# TRABAJO PRÁCTICO NLP

*Procesamiento Del Lenguaje Natural*

*Tecnicatura Universitaria en Inteligencia Artificial*

**Tomás Navarro Miñón**

**Ramiro Sagrera**

**Salvador Sanchez**

17/06/2023

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

# INDICE

# INTRODUCCIÓN

La finalidad de este trabajo es la utilización del conocimiento adquirido en la cátedra de Procesamiento del Lenguaje Natural donde mediante una serie de ejercicios podemos demostrar lo aprendido hasta la unidad 3. El mismo consta de 5 desafíos distintos donde debemos superarlos con diferentes técnicas utilizadas.

# EJERCICIO 1:

Para este ejercicio necesitamos realizar un procedimiento de web scraping, para ello se crearon las siguientes funciones.

- **obtener_enlaces_noticias:** Toma una URL del tipo de noticias y la cantidad de URLs que se desea obtener y devuelve todas las URLs de las noticias que allí se encuentran.
- **web_scraping:** Realiza web scraping de un artículo de noticias a partir de la URL de una noticia retornando el título y el cuerpo de esta.
- **construir_dataset:** Construye un df a partir de una categoría y una lista de URLs con ayuda de la función web_scraping.

El script se ejecuta con el comando:

```
python .\ejercicio1.py
```

El proceso de ejecución inicia con la carga de librerías necesarias.

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
```

obtenemos las URLs de cada categoría que vamos a necesitar con ayuda de la función obtener_enlaces_noticias.

```
urls_policiales =
obtener_enlaces_noticias('https://www.rosario3.com/seccion/policiales/', 20)
urls_deporte =
obtener_enlaces_noticias('https://www.rosario3.com/seccion/deportes/', 20)
urls_politica =
obtener_enlaces_noticias('https://www.rosario3.com/seccion/politica/', 20)
urls_tecnologia =
obtener_enlaces_noticias('https://www.rosario3.com/seccion/tecnologia/', 20)
```

Con las URLs obtenidas generamos el DataFrame concatenando los datos de cada tipo de noticia.

```
df_policiales = construir_dataset("Policiales", urls_policiales)
df_deporte = construir_dataset("Deporte", urls_deporte)
df_politica = construir_dataset("Politica", urls_politica)
df_tecnologia = construir_dataset("Tecnologia", urls_tecnologia)

df_completo = pd.concat([df_policiales,
df_deporte,df_politica,df_tecnologia], ignore_index=True)
```

Por ultimo guardamos el dataset en un archivo .csv para su posterior análisis.

```
df_completo.to_csv("dataset_tp2.csv", index=False)
```

## EJERCICIO 2:

Con el dataset obtenido en el ejercicio anterior, lo que se nos pedía es poder realizar un modelo predictivo que pudieses clasificar noticias solamente con el título de las mismas.

Para esto utilizamos la librería de Python conocida como Scikit-Learn donde de la misma importamos para la realización del ejercicio.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import nltk
from nltk.corpus import stopwords
```

Lo primero que hacemos es abrir el archivo con pandas para poder crear un dataframe, facilitando así la comprensión y manipulación de los mismos por parte nuestra.

El proceso siguiente es realizar el etiquetado de los mismos. Las regresiones logísticas a la hora de clasificar no entienden de palabras, entienden de números y en el caso de las noticias de vectores. Para esto lo que hacemos es etiquetar las noticias y poder entrenar de esta manera nuestra regresión logística que se utilizara para clasificación. El paso siguiente es convertir todas las noticias a minúsculas y nos queda eliminar las stopwords (esto se realiza ya que las stopwords son palabras comunes en nuestro vocabulario, pueden ser artículos, pronombres, etc. Son palabras que repetimos mucho), y nos queda vectorizar para poder realizar la regresión logística.

```
df=pd.read_csv(r"C:\Users\tomas\OneDrive\Desktop\facultad\4to
cuatrimestre\Procesamiento del lenguaje natural\TP_1\dataset_tp2.csv")

categoria_mapping = {'Policiales': 0, 'Politica': 1, 'Deporte': 2, 'Tecnologia':
3}
df["labels"]=df["Categoría"].map(categoria_mapping)

#nltk.download('stopwords')
spanish_stop_words = stopwords.words('spanish')

X = df["Titulo"].str.lower()
y = df["labels"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

vectorizer = TfidfVectorizer(stop_words=spanish_stop_words)
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)
```

