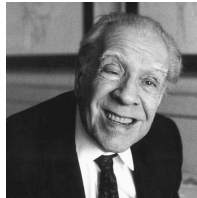


# Anexo 8

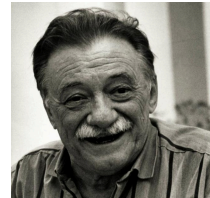
## Proyecto 8: Clasificador de texto

Mg. Luis Felipe Bustamante Narváez

En este ejercicio realizaremos un clasificador de texto basado en la forma escritural, sintáctica y semántica de dos escritores latinoamericanos, por un lado al argentino Jorge Luis Borges, y por otro frente al uruguayo Mario Benedetti.



Jorge Luis Borges



Mario Benedetti

### 1. Temas y Filosofía

**Borges:** Su poesía es filosófica, abstracta y llena de referencias literarias, mitológicas y metafísicas. Le interesaban temas como el tiempo, el infinito, el destino, la identidad y la memoria. Su tono es intelectual y a veces enigmático.

**Benedetti:** Escribe de manera más directa y accesible. Sus temas son el amor, la vida cotidiana, la lucha social, el exilio y la esperanza. Su tono es cálido, humano y cercano al lector.

### 2. Lenguaje y Estilo

**Borges:** Usa un lenguaje elegante, erudito y con muchas metáforas complejas. Su poesía es reflexiva, con estructuras clásicas y a veces con formas fijas como sonetos.

**Benedetti:** Usa un lenguaje sencillo, directo y coloquial. Sus poemas parecen conversaciones o pensamientos escritos sin mucha ornamentación.

### 3. Estructura y Ritmo

**Borges:** Tiende a usar estructuras más tradicionales con rima y métrica cuidadas, aunque también experimenta con versos libres.

**Benedetti:** Prefiere el verso libre y la naturalidad del habla cotidiana, sin preocuparse demasiado por la métrica.

## Librerías

```
In [... pip install tensorflow
In [... pip install colorama
In [... import numpy as np
import matplotlib.pyplot as plt
import string
from sklearn.model_selection import train_test_split
import os
from tqdm import tqdm
import PyPDF2
from IPython.display import display, HTML
from tensorflow.keras.preprocessing.sequence import pad_sequences
from colorama import Fore, Back, Style
```

## Cargamos los documentos

```
In [... def extraer_texto_desde_pdf(ruta_archivo):
with open(ruta_archivo, 'rb') as archivo:
lector = PyPDF2.PdfReader(archivo)
```

```

    texto = ''
    for pagina in range(len(lector.pages)):
        texto += lector.pages[pagina].extract_text()
    return texto

```

```
In [... ruta_carpeta = 'textos'
```

## Guardamos los textos en una lista

```

In [... todos_los_textos = []
for archivo in tqdm(os.listdir(ruta_carpeta),
                    bar_format=f'{Back.WHITE}{Fore.GREEN}{{1_bar}}{{bar}}{{Style.RESET_ALL}}'):
    if archivo.endswith('.pdf'):
        ruta_completa = os.path.join(ruta_carpeta, archivo)
        try:
            documento = extraer_texto_desde_pdf(ruta_completa)
            todos_los_textos.append(documento)
        except Exception as e:
            print(f'Error al procesar {archivo}: {e}')

```

```
100%|██████████
```

```
In [... todos_los_textos[0]
```

## Procesamiento de los datos

Vamos a separar los textos por etiquetas, enumerando los textos de Borgues con la etiqueta 0 y los de Benedetti con la etiqueta 1

```

In [... # Eliminamos espacios al inicio y al final para evitar problemas con el pdf
for texto in todos_los_textos:
    archivo = texto.strip()
    archivo

```

```

In [... # Creamos las listas vacías
textos = []
etiquetas = []

```

```

In [... # Mostramos línea por línea los textos de cada escritor
count = 0 #contador de línea para pruebas
for etiqueta, texto in enumerate(todos_los_textos):
    print(f'\n--- Texto {etiqueta} ---\n')
    for linea in texto.split(' \n'):
        count += 1 #contador de líneas para prueba
        print(linea)
        # Convertimos a minúsculas
        linea = linea.rstrip().lower()
        print(linea)
        # Eliminamos signos de puntuación
        if linea:
            linea = linea.translate(str.maketrans(' ', '', string.punctuation))
            print(linea)
            # Agregamos el texto limpio y le asignamos su respectiva etiqueta
            textos.append(linea)
            etiquetas.append(etiqueta)

```

```

In [... # Mostramos las listas
textos

```

## Entrenamiento

X representa la lista de los textos, y Y, representa la lista de las etiquetas, quien sería nuestra variable a predecir.

```
In [... X_train, X_test, Y_train, Y_test = train_test_split(textos, etiquetas, train_size=0.9, random_state=42)
```

```

In [... # Mostramos en forma de tupla el tamaño de cada muestra
len(Y_train), len(Y_test)

```

```
Out[... (2222, 247)
```

```
In [... # Probamos Las muestras de entrenamiento
X_train[0], Y_train[0]
```

```
Out[...] ('o acaso no la mira', 1)
```

```
In [... # Probamos Las muestras de prueba
X_test[0], Y_test[0]
```

```
Out[...] ('a inventar la verdad', 1)
```

## Representación de palabras desconocidas

### <unk>

Es una convención utilizada a menudo en **NPL** para representar palabras desconocidas o fuera del vocabulario. Por ejemplo, *si una palabra no se encontró en la muestra de entrenamiento, pero aparece en la muestra de prueba, será desconocida, y se requiere agregarle un índice que diferencie a esta palabra.*

```
In [... indice = 1
indice_palabras = {'<unk>': 0}
```

## Construcción del diccionario de codificación de palabras a índice

```
In [... for texto in X_train:
    # Separamos cada palabra
    tokens = texto.split()
    #print(tokens) # Probamos como se ven Los tokens
    for token in tokens:
        # Buscamos si La palabra no está en el índice para luego agregarla sin repetir
        if token not in indice_palabras:
            indice_palabras[token] = indice
            indice += 1
```

```
In [... # Mostramos el índice de palabras - palabras únicas
indice_palabras
```

```
In [... # tamaño de palabras únicas
indice_palabras
```

## Conversión del índice de palabras de String a enteros

Cómo el entrenamiento no se debe hacer con palabras, creamos una muestra convertida a su valor específico en enteros

```
In [... # Listas para enteros
X_train_int = []
X_test_int = []
# Banderas para ejecutarse una sola vez
X_int_train_hecho = False
X_int_test_hecho = False
```

```
In [... # Conversión de Los datos de entrenamiento
if not X_int_train_hecho:
    for texto in X_train:
        # dividimos de nuevo en palabras
        tokens = texto.split()
        # Por cada palabra encontrada la cambia por su valor numérico de la clave del diccionario
        linea_entero = [ indice_palabras[token] for token in tokens ]
        #print(linea_entero)
        # Agregamos el nuevo valor a la lista de entrenamiento
        X_train_int.append(linea_entero)
    X_int_train_hecho = True
    print("Conversión de entrenamiento ejecutada con éxito.")
else:
    print("La conversión de entrenamiento ya se había ejecutado previamente.")
```

Conversión de entrenamiento ejecutada con éxito.

```
In [... # Mostramos la conversión de entrenamiento
X_train_int
```

```
In [... # Conversión de Los datos de prueba -- Como puede haber desconocidos, debemos hacer esto:
if not X_int_test_hecho:
    for texto in X_test:
        tokens = texto.split()
        linea_entero = [indice_palabras.get(token, 0) for token in tokens] #trae el token o 0
        #print(linea_entero)
        X_test_int.append(linea_entero)
    X_int_test_hecho = True
    print("Conversión de prueba ejecutada con éxito.")
else:
    print("La conversión de prueba ya se había ejecutado previamente.")
```

Conversión de prueba ejecutada con éxito.

```
In [... # Mostramos la conversión de prueba
len(X_test_int)
```

```
Out[...] 247
```

## Matriz de Transición

Como se indicó en la teoría de los procesos de **Markov**, se requiere construir una matriz de transición y los estados iniciales para cada escritor:

