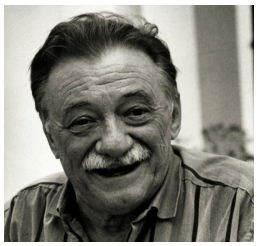
Anexo 9

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 [...]
```python
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 [...]
```python
# 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.

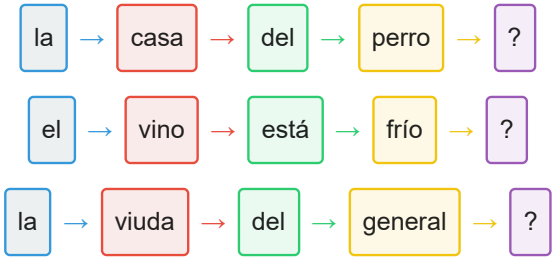
Por ejemplo, en el diagrama anterior, la palabra **la** aparece como primera palabra de la frase **la casa del perro** si suponemos que es la única frase de nuestro texto, entonces dicha palabra se repite **1** vez.

Luego, el diccionario inicial quedaría así:

In [...]
```python
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 }
```

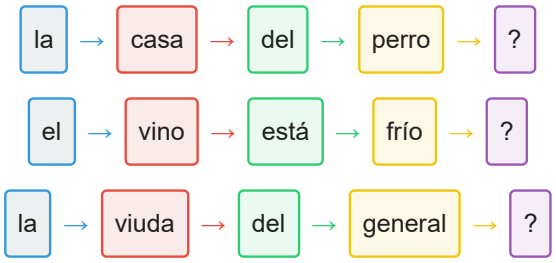
```
primer_orden = { 'la': ['casa', 'viuda'],
                 'casa': ['del'],
                 'del': ['perro', 'general'],
                 'perro': ['END'],
                 'el': ['vino'],
                 'vino': ['frío'],
                 'frío': ['END'],
                 'viuda': ['del'],
                 '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 cnvertir 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 [...
```python
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 [...
```python
# Verificamos que nos traería una palabra en particular
print(primer_orden_prob['poemas'])
```

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

In [...
```python
# 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 [...
```python
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 recelos
mientras los g randes 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').

Mg. Luis Felipe Bustamante Narváez