Una vez vectorizadas las palabras y eliminadas las stopwords, nos queda separar los datos para poder hacer validación cruzada, la validación cruzada es el proceso por el cual el modelo se testea. Ya que no podemos entrenar y con los mismos datos válidos, no tendría sentido, sería algo similar a estudiar un ejercicio y que en el examen nos tomen el mismo ejercicio, ¡una suerte total! pero no estaríamos realizando una evaluación verdaderamente.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

vectorizer = TfidfVectorizer(stop_words=spanish_stop_words)
X_train_vectorized = vectorizer.fit_transform(X_train)
X_test_vectorized = vectorizer.transform(X_test)


modelo_LR = LogisticRegression()
modelo_LR.fit(X_train_vectorized, y_train)
```

Realizamos el proceso mencionado y obtenemos métricas, a la hora de realizar modelos predictivos, las métricas son muy útiles porque nos dicen de qué tan bueno es el modelo. Hay muchas, pero en este caso aplicamos unas conocidas como accuracy es una métrica que mide la proporción de predicciones correctas realizadas por un modelo de machine learning y también realizamos un reporte de todas las métricas

```
# Evaluación del modelo de Regresión Logística
y_pred_LR = modelo_LR.predict(X_test_vectorized)
acc_LR = accuracy_score(y_test, y_pred_LR)
report_LR = classification_report(y_test, y_pred_LR)

print("Precisión Regresión Logística:", acc_LR)
print("Reporte de clasificación Regresión Logística:\n", report_LR)
```

Y ya para terminar de evaluar el modelo, sobre el final del archivo, realizamos una muestra visual para que nosotros podamos evaluar cómo clasifica y lo hacemos predecir.

```
nuevas_frases = [
    "Inteligencia artificial y biología sintética: las tecnologías que, según un experto, podrían beneficiar o destuir a la humanidad",
    "Tenía pedido de captura por tentativa de homicidio y cayó por disparos en una plaza",
    "Lo asaltaron y le robaron la moto cuando iba a trabajar",
    "El récord de Rosario Central en el Gigante de Arroyito que supera al Manchester City",
    "Fiscal impulsa una investigación contra Milei y Marra por la corrida contra el peso",
    "Massa creció en Santa Fe pero Milei fue el más votado: tercios en Diputados y un socialista",
    "Elecciones 2023: las cinco boletas que hay este domingo en el cuarto oscuro",
    "Video: el recibimiento de los hinchas de Central en el Gigante",
    "Robaron una moto a mano armada y escaparon a los tiros",
    "Usó la inteligencia artificial para mostrar cómo se verían nietos robados en dictadura: "
]


nuevas_frases = [frase.lower() for frase in nuevas_frases]


nuevas_frases_vectorizadas = vectorizer.transform(nuevas_frases)


etiquetas_predichas = modelo_LR.predict(nuevas_frases_vectorizadas)

for i, etiqueta in enumerate(etiquetas_predichas):
    categoria_predicha = [key for key, value in categoria_mapping.items() if value == etiqueta][0]
    print(f"La frase '{nuevas_frases[i]}' pertenece a la categoría: {categoria_predicha}")
```

## EJERCICIO 3:

Para cada categoría de noticias vamos a procesar el texto obtenido y luego realizar un conteo de palabras y mostrar la importancia de las mismas mediante una nube de palabras.

Para ello primero vamos a importar las librerías necesarias:

```
import pandas as pd
import unicodedata
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize, sent_tokenize
import es_core_news_sm
from nltk.probability import FreqDist
from wordcloud import WordCloud
import matplotlib.pyplot as plt
```

Luego creamos las funciones para limpieza:

- Conversión de texto a minúsculas

```
def convert_lower(df, column):
      """
      Función que recibe un dataframe y convierte la columna
      de texto a minúsculas

      """

      df[column] = df[column].str.lower()

      return df
```

- Eliminación de signos de puntuación

```
def delete_puntuation(df, column):
      """
      Función que recibe un dataframe y
      elimina todos los signos de puntuación del texto

      """

      df[column] = df[column].str.replace('[^\w\s]', '')

      return df
```

- Eliminación de tildes

```
def remove_accents(text):
      """
      Función que elimina todos las tildes del texto

      """

      nfkd_form = unicodedata.normalize('NFKD', text)

      return ''.join([c for c in nfkd_form if not
unicodedata.combining(c)])
```

- Eliminación de stopwords