1. Creamos un vector **V** con el tamaño total del **indice\_palabras**
2. Creamos la matriz **A0** para las palabras de **Borges**
3. Creamos el vector de probabilidad inicial **pi0** para las palabras de **Borges**
4. Creamos la matriz **A1** para las palabras de **Benedetti**
5. Creamos el vector de probabilidad inicial **pi1** para las palabras de **Benedetti**

```
In [... V = len(indice_palabras)
# Creamos Las matrices y vectores con 1 para poder hacer el suavizado
A0 = np.ones((V, V))
pi0 = np.ones(V)
A1 = np.ones((V, V))
pi1 = np.ones(V)
#Mostramos, por ejemplo
pi0
```

```
Out[...] array([1., 1., 1., ..., 1., 1., 1.])
```

## Función de conteo de palabras

```
In [... def compute_counts(texto_as_int, A, pi):
    #Recorremos Los tokens
    for tokens in texto_as_int:
        #Creamos el posible último elemento como referencia
        last_index = None
        #Recorremos cada elemento de cada línea
        for index in tokens:
            #Nos ubicamos en la primera secuencia
            if last_index is None:
                # Agregamos el valor inicial
                pi[index] +=1
            else:
                # Agregamos Los valores a La matriz
                A[last_index, index] += 1
            # Asignamos el valor actual al last_index
            last_index = index
```

```
In [... # Llamamos la función
#Para Borges
compute_counts([t for t, y in zip(X_train_int, Y_train) if y == 0], A0, pi0)
#Para Benedetti
compute_counts([t for t, y in zip(X_train_int, Y_train) if y == 1], A1, pi1)
```

```
In [... # Probamos
A1

Out[...] array([[1., 1., 1., ..., 1., 1., 1.],
               [1., 1., 2., ..., 1., 1., 1.],
               [1., 1., 1., ..., 1., 1., 1.],
               ...,
               [1., 1., 1., ..., 1., 1., 1.],
               [1., 1., 1., ..., 1., 1., 1.],
               [1., 1., 1., ..., 1., 1., 1.]])
```

## Explicación

**pi0** = array([ 1., 10., 1., ..., 1., 1., 1.])

**pi1** = array([ 1., 14., 1., ..., 1., 1., 1.])

**A0** = array([[1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], ..., [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.]])

**A1** = array([[1., 1., 1., ..., 1., 1., 1.], [1., 1., 2., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], ..., [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.], [1., 1., 1., ..., 1., 1., 1.]])

En el vector inicial **pi0**, la primera posición corresponde a los unk, la segunda posición corresponde a la palabra **o**, y nos está indicando que en los textos de Borges aparece iniciando la línea 10 veces. Si observamos los textos de Benedetti, **pi1**, nos indica que aparece 14 veces comenzando la línea. De esta manera podemos proceder a encontrar la probabilidad.

Con respecto a las matrices de transición, podemos observar en la matriz de Benedetti, **A1**, que en la segunda fila, hubo una transición de la palabra actual a la siguiente, valores que nos indicarán el comportamiento natural de los textos.

## Distribución de Probabilidad

### Normalizamos

Para observar las probabilidades, se requiere normalizar los vectores y matrices generados en el conteo, para que su valor oscile entre 0 y 1, como debe ser.

Esta es una manera empírica de demostrar las fórmulas mencionadas en la teoría

```
In [... #Conservamos Los datos originales A0, A1, pi0 y pi1, creando Las variables nomralizadas
#Para esto vamos a guardar Los datos originales
A0_norm = A0.copy()
pi0_norm = pi0.copy()
A1_norm = A1.copy()
pi1_norm = pi1.copy()
# Bandera de normalizado
normalize = False
```

```
In [... # Borges
if not normalize:
    # Borges
    A0_norm /= A0_norm.sum(axis=1, keepdims = True)
    pi0_norm /= pi0_norm.sum()
    # Benedetti
    A1_norm /= A1_norm.sum(axis=1, keepdims = True)
    pi1_norm /= pi1_norm.sum()
    print("Normalización ejecutada con éxito.")
    normalize = True
else:
    print("Las variables ya fueron normalizadas previamente.")
```

Normalización ejecutada con éxito.

```
In [... # Probamos
pi0_norm
```

```
Out[... array([0.00023111, 0.00231107, 0.00023111, ..., 0.00023111, 0.00023111,
0.00023111])
```

```
In [... A0_norm
```

```
Out[... array([[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472,
0.00029472],
[0.00029155, 0.00029155, 0.00029155, ..., 0.00029155, 0.00029155,
0.00029155],
[0.00029455, 0.00029455, 0.00029455, ..., 0.00029455, 0.00029455,
0.00029455],
...,
[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472,
0.00029472],
[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472,
0.00029472],
[0.00029464, 0.00029464, 0.00029464, ..., 0.00029464, 0.00029464,
0.00029464]])
```

## Explicación

```
pi0_norm = array([0.00023111, 0.00231107, 0.00023111, ..., 0.00023111, 0.00023111, 0.00023111])
```

```
A0_norm = array([[0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472, 0.00029472], [0.00029155, 0.00029155,
0.00029155, ..., 0.00029155, 0.00029155, 0.00029155], [0.00029455, 0.00029455, 0.00029455, ..., 0.00029455, 0.00029455,
0.00029455], ..., [0.00029472, 0.00029472, 0.00029472, ..., 0.00029472, 0.00029472, 0.00029472], [0.00029472, 0.00029472,
0.00029472, ..., 0.00029472, 0.00029472, 0.00029472], [0.00029464, 0.00029464, 0.00029464, ..., 0.00029464, 0.00029464,
0.00029464]])
```

En el vector inicial **pi0**, observemos que no aparece ningún valor en cero, lo que indica que el método de suavizar funcionó perfectamente, al igual que en la matriz **A0**, lo que permite una **distribución de probabilidad**.

## Espacio logarítmico

Como vimos en la teoría, estas probabilidades pueden tener un desbordamiento por debajo, ya que se aproximan a cero, entonces, para evitar errores computacionales, usaremos el espacio logarítmico.

```
In [... #Borges
log_A0_norm = np.log(A0_norm)
log_pi0_norm = np.log(pi0_norm)
#Benedetti
log_A1_norm = np.log(A1_norm)
log_pil_norm = np.log(pil_norm)
```

```
In [... # Probamos
log_pi0_norm
```

```
Out[... array([-8.37262974, -6.07004465, -8.37262974, ..., -8.37262974,
-8.37262974, -8.37262974])
```

```
In [... # Probamos
log_A0_norm
```

```
Out[... array([[ -8.12946976, -8.12946976, -8.12946976, ..., -8.12946976,
-8.12946976, -8.12946976],
[ -8.14031554, -8.14031554, -8.14031554, ..., -8.14031554,
-8.14031554, -8.14031554],
[ -8.13005904, -8.13005904, -8.13005904, ..., -8.13005904,
-8.13005904, -8.13005904],
...,
[ -8.12946976, -8.12946976, -8.12946976, ..., -8.12946976,
-8.12946976, -8.12946976],
[ -8.12946976, -8.12946976, -8.12946976, ..., -8.12946976,
-8.12946976, -8.12946976],
[ -8.12976445, -8.12976445, -8.12976445, ..., -8.12976445,
-8.12976445, -8.12976445]])
```

