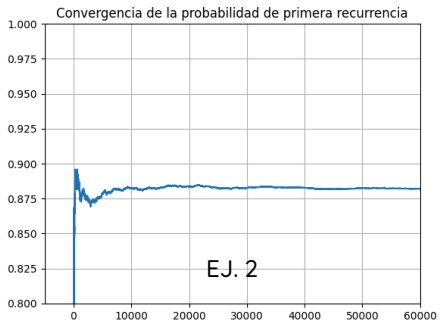
Pseudocódigo MONTECARLO:

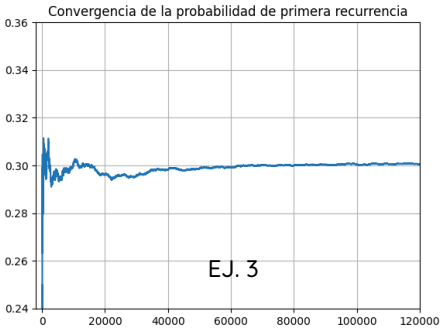
```
ProbNentreJ(J,N):
   pasosDados=0;
   while not converge() and (it < MIN_IT):
       e = sig_simb_entrada();     # Se genera un símbolo de entrada según dist_S1
       s = sig_simb_salida(e);     # Se genera un símbolo de salida según m. condicional
       if (s == j):                # Al darse una ocurrencia del símbolo j:
          if (pasosDados <= N):      # Se pregunta por la distancia a la última ocurrencia
             exitos++;               # EXITO: se actualiza la probabilidad
             prob = exitos / cantJ
             cantJ++
          pasosDados=0;           # Se actualiza la distancia a la última ocurrencia
       else: pasosDados++;        # Se actualiza la distancia a la última ocurrencia
       it++;
   return prob
```

RESULTADOS DE ALGUNAS PRUEBAS:

	PARÁMETROS DEL PROBLEMA		PARÁMETROS DE LA EJECUCIÓN		RESULTADOS
	J	N	Epsilon	MIN_IT	ProbNentreJ(J,N)
EJ. 1	'M'	1	5×10^{-7}	100	**0.509189**
EJ.2	'M'	3	5×10^{-7}	100	**0.881273**
EJ. 3	'B'	2	5×10^{-7}	100	**0.30267**







CONCLUSIÓN

El presente trabajo nos permitió comprender el papel crucial que las herramientas computacionales desempeñan en el campo de la Teoría de la Información, ya que nos permiten realizar un análisis más profundo de los datos, facilitando la visualización de patrones y tendencias que no nos serían evidentes a simple vista.
Mediante el uso de técnicas de análisis y representación de datos pudimos identificar comportamientos estacionales, comparar climas entre las ciudades y simular escenarios estocásticos. Comprender los datos en este contexto particular serviría para una variedad de aplicaciones prácticas, como la planificación urbana, la gestión de recursos energéticos y la toma de decisiones informadas en políticas ambientales.