# Pre-análisis

Vamos a revisar diferentes elementos que nos permitan entender mejor lo que desarrollamos

```
In [... # Conteo de etiquetas de clase 0 (Borges) en Y_train
count_Y_0 = sum(y == 0 for y in Y_train)
# Conteo de etiquetas de clase 1 (Benedetti) en Y_train
count_Y_1 = sum(y == 1 for y in Y_train)
# Cantidad total de ejemplos de entrenamiento
total = len(Y_train)
# Probabilidad a priori de la clase 0
p0 = count_Y_0 / total
# Probabilidad a priori de la clase 1
p1 = count_Y_1 / total
# Logaritmo de la clase a priori 0
log_p0 = np.log(p0)
# Logaritmo de la clase a priori 1
log_p1 = np.log(p1)

display(HTML(f'''
Se encontró {count_Y_0} etiquetas de clase 0, <b style='color:fuchsia;'>Borges</b>,<br>
Se encontró {count_Y_1} etiquetas de clase 1, <b style='color:skyblue;'>Benedetti</b>,<br>
para un total de <b style='color:red;'>{total}</b> ejemplos de entrenamiento.
<hr>
Las probabilidades a priori serían las siguientes:<br>
<table style="border: 1px solid black; border-collapse: collapse;">
  <tr>
    <td style="border: 1px solid black; padding: 5px;">Borges</td>
    <td style="border: 1px solid black; padding: 5px;">{p0}</td>
  </tr>
  <tr>
    <td style="border: 1px solid black; padding: 5px;">Benedetti</td>
    <td style="border: 1px solid black; padding: 5px;">{p1}</td>
  </tr>
</table>
<hr>
Como usamos el espacio logarítmico, estas serían las probabilidades reales de encontrar un texto de la clase 0 o 1:
<table style="border: 1px solid black; border-collapse: collapse;">
  <tr>
    <td style="border: 1px solid black; padding: 5px;">Borges</td>
    <td style="border: 1px solid black; padding: 5px;">{log_p0}</td>
  </tr>
  <tr>
    <td style="border: 1px solid black; padding: 5px;">Benedetti</td>
    <td style="border: 1px solid black; padding: 5px;">{log_p1}</td>
  </tr>
</table>
<hr>
'''))
```

Se encontró 934 etiquetas de clase 0, **Borges**,  
Se encontró 1288 etiquetas de clase 1, **Benedetti**,  
para un total de **2222** ejemplos de entrenamiento.

Las probabilidades a priori serían las siguientes:

Borges	0.42034203420342037
Benedetti	0.5796579657965797

Como usamos el espacio logarítmico, estas serían las probabilidades reales de encontrar un texto de la clase 0 o 1:

Borges	-0.8666865319707326
Benedetti	-0.5453170635352763

# Construcción del Clasificador

```
In [... # Creamos una clase
class Classifier:
    # Constructor
    def __init__(self, log_As, log_pis, log_apriors):
        self.log_As = log_As
        self.log_pis = log_pis
        self.log_apriors = log_apriors
        # número de clases
        self.k = len(log_apriors)




    # Método de verosimilitud
    def _compute_log_likelihood(self, input_, class_):
        log_A = self.log_As[class_]
        log_pi = self.log_pis[class_]
        #Repetimos lo hecho en el ejemplos de creación de la matriz
        last_index = None
        log_prob = 0
        #Recorremos la entrada del usuario
        for index in input_:
            if last_index is None:
                #Primer token en la secuencia
                log_prob += log_pi[index]
            else:
                #Calculamos la probabilidad de transición del a palabra anterior a la actual
                log_prob += log_A[last_index, index]
                #Actualizamos el index para la próxima iteración
                last_index = index
        return log_prob

    # Función de predicción
    def predict(self, inputs):
        predictions = np.zeros(len(inputs))
        for i, input_ in enumerate(inputs):
            # Calcula los logaritmos de las probabilidades posteriores para cada clase
            posteriors = [self._compute_log_likelihood(input_, c) + self.log_apriors[c] \
                          for c in range(self.k)]
            #Elige la clase de mayor probabilidad posterior como la predicción
            pred = np.argmax(posteriors)
            predictions[i] = pred
        return predictions
```

## Explicación

### 1 Constructor (init) ¿Qué parámetros recibe?

log\_As: Matrices de probabilidades de transición entre palabras en logaritmo. log\_pis: Probabilidades iniciales de cada palabra en logaritmo. log\_apriors: Probabilidades previas (prior) de cada clase en logaritmo. self.k: Número total de clases.

 ¿Por qué usa logaritmos?  Evita problemas de underflow cuando se multiplican muchas probabilidades pequeñas.  Convierte productos en sumas, lo que hace más fácil la optimización.

### 2 Método \_compute\_log\_likelihood (Cálculo de Verosimilitud)

 ¿Qué hace?

Calcula la log-verosimilitud de una secuencia (input\_) dada una clase (class\_). Usa la probabilidad inicial de la primera palabra (log\_pi[index]). Luego, suma las probabilidades de transición entre palabras (log\_A[last\_index, index]). Retorna log\_prob, que indica qué tan probable es la secuencia dada la clase.

### 3 Método predict (Clasificación)

 ¿Qué hace?

Inicializa predictions con ceros (un array para almacenar las predicciones).

Para cada entrada en inputs: Calcula las log-verosimilitudes para cada clase. Suma la probabilidad previa (prior) log\_apriors[c] de cada clase. Elige la clase con mayor probabilidad posterior usando np.argmax(). Devuelve predictions, que contiene las clases predichas.



# Objeto de la clase Clasifier

```
In [...] # Creamos un objeto de la clase Clasifier para llamar los métodos del clasificador
clf = Classifier([log_A0_norm, log_A1_norm], [log_pi0_norm, log_pi1_norm], [log_p0, log_p1])
```

## Explicación

La clase Classifier, recibe 3 parámetros en su constructor, es decir 3 atributos. En su orden estos atributos son:

- 1 [log\_A0\_norm, log\_A1\_norm] que serán los argumentos de log\_As
- 2 [log\_pi0\_norm, log\_pi1\_norm] que serán los argumentos de log\_pis
- 3 [log\_p0, log\_p1] que serán los argumentos de log\_apriors

Es decir,

- 1 Las matrices de transición normalizadas
- 2 Los vectores con los valores iniciales o estados iniciales
- 3 Las probabilidades de cada clase utilizando el espacio logarítmico

## Predicción

```
In [...] # Llamamos al método predict (Datos de entrenamiento: Aprox 1.0)
P_train = clf.predict(X_train_int)
# Mostramos la predicción con la muestra de entrenamiento
print(f'Accuraci Train: {np.mean(P_train == Y_train)}')
```

Accuraci Train: 0.9932493249324933

```
In [...] len(X_test_int)
```

Out[...] 247

```
In [...] # Llamamos al método predict (Datos de prueba)
P_test = clf.predict(X_test_int)
# Mostramos la predicción con la muestra de prueba
print(f'Accuraci Test: {np.mean(P_test == Y_test)}')
```

Accuraci Test: 0.7530364372469636

## Probamos con textos nuevos

### Textos en los estilos de Borges y Benedetti

Le solicitamos a ChatGPT4 que nos generara textos con escritura similar a cada uno de nuestros poetas, y esto fue lo que nos entregó.

#### Estilo Borges

*En el vasto archivo del tiempo,  
donde las sombras del ayer se funden  
con las visiones inciertas del mañana,  
el hombre camina por un laberinto  
de palabras y azares.  
Sabe, aunque lo olvida a menudo,  
que su destino está tejido con hilos invisibles,  
urdidos por manos que jamás verá.*

*Alguna tarde,  
quizá en la penumbra de una biblioteca  
o en la geometría secreta de un sueño,  
descubrirá que su vida no ha sido sino el eco de otras vidas,*

*el reflejo de una historia que ya ha sido escrita  
en un idioma remoto y perfecto.*

---

## Estilo Benedetti

*La ciudad despierta con su ritmo de siempre,  
entre murmullos de bocacalles  
y pasos apurados que no saben bien a dónde van.*

*En un café cualquiera,  
un hombre revuelve su taza con la mirada perdida,  
tal vez recordando un amor que ya no está,  
tal vez soñando con la ternura que aún no llega.*

*Afuera, la vida sigue su curso,  
con su cuota justa de olvidos y de esperanzas,  
con sus rutinas que a veces duelen y a veces salvan.*

*Porque, al final de cuentas,  
lo importante no es cuánto nos golpea el tiempo,  
sino con quién elegimos compartirlo.*

## Funciones para poner a prueba el nuevo texto

```
In [... def preprocesar_texto(texto):
    texto = texto.lower().translate(str.maketrans('', '', string.punctuation))
    tokens = texto.split()
    return tokens

def convertir_a_indices(tokens, indice_palabras):
    return [indice_palabras.get(token, 0) for token in tokens] # 0 para <unk>

def calcular_probabilidad(tokens_indices, log_pi, log_A):
    if not tokens_indices:
        return -np.inf # Evitar errores con texto vacío
    log_prob = log_pi[tokens_indices[0]] # Probabilidad inicial
    for i in range(len(tokens_indices) - 1):
        log_prob += log_A[tokens_indices[i], tokens_indices[i + 1]] # Probabilidad de transición
    return log_prob

def clasificar_texto(texto, log_pi0_norm, log_A0_norm, log_pi1_norm, log_A1_norm, indice_palabras):
    tokens = preprocesar_texto(texto)
    tokens_indices = convertir_a_indices(tokens, indice_palabras)

    log_prob_borges = calcular_probabilidad(tokens_indices, log_pi0_norm, log_A0_norm)
    log_prob_benedetti = calcular_probabilidad(tokens_indices, log_pi1_norm, log_A1_norm)

    #Conversión de LOG-PROBABILIDADES a PROBABILIDADES
    max_log_prob = max(log_prob_borges, log_prob_benedetti) # Para evitar underflow numérico
    prob_borges = np.exp(log_prob_borges - max_log_prob)
    prob_benedetti = np.exp(log_prob_benedetti - max_log_prob)
    total_prob = prob_borges + prob_benedetti
    prob_borges /= total_prob # Normalizamos
    prob_benedetti /= total_prob

    print(f"Probabilidad Borges: {prob_borges:.4f}")
    print(f"Probabilidad Benedetti: {prob_benedetti:.4f}")

    return "Borges" if prob_borges > prob_benedetti else "Benedetti"
```

## Explicación

Este código implementa un clasificador de texto que determina si un texto nuevo se asemeja más a un escrito de Borges o de Benedetti. Lo hace utilizando un modelo basado en cadenas de Markov, donde se calculan probabilidades de transición entre palabras.

#### 1 preprocesar\_texto(texto)

Objetivo: Convierte el texto en minúsculas, elimina signos de puntuación y lo divide en palabras (tokens).

Proceso:

1. `texto.lower()` → Convierte todo el texto a minúsculas.
2. `.translate(str.maketrans("", "", string.punctuation))` → Elimina la puntuación.
3. `.split()` → Divide el texto en palabras, generando una lista de tokens.

#### 2 convertir\_a\_indices(tokens, indice\_palabras)

Objetivo: Convierte cada palabra (token) en un índice numérico basado en un diccionario `indice_palabras`.

Proceso:

1. Usa `indice_palabras.get(token, 0)` para obtener el índice de cada palabra en el diccionario.
2. Si la palabra no está en el diccionario, se asigna 0 (se usa para , palabras desconocidas).

#### 3 calcular\_probabilidad(tokens\_indices, log\_pi, log\_A)

Objetivo: Calcula la log-probabilidad de una secuencia de palabras en base a un modelo de cadenas de Markov.

Proceso:

1. Si `tokens_indices` está vacío, retorna `-np.inf` (evita errores).
2. Toma la log-probabilidad inicial de la primera palabra: `log_pi[tokens_indices[0]]`.
3. Para cada par de palabras consecutivas en la secuencia, suma la log-probabilidad de transición `log_A`.

#### 4 clasificar\_texto(...)

Objetivo: Clasifica un texto como "Borges" o "Benedetti" en función de su probabilidad de generación en cada modelo.

Proceso:

1. Preprocesa el texto → Obtiene los tokens.
2. Convierte los tokens a índices → Transforma palabras en números.
3. Calcula log-probabilidades para ambos modelos (Borges y Benedetti).
4. Convierte log-probabilidades a probabilidades reales para interpretación:
  - Se resta el máximo log-probabilidad para evitar problemas numéricos (underflow).
  - Se aplica `np.exp(log_prob - max_log_prob)` para convertir log a probabilidad.
  - Se normaliza dividiendo entre la suma total.
5. Imprime las probabilidades y retorna la clasificación.

## Ingreso del nuevo texto - Similar a Borges

```
In [...] texto_prueba = """
En la vastedad de la memoria,
donde los ecos de los días transcurridos
se entrelazan con los espejismos del porvenir,
el hombre camina por un laberinto
de palabras y azares.

Sabe, aunque lo olvida a menudo,
que su destino está tejido con hilos invisibles,
urdidos por manos que jamás verá.

Alguna tarde,
quizá en la penumbra de una biblioteca
o en la geometría secreta de un sueño,
descubrirá que su vida no ha sido sino el eco de otras vidas,
el reflejo de una historia que ya ha sido escrita
en un idioma remoto y perfecto.
"""

resultado = clasificar_texto(texto_prueba, log_pi0_norm, log_A0_norm, log_pi1_norm, log_A1_norm, indice_palabras)
print(f"Clasificación: {resultado}")
```

Probabilidad Borges: 1.0000  
Probabilidad Benedetti: 0.0000  
Clasificación: Borges

## Ingreso del nuevo texto - Similar a Benedetti

```
In [... texto_prueba = """
La ciudad despierta con su ritmo de siempre,
entre murmullos de bocacalles
y pasos apurados que no saben bien a dónde van.

En un café cualquiera,
un hombre revuelve su taza con la mirada perdida,
tal vez recordando un amor que ya no está,
tal vez soñando con la ternura que aún no llega.

Afuera, la vida sigue su curso,
con su cuota justa de olvidos y de esperanzas,
con sus rutinas que a veces duelen y a veces salvan.

Porque, al final de cuentas,
lo importante no es cuánto nos golpea el tiempo,
sino con quién elegimos compartirlo.
"""
resultado = clasificar_texto(texto_prueba, log_pi0_norm, log_A0_norm, log_pi1_norm, log_A1_norm, indice_palabras)
print(f"Clasificación: {resultado}")

Probabilidad Borges: 0.0028
Probabilidad Benedetti: 0.9972
Clasificación: Benedetti
```

## Guardar como .py

```
In [... #!/jupyter nbconvert --to script Proyecto8.ipynb
```

## Conclusiones

Se puede observar como los datos de prueba, permiten clasificar los textos con un 75% de precisión, de tal manera que al poner textos de estos poetas, fácilmente podrá indicar quien lo escribió.

---

Mg. Luis Felipe Bustamante Narváez

# Anexo 9

## Proyecto 9: Generador de texto

Mg. Luis Felipe Bustamante Narváez

En este ejercicio realizaremos un generador de texto, basado en el archivo pdf trabajado en el clasificador **mario-benedetti.pdf**. Este archivo, nos permitirá entrenar un modelo de **Markov** de **Segundo Orden**.

Tengamos en cuenta los siguientes detalles de la manera de escribir del autor:



Mario Benedetti

Mario Benedetti (1920-2009), fue un escritor, poeta y periodista uruguayo, considerado una de las figuras más importantes de la literatura latinoamericana. Nació el 14 de septiembre de 1920 en Paso de los Toros, Uruguay. Su obra abarcó poesía, narrativa, ensayo y teatro, con un estilo sencillo y cercano que exploraba el amor, la memoria, el exilio y la lucha social. Durante la dictadura en Uruguay (1973-1985), se exilió en varios países como Argentina, Cuba y España, lo que marcó profundamente su escritura. Entre sus libros más conocidos se encuentran La tregua (1960), Gracias por el fuego (1965) y El amor, las mujeres y la vida (1995). Benedetti falleció el 17 de mayo de 2009 en Montevideo, dejando un legado literario que sigue conmoviendo a lectores de todas las generaciones.

### 1. Temas y Filosofía

Escribe de manera directa y accesible. Sus temas son el amor, la vida cotidiana, la lucha social, el exilio y la esperanza. Su tono es cálido, humano y cercano al lector.

### 2. Lenguaje y Estilo

Usa un lenguaje sencillo, directo y coloquial. Sus poemas parecen conversaciones o pensamientos escritos sin mucha ornamentación.

### 3. Estructura y Ritmo

Prefiere el verso libre y la naturalidad del habla cotidiana, sin preocuparse demasiado por la métrica.

## Librerías

```
In [...]  
import numpy as np  
import string  
import PyPDF2  
import itertools
```

## Convertimos el pdf en .txt

En esta oportunidad, como ya conocemos el manejo de pdf's, vamos a usar un conversor a .txt, para agilizar la manipulación de los datos.

```
In [... def pdf_to_text(pdf_path, txt_path):
    with open(pdf_path, 'rb') as pdf_file:
        reader = PyPDF2.PdfReader(pdf_file)
        text = ''
        for page in reader.pages:
            text += page.extract_text() + '\n'
    with open(txt_path, 'w', encoding='utf-8') as txt_file:
        txt_file.write(text)

    print(f'Texto extraído y guardado en "{txt_path}")')
```

```
In [... # Ruta pdf
pdf = 'textos/mario_benedetti.pdf'
txt = 'textos/mario_benedetti.txt'
pdf_to_text(pdf, txt)
```

Texto extraído y guardado en "textos/mario\_benedetti.txt"

## Diccionarios

### Diccionario inicial

Supongamos que tenemos la siguiente frase:



Necesitamos obtener la primera palabra de cada frase, y las veces que dicha palabra aparece dentro del texto como primera palabra.

Por ejemplo, en el diagrama anterior, la palabra **la** aparece como primera palabra de la frase **la casa del perro** si suponemos que es la única frase de nuestro texto, entonces dicha palabra se repite **1** vez.

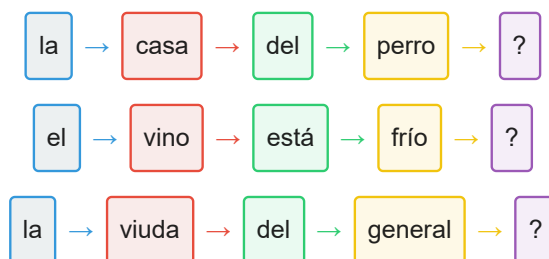
Luego, el diccionario inicial quedaría así:

```
pa_inicial = { 'la': 1 }
```

```
In [... pa_inicial = {}
```

### Diccionario de Markov Primero Orden

Supongamos que tenemos las siguientes frases:



Necesitamos obtener la primera palabra de cada frase, y las veces que dicha palabra aparece dentro del texto como primera palabra, como vimos anteriormente, además necesitamos saber cuales son las secuencias de primer orden, para componer nuestro diccionario.

Luego, el diccionario inicial y el diccionario de primer orden quedarían así:

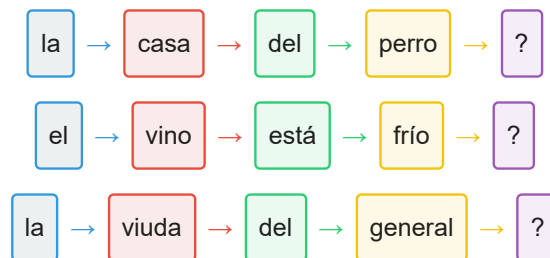
```
pa_inicial = { 'la': 2, 'el': 1 }
```

```
primer_orden = { 'la': ['casa', 'viuda'],  
                 'casa': ['del'],  
                 'del': ['perro', 'general'],  
                 'perro': ['END'],  
                 'el': ['vino'],  
                 'vino': ['frío'],  
                 'frío': ['END'],  
                 'viuda': ['del'],  
                 'general': ['END']  
               }
```

```
In [... primer_orden = {}]
```

## Diccionario de Markov Segundo Orden

Supongamos para el ejemplo anterior:



Necesitamos el conjunto de palabras que anteceden a una palabra cualquiera, para ello, usamos un diccionario donde la clave sean las dos palabras inmediatamente anteriores en el tiempo.

Luego, el diccionario de segundo orden quedaría así:

```
segundo_orden = { ('la', 'casa'): ['del'],  
                  ('casa', 'del'): ['perro'],  
                  ('el', 'vino'): ['está'],  
                  ('vino', 'está'): ['frío'],  
                  ('la', 'viuda'): ['del'],  
                  ('viuda', 'del'): ['general'],  
                  }
```

Notemos, que las claves de cada diccionario, son conjuntos de palabras.

```
In [... segundo_orden = {}
```

```
In [... # Función para remover puntuación y poner en minúsculas
def remove_punct_lower(txt):
    txt = txt.translate(str.maketrans('', '', string.punctuation))
    txt = txt.lower()
    return txt
```

```
In [... # Función para añadir valores al diccionario
def add_dict(dicc, key, value):
    if key not in dicc:
        dicc[key] = []
    dicc[key].append(value)
```

## Cargamos el archivo

```
In [... # Reiniciar diccionarios
pa_inicial = {}
primer_orden = {}
segundo_orden = {}
```

```
In [... # Si necesitamos reiniciar este código para efectos prácticos, basta con reiniciar los dict
with open('textos/mario_benedetti.txt', 'r', encoding='utf-8') as archivo:
    for linea in archivo:
        #print(linea) #comentar
        #llamamos a la función
        tokens = remove_punct_lower(linea).split()
        #print(tokens) #comentar
        T = len(tokens)
        #print(f'Tamaño de la línea: {T}') #Comentar
        #Recorremos los elementos de la fila
        for i in range(T):
            token = tokens[i]
            if i == 0:
                pa_inicial[token] = pa_inicial.get(token, 0.) + 1
                #print(f'Palabra inicial: {token}') #comentar
            else:
                t_1 = tokens[i-1]
                #últimas 2 palabras de cada frase
                if i == T-1:
                    add_dict(segundo_orden, (t_1, token), 'END')
                    #palabras con una sola palabra previa
                    if i == 1:
                        add_dict(primer_orden, t_1, token)
                    else:
                        t_2 = tokens[i-2]
                        add_dict(segundo_orden, (t_2, t_1), token)
```

```
In [... dict(itertools.islice(pa_inicial.items(), 5))
```

```
Out[... {'poemas': 3.0, 'mario': 1.0, 'la': 26.0, 'una': 16.0, 'genera': 1.0}
```

```
In [... dict(itertools.islice(segundo_orden.items(), 5))
```



```
Out[... {'poemas', 'varios'): ['END'],
        ('mario', 'benedetti'): ['END'],
        ('buena', 'tiniebla'): ['END'],
        ('la', 'buena'): ['tiniebla', 'fe', 'suerte'],
        ('una', 'mujer'): ['desnuda', 'desnuda', 'desnuda', 'querida', 'dice']}
```

```
In [... dict(itertools.islice(primer_orden.items(), 3))
```

```
Out[... {'poemas': ['varios', 'a', 'al'],
        'mario': ['benedetti'],
        'la': ['buena',
               'madriguera',
               'lluvia',
               'vida',
               'patria',
               'política',
               'claridad',
               'primavera',
               'misma',
               'luz',
               'válida',
               'madre',
               'incógnita',
               'pareja',
               'pareja',
               'despareja',
               'gente',
               'cosa',
               'querida',
               'culpa',
               'culpa',
               'ebriedad',
               'política',
               'muerte',
               'generosidad',
               'soledad']}
```

## Explicación

Analicemos los resultados de los cinco primeros elementos de cada uno de los diccionarios creados a partir del texto.

1. pa\_inicial:

```
{ 'poemas': 3.0, 'mario': 1.0, 'la': 26.0, 'una': 16.0, 'genera': 1.0 }
```

Indica, por ejemplo que, la palabra **la** aparece como palabra inicial **16.0**.

2. primer\_orden:

```
{
  'poemas': [
    'varios',
    'a',
    'al'
  ],
  'mario': [
    'benedetti'
```

```

],
'la': [
    'buena',
    'madriguera',
    'lluvia',
    'vida',
    .
    .
    .
    'política',
    'muerte',
    'generosidad',
    'soledad'
]
}

```

Indica, por ejemplo que, después de la palabra 'la' aparecen las siguientes palabras: 'buena', 'madriguera', 'lluvia', etc..

3. segundo\_orden:

```

{
('poemas', 'varios'): ['END'],
('mario', 'benedetti'): ['END'],
('buena', 'tiniebla'): ['END'],
('la', 'buena'): [
    'tiniebla',
    'fe',
    'suerte'
],
('una', 'mujer'): [
    'desnuda',
    'desnuda',
    'desnuda',
    'querida',
    'dice'
]
}

```

Indica, por ejemplo que, después de las palabras ('la', 'buena') aparecen las siguientes palabras: 'tiniebla', 'fe' y 'suerte'.

## Probabilidades

### Normalización

#### Palabras iniciales

```

In [... # Calculamos el total de apariciones de todas las palabras iniciales
inicial_total = sum(pa_inicial.values())
inicial_total

```

Out[... 1474.0

```

In [... # Creamos el diccionario inicial de probabilidad
pa_inicial_prob = pa_inicial.copy()

```

```
for key, value in pa_inicial.items():
    pa_inicial_prob[key] = value / inicial_total
```

```
In [... dict(itertools.islice(pa_inicial_prob.items(), 3))
```

```
Out[... {'poemas': 0.0020352781546811396,
        'mario': 0.0006784260515603799,
        'la': 0.017639077340569877}
```

## Función de conversión a probabilidad

```
In [... # Función para convertir las listas de primer orden en diccionarios
# 'poemas': ['varios', 'a', 'al'],
def list_to_pdicc(listas):
    # Creamos un diccionario vacío
    d = {}
    # Obtenemos la longitud de la lista de elementos
    n = len(listas)

    # Ciclo para contar la ocurrencia de cada elemento en la lista
    for lista in listas:
        d[lista] = d.get(lista, 0.) + 1

    # Ciclo para convertir los conteos en probabilidades relativas
    for key, value in d.items():
        d[key] = value / n

    # Retornamos el diccionario de probabilidades
    return d
```

## Primer Orden

```
In [... # Llamamos la función de probabilidad de primer orden
primer_orden_prob = primer_orden.copy()
for t_1, t in primer_orden.items():
    primer_orden_prob[t_1] = list_to_pdicc(t)
```

```
In [... dict(itertools.islice(primer_orden_prob.items(), 2))
```

```
Out[... {'poemas': {'varios': 0.3333333333333333,
                    'a': 0.3333333333333333,
                    'al': 0.3333333333333333},
        'mario': {'benedetti': 1.0}}
```

## Segundo Orden

```
In [... # Llamamos la función de probabilidad de segundo orden
segundo_orden_prob = segundo_orden.copy()
for s, t in segundo_orden.items():
    segundo_orden_prob[s] = list_to_pdicc(t)
```

```
In [... dict(itertools.islice(segundo_orden_prob.items(), 5))
```

```
Out[... ({'poemas', 'varios': {'END': 1.0},
        ('mario', 'benedetti'): {'END': 1.0},
        ('buena', 'tiniebla'): {'END': 1.0},
        ('la', 'buena'): {'tiniebla': 0.3333333333333333,
                          'fe': 0.3333333333333333,
                          'suerte': 0.3333333333333333},
        ('una', 'mujer'): {'desnuda': 0.6, 'querida': 0.2, 'dice': 0.2}}
```

## Función de prueba

Con esta función, realizaremos las pruebas necesarias para evaluar el comportamiento de las probabilidades de aparición de una palabra.

La función recibe dos parámetros, **diccionario** e **imprimir**; el primer parámetro recibe el diccionario de trabajo a evaluar, y el segundo recibe un booleano, para saber si mostramos o no los pasos de la ejecución.

```
In [... def palabra_ejemplo(d, imp):
    # Generamos un número aleatorio en el rango [0, 1]
    p0 = np.random.random()
    if imp:
        print(f"p0:\t\t{p0}")

    # Inicializamos una variable para realizar la suma acumulativa de probabilidades
    cumulative = 0
    if imp:
        print(f"prob. acum.:\t{cumulative}")

    # Ciclo que recorre cada clave (k) y su probabilidad (p) en el diccionario (d)
    for k, p in d.items():
        cumulative += p
        if imp:
            print(f"Prob:\t\t{p}\titem:\t'{k}'")
            print(f"prob. acum.:\t{cumulative}")

    # Comprobamos si el número aleatorio es menor que la acumulación de probabilidades
    if p0 < cumulative:
        # Si la condición se cumple, devuelve la clave (k) seleccionada
        respuesta = f"La palabra siguiente debería ser '{k}'"
        return respuesta if imp else k
```

```
In [... # Verificamos que nos traería una palabra en particular
print(primer_orden_prob['poemas'])

{'varios': 0.3333333333333333, 'a': 0.3333333333333333, 'al': 0.3333333333333333}
```

```
In [... # Llamamos la función
palabra_ejemplo(primer_orden_prob['poemas'], True)

p0:          0.36180865610514357
prob. acum.: 0
Prob:        0.3333333333333333      item:  'varios'
prob. acum.: 0.3333333333333333
Prob:        0.3333333333333333      item:  'a'
prob. acum.: 0.6666666666666666
```

```
Out[... "La palabra siguiente debería ser 'a'"
```

## Generador

### Función del generador de texto

```
In [... def generador(tamaño):
    for i in range(tamaño):
        oracion = []
        #Palabra inicial
        pal_0 = palabra_ejemplo(pa_inicial_prob, False)
```

```

oracion.append(pal_0)
#segunda palabra
pal_1 = palabra_ejemplo(primer_orden_prob[pal_0], False)
oracion.append(pal_1)

# Segundo orden hasta el final
while True:
    pal_2 = palabra_ejemplo(segundo_orden_prob[(pal_0, pal_1)], False)
    if pal_2 == 'END':
        break
    oracion.append(pal_2)
    pal_0 = pal_1
    pal_1 = pal_2
texto = ' '.join(oracion)
print(texto)

```

## Explicación

- Esta función recibe un parámetro tamaño, que representa la cantidad de oraciones que se generarán:

```
def generador (tamaño):
```

- Se inicia un bucle que se ejecutará **tamaño** veces, es decir, generará **tamaño** oraciones.

```
for i in range (tamaño):
```

- Se crea una lista vacía llamada oracion, donde se almacenarán las palabras generadas.

```
oracion = []
```

- Se elige la primera palabra de la oración con la función **palabra\_ejemplo(pa\_inicial\_prob, False)**. **pa\_inicial\_prob** es un diccionario que contiene probabilidades de palabras las iniciales. La palabra seleccionada se agrega a oracion.

```
pal_0 = palabra_ejemplo(pa_inicial_prob, False )
oracion.append (pal_0)
```

- Se elige la segunda palabra, dependiendo de la primera (**pal\_0**). **primer\_orden\_prob** es el diccionario que mapea palabras iniciales a posibles segundas palabras con sus probabilidades. **palabra\_ejemplo(primer\_orden\_prob[pal\_0], False)** selecciona la segunda palabra basada en **pal\_0**. Se agrega **pal\_1** a **oracion**.

```
pal_1 = palabra_ejemplo(primer_orden_prob [ pal_0 ], False )
oracion.append (pal_1)
```

- Se entra en un bucle while True para generar palabras de manera recurrente. **segundo\_orden\_prob** es un diccionario que usa tuplas (**pal\_0, pal\_1**) como clave y devuelve las probabilidades de la siguiente palabra. Se

selecciona la siguiente palabra (**pal\_2**) en función de las dos palabras anteriores.

```
while True:  
    pal_2 = palabra_ejemplo(segundo_orden_prob [ (pal_0, pal_1) ], False )
```

- Si **pal\_2** es 'END', se termina la generación de palabras y se sale del bucle.

```
if pal_2 == 'END':  
    break
```

- Si **pal\_2** no es 'END', se agrega a la lista oracion. Luego, se actualizan las variables: **pal\_0 toma el valor de pal\_1** y **pal\_1 toma el valor de pal\_2**. Esto asegura que en la siguiente iteración se utilicen las dos palabras más recientes para predecir la siguiente.

```
oracion.append (pal_2)  
pal_0 = pal_1  
pal_1 = pal_2
```

- La lista oracion se convierte en una cadena de texto separada por espacios con ' '.join(oracion). Se imprime la oración generada.

```
texto = ' '.join (oracion)  
print (texto)
```

```
In [ ... # Probamos La creación de un poema, de acuerdo con La manera de escritura de Mario Benedetti  
generador(5)
```

```
mi nombre con su hermano el insociable  
en esas noches en que fui un viejo cargado de celos  
mientras los grandes temas  
la muerte  
pero no viceversa por algo en el aire que absorbieron noche a noche
```

## Conclusiones

Aquí construimos un generador de texto basado en modelos de Markov de segundo orden. usando probabilidades de aparición de palabras para construir oraciones de manera secuencial hasta encontrar una palabra de terminación ('END').

# Anexo 10

## Proyecto 10: Spinning de Texto

Mg. Luis Felipe Bustamante Narváez

### Librerías

```
In [... import numpy as np
import pandas as pd
import nltk
from nltk import word_tokenize
from nltk.tokenize.treebank import TreebankWordDetokenizer
import asyncio
from tqdm import tqdm
from colorama import Fore, Back, Style
import os
from itertools import islice
from IPython.display import display, Markdown
```

```
In [... # Descargamos el conjunto de datos del tokenizador en español
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data] C:\Users\luis\AppData\Roaming\nltk_data...
[nltk_data] Package punkt is already up-to-date!
```

Out[... True

### Cargamos los datos

```
In [... # Es común que los archivos vengan codificados con ISO
path = 'data/data_larazon_publico_v2.csv'
path_utf = 'data/new_data.csv'
try:
    df = pd.read_csv(path, encoding='utf-8')
    print('Encoding utf-8')
except Exception:
    print('Encoding ISO-8859-1 a utf-8')
    df_iso = pd.read_csv(path, encoding='ISO-8859-1')
    df_iso.to_csv(path_utf, encoding='utf-8', index=False)
await asyncio.sleep(3) #Espera 3 seg para abrir el nuevo archivo en espera de ser guardad
df = pd.read_csv(path_utf, encoding='utf-8')
```

Encoding utf-8

```
In [... df
```

Out[...]	Unnamed: 0	indi	cuerpo	titular
	0	0	0	dos semanas después de su puesta de largo y pr... el submarino s-80 ya flota
	1	1	1	este viernes, el presidente del gobierno, pedr... calviño y calvo alaban (sin darse cuenta) la g...
	2	2	2	el ministro del interior, fernando grande-marl... el geo de la policía tendrá una nueva sede en ...
	3	3	3	son días muy duros para la familia de olivia y... la madre de las niñas "sobran las palabras par...
	4	4	4	sólo quedan 10 presos de eta por recibir los b... sólo quedan 10 presos de eta por recibir el be...
	...	...	...	...
	58419	58420	18419	la comisión europea inició este un procedimien... bruseles abre un expediente a españa por no de...
	58420	58421	18420	el pleno de la asamblea de madrid ha aprobado ... aprobado el proyecto de ley para que las mujer...
	58421	58422	18421	la comisión de investigación parlamentaria del... la comisión del alvia arranca escuchando a la ...
	58422	58423	18422	erc y pdecat han calificado este jueves de "in... erc y pdecat piden explicaciones a interior po...
	58423	58424	18423	la junta de portavoces del congreso ha acordad... el congreso aplaza la primera sesión de contro...

58424 rows × 4 columns

```
In [ ... df.head()
```

Out[...]	Unnamed: 0	indi	cuerpo	titular
	0	0	0	dos semanas después de su puesta de largo y pr... el submarino s-80 ya flota
	1	1	1	este viernes, el presidente del gobierno, pedr... calviño y calvo alaban (sin darse cuenta) la g...
	2	2	2	el ministro del interior, fernando grande-marl... el geo de la policía tendrá una nueva sede en ...
	3	3	3	son días muy duros para la familia de olivia y... la madre de las niñas "sobran las palabras par...
	4	4	4	sólo quedan 10 presos de eta por recibir los b... sólo quedan 10 presos de eta por recibir el be...

## Creamos la Serie con las noticias

```
In [ ... # Tomamos solamente la columna abstract para crear una serie
textos = df['cuerpo']
```

```
In [ ... textos.head()
```

```
Out[... 0    dos semanas después de su puesta de largo y pr...
1    este viernes, el presidente del gobierno, pedr...
2    el ministro del interior, fernando grande-marl...
3    son días muy duros para la familia de olivia y...
4    sólo quedan 10 presos de eta por recibir los b...
Name: cuerpo, dtype: object
```

```
In [ ... #muestra de las noticias (solo las primeras 500 palabras)
textos[0][:500]
```

```
Out[... 'dos semanas después de su puesta de largo y presentación en sociedad, el primer submarino s-8
0 para la armada, el s-81 "isaac peral", ha entrado hoy en el agua tras una delicada y larga m
aniobra que se ha retrasado varios días por las condiciones meteorológicas. de esta forma, tra
s completar su construcción 17 años después de que arrancara el programa, navantia ha cumplido
otro importante hito.españa.submarino s-80 tras 17 años y 3.900 millones, el "isaac peral" ya
está aquíespaña.el comandante '
```

## Probabilidades

### Matriz de conteo

```
In [ ... # Creamos el diccionario de probabilidad
# key: (w(t-1), w(t+1)), value: {w(t): count(w(t))}
probs = {}
```



```
In [...] # Separador
#variable de formato de la barra de progreso
bar_format_ = (f'{Back.WHITE}{Fore.GREEN}{{l_bar}}{{bar}}{{Style.RESET_ALL}} '
               f'{Fore.CYAN}{{n_fmt}}/{{total_fmt}} '
               f'[[{elapsed}]<[{remaining}]]{{Style.RESET_ALL}}'
               )
for doc in tqdm(textos, bar_format=bar_format_, desc='Creando matriz: '):
    #Separamos cada noticia por puntos
    lineas = doc.split('.')
    for linea in lineas:
        #Tokenizamos cada línea
        tokens = word_tokenize(linea, language='spanish')
        #Mostramos Los tokens
        #print(tokens) #Este proceso tarda bastante, se hace a modo de prueba
        #Condicionamos Las palabras finales
        if len(tokens) >= 2:
            for i in range(len(tokens) - 2):
                t_0 = tokens[i] #palabra anterior
                t_1 = tokens[i+1] #palabra actual
                t_2 = tokens[i+2] #palabra siguiente
                #Creamos la clave del diccionario
                key = (t_0, t_2)
                #preguntamos si la clave no está en el diccionario
                if key not in probs:
                    #asinamos una clave vacía
                    probs[key] = {}
                #preguntamos si la palabra actual no es una clave
                if t_1 not in probs[key]:
                    #asignamos valor inicial de 1 al diccionario de valores de las probs
                    probs[key][t_1] = 1
                else:
                    #sumamos el valor de aparición de la palabra actual
                    probs[key][t_1] += 1

#mostramos las líneas a modo de prueba
#lineas
```

```
100%|██████████ 58424/58424 [06:03<00:00]
```

```
In [...] # Mostramos el diccionario probs, pero solo una parte para hacer corto el proceso
dict(islice(probs.items(),1))
```

```
Out[...] {('dos', 'después'): {'semanas': 95,
                              'años': 283,
                              'días': 296,
                              'meses': 208,
                              'horas': 35,
                              'siglos': 4,
                              'minutos': 4,
                              'décadas': 16,
                              'elecciones': 2,
                              'día': 3,
                              'jornadas': 2,
                              'legislaturas': 1,
                              'domingos': 1,
                              'negocios': 1,
                              'pasiones': 1,
                              'decenios': 1,
                              'iniciativas': 1,
                              'dispositivos': 1}}
```

```
In [... len(probs)
```

```
Out[... 4875993
```

## Normalización

```
In [... # Creamos una copia del diccionario para mantener los datos
d_probs = probs.copy()
#Recorremos las claves y los valores del diccionario probs
for key, d in tqdm(d_probs.items(), bar_format=bar_format_, desc='Normalizando: '):
    #sumamos los valores de repetición de cada una de las palabras
    total = sum(d.values())
    #Recorremos la clave y el valor del diccionario de los valores creado
    for k, v in d.items():
        d[k] = v / total
```

```
Normalizando: 100% [██████████] 4875993/4875993 [00:21<00:00]
```

```
In [... # Mostramos el diccionario d_probs, pero solo una parte para hacer corto el proceso
dict(islice(d_probs.items(),1))
```

```
Out[... {'(dos', 'después)': {'semanas': 0.09947643979057591,
    'años': 0.2963350785340314,
    'días': 0.3099476439790576,
    'meses': 0.21780104712041884,
    'horas': 0.03664921465968586,
    'siglos': 0.004188481675392671,
    'minutos': 0.004188481675392671,
    'décadas': 0.016753926701570682,
    'elecciones': 0.0020942408376963353,
    'día': 0.0031413612565445027,
    'jornadas': 0.0020942408376963353,
    'legislaturas': 0.0010471204188481676,
    'domingos': 0.0010471204188481676,
    'negocios': 0.0010471204188481676,
    'pasiones': 0.0010471204188481676,
    'decenios': 0.0010471204188481676,
    'iniciativas': 0.0010471204188481676,
    'dispositivos': 0.0010471204188481676}}
```

```
In [... len(d_probs)
```

```
Out[... 4875993
```

## Ejemplo de Detokenización

Permite volver a unir los tokens en frases, por ejemplo:

```
In [... detokenizar = TreebankWordDetokenizer()
ejemplo = 'Bootcamp de Inteligencia Artificial'
print(f'Frase original: {ejemplo}')
token_ejemplo = word_tokenize(ejemplo, language='spanish')
print(f'Frase tokenizada: {token_ejemplo}')
detoken_ejemplo = detokenizar.detokenize(token_ejemplo)
print(f'Frase Detokenizada: {detoken_ejemplo}')
```

```
Frase original: Bootcamp de Inteligencia Artificial
```

```
Frase tokenizada: ['Bootcamp', 'de', 'Inteligencia', 'Artificial']
```

```
Frase Detokenizada: Bootcamp de Inteligencia Artificial
```

# Spinner

In [...]

```
# Función de prueba para una palabra random
def sample_word(d):
    p0 = np.random.random()
    cumulative = 0
    for key, p in d.items():
        cumulative += p
        if p0 < cumulative:
            return key
```

In [...]

```
# Función spinner para una línea
# CADA COMENTARIO DONDE ESTÁ EL RETURN ES UN EJEMPLO PARA IR ANALIZANDO EL CÓDIGO
def spin_line(linea, imp):
    tokens = word_tokenize(linea, language='spanish')
    i = 0
    salida = [tokens[0]]
    #return salida #ejemplo de ejecución --- comentar
    if len(tokens) >= 2:
        while i < (len(tokens) - 2):
            t_0 = tokens[i] #palabra anterior
            t_1 = tokens[i+1] #palabra actual
            t_2 = tokens[i+2] #palabra siguiente
            #creamos la clave
            key = (t_0, t_2)
            #creamos el diccionario de distribución
            p_dist = d_probs[key]
            #i = 1100000 #Para desbordar el while ----- comentar
            #return p_dist #ejemplo de ejecución ---- comentar
            #Cuando el diccionario tenga más de una palabra y un spinning del x%
            if len(p_dist) > 1 and np.random.random() < 0.3:
                #selecciona una palabra al azar de la función de prueba de palabras
                middle = sample_word(p_dist)
                #i = 1100000 #Para desbordar el while ----- comentar
                #return middle #ejemplo de ejecución ---- comentar

                #Validamos si deseamos mostrar la palabra de cambio automáticamente
                # Si imp es True, muestra el texto cambiado
                # Si imp es False, muestra la palabra actual y el cambio que sugiere
                if imp:
                    #agregamos la palabra nueva en la posición t_1
                    salida.append(middle)
                    #agregamos la palabra t_2, que va al final
                    salida.append(t_2)
                    #movemos el cursor 2 posici. para que no haga 2 spin en 2 pal. seguidas
                    i += 2
                else:
                    #agregamos a la salida la palabra t_1, es decir la que queremos cambiar
                    salida.append(t_1)
                    #agregamos, para visualizar, la palabra por la que nos va a cambiar
                    salida.append('<' + middle + '>')
                    #agregamos la palabra t_2, que va al final
                    salida.append(t_2)
                    #movemos el cursor dos posici. para que no haga dos spin en 2 pal seguidas
                    i += 2
                #en caso que el diccionario sea <= 1 o que el random no entre al spinner
            else:
                #agregamos la palabra siguiente y ubicamos el cursor en la siguiente palabra
```

```

        salida.append(t_1)
        i += 1
    # si ya estamos en la última palabra a poner a prueba
    if i == len(tokens) - 2:
        #agregamos la última palabra al diccionario
        salida.append(tokens[-1])
    # retornamos la salida detokenizada ya que es una lista ESTE NO SE COMENTA, ES EL FIN
    detoken = detokenizar.detokenize(salida)
    return detoken

```

```

In [... # Función spinner para recorrer el documento
def spin_document(doc, imp):
    lineas = doc.split('.')
    output = []
    for linea in lineas:
        if linea:
            new_line = spin_line(linea, imp)
        else:
            new_line = linea
        output.append(new_line)
    #corregimos el posible error de tener cadenas en None
    try:
        return '\n'.join(output)
    except Exception:
        return '\n'.join(filter(None, output))

```

```

In [... #Código para pruebas de creación
#spin_document('dos años después cómo están')
#spin_line('dos años después cómo están')

```

## Texto (noticia) de prueba para el modelo

```

In [... #Recordemos qué tenía nuestro df textos
textos.head()

```

```

Out[... 0    dos semanas después de su puesta de largo y pr...
      1    este viernes, el presidente del gobierno, pedr...
      2    el ministro del interior, fernando grande-marl...
      3    son días muy duros para la familia de olivia y...
      4    sólo quedan 10 presos de eta por recibir los b...
      Name: cuerpo, dtype: object

```

```

In [... #seleccionamos un índice cualquiera de alguna noticia del df textos
i = np.random.choice(textos.shape[0])
display(Markdown('---'))
display(Markdown(f'**Índice seleccionado:** {i}'))
display(Markdown('---'))
#tomamos el texto que se encuentra en dicho índice
doc = textos.iloc[i]
#Recortamos el texto, solo para mostrarlo; no se altera el texto inicial
doc_recortado = doc.split() #separamos el texto en palabras
doc_recortado = ' '.join(doc_recortado[:100])
display(Markdown(f'**Texto seleccionado:**'))
print(f'{doc_recortado}...')
display(Markdown('---'))

```

```

#Generamos el Spinning Article - Text

```

```
imp = True
new_doc = spin_document(doc, imp)
```

```
#Recortamos el nuevo texto, solo para mostrarlo; no se altera el texto generado por el spin
new_doc_recortado = new_doc.split() #separamos el texto en palabras
new_doc_recortado = ' '.join(new_doc_recortado[:100])
display(Markdown(f'**Texto Spinning:**\n\n'))
print(f'{new_doc_recortado}...')
display(Markdown('---'))
```

---

**Índice seleccionado:** 35576

---

#### **Texto seleccionado:**

un centenar de taxistas se han concentrado a las 8,00 horas de este viernes en la entrada del cementerio de la almudena de madrid y, pasada esta hora, tenían bloqueada la zona. esta acción se produce en el marco de las protestas que está protagonizando el colectivo desde el pasado lunes por la regulación del sector. se trata de la primera acción del día, que ira acompañada de otras como el comienzo de la huelga de hambre que van a iniciar 16 compañeros a partir de las 10,00 horas de hoy en los alrededores de ifema, centro de operaciones de...

---

#### **Texto Spinning:**

un centenar de taxistas se han concentrado a las 2,00 horas de este campo en la entrada del cementerio de la almudena de presentarlo y la pasada esta hora, tenían bloqueada la zona esta cifra se produce en el toque de las protestas que está protagonizando el colectivo según el próximo lunes por una regulación del sector se trate de la primera acción del día después que ira acompañada de otras sin el comienzo de la historia de hambre que maltratan a iniciar 16 compañeros a cambio de las 10,00 horas de hoy mantener los sistemas de ifema, instructor de...

---

## **Errores de tipo NoneType - Análisis**

Cuando existe un valor None en el output de la función spin\_document, no se puede definir el nuevo texto sugerido. Para solucionar, basta con filtrar el output antes de hacer el join.

```
'\n'.join(filter(None, output))
```

## **Conclusiones**

El Article Spinning, permite realizar cambios de palabras con el fin de brindar otra opción a un texto ya construido y cambiarle sus palabras de modo que conserve la idea contextual, pero con otro estilo de escritura. El uso de N-Grams através de las cadenas de Markov, permiten utilizar las probabilidades de ocurrencia de una palabra cuando ésta se encuentra en medio de dos palabras previamente entrenadas. Aunque el modelo es bueno, se requiere de un filtro de fuentes más preciso de un tema en específico, pero este es un sencillo ejemplo que nos deja el desafío de usar Spinning Text dentro de NPL.

---