```
def remove_stopwords(df, column, stop_words):
      """
      Función que recibe un dataframe y elimina palabras de parada del
texto

      """

      df[column] = df[column].astype(str)
      df['Texto_Limpio'] = df[column].apply(lambda x: ' '.join([word for
word in word_tokenize(x) if word.lower() not in stop_words]))

      return df
```

- Lematización del texto

```
def lemmatisation(df, column, nlp):
      """
      Función que recibe un dataframe, lematiza el texto y lo agrega al
dataframe

      """

      lemmas = []
      for text in df[column]:
      doc = nlp(text)
      lemma_list = [token.lemma_ for token in doc]
      lemmas.append(' '.join(lemma_list))
      df['Texto_Lemat'] = lemmas

      return df
```

- Contabilización de palabras:

```
def word_count(text):
      """
      Función que recibe un dataframe y cuenta la cantidad de palabras
de un texto

      """

      word_tokens = word_tokenize(text)

      return len(word_tokens)
```

- Nube de palabras general

```
def words_cloud(df, column):
      """
      Función que realiza una visualización de la frecuencia
      mediante nube de palabras

      """

      text = ' '.join(df[column])
      wordcloud = WordCloud(width = 800, height = 800,
      background_color ='white',
      min_font_size = 10).generate(text)

      plt.figure(figsize = (8, 8), facecolor = None)
      plt.imshow(wordcloud)
      plt.axis("off")
      plt.title('General', fontsize=20)
      plt.tight_layout(pad = 0)
      plt.show()
```

- Nube de palabras por categoría

```
def words_cloud_category(df, text_column, category_column,
category_value):
      """
      Función que genera una nube de palabras para un valor específico
en la columna de categoría

      """

      filtered_df = df[df[category_column] == category_value]
      text = ' '.join(filtered_df[text_column])
```

```
      wordcloud = WordCloud(width=800, height=800,
background_color='white', min_font_size=10).generate(text)

      plt.figure(figsize=(8, 8), facecolor=None)
      plt.imshow(wordcloud)
      plt.axis("off")
      plt.title(category_value, fontsize=20)
      plt.tight_layout(pad=0)
      plt.show()
```

Luego, cargamos el dataframe generado en el ejercicio 2 y seteamos el nombre de la columna a trabajar:

```
df = df = pd.read_csv('dataset_tp2.csv')
column = 'Texto'
```

Convertimos todo el texto a minúsculas y eliminamos todos los signos de puntuación del texto:

```
df = convert_lower(df, column)
df = delete_puntuation(df, column)
```

Definimos las palabras de parada y creamos una nueva columna sin ellas:

```
nltk.download('stopwords')
nltk.download('punkt')
stop_words = set(stopwords.words('spanish'))
df = remove_stopwords(df, column, stop_words)
```

Lematizamos el texto (sin stopwords) utilizando el paquete de spaCy para lenguaje español:

```
nlp = es_core_news_sm.load()
df = lemmatisation(df, 'Texto_Limpio', nlp)
```

Aplicamos la función para contar palabras, las agregamos al datafame e imprimimos el resultado:

```
df['Cant_Palabras'] = df[column].apply(lambda x: word_count(x))
print(df)
```

Realizamos una nube de palabras para cada categoría:

```
for i in df['Categoría'].unique():
      words_cloud_category(df, 'Texto_Limpio', 'Categoría', i)
```

Realizamos una nube de palabras general

```
words_cloud(df, 'Texto_Limpio')
```

Exportamos el dataframe a un archivo csv:

```
df.to_csv("dataset_tp2_procesado.csv", index=False)
```

Conclusión del resultado obtenido:

Vemos que las nubes de palabras creadas por categorías contienen muchas palabras que no deberían tener una gran cantidad de apariciones, como nombres de calles o apellidos. Esto creemos que se da porque las mismas están creadas con una baja cantidad de artículos de noticias y esto hace que el número general de apariciones de palabras sea bajo. Cuando analizamos la nube de palabras de todas las categorías sin discriminar, vemos que este inconveniente se matiza un poco.

De todas maneras, también observamos que el proceso de stopwords no se realiza de forma óptima ya que vemos algunas palabras que consideramos que al usarlas individualmente sin acompañar a las palabras claves del texto carecen de sentido explicativo de un texto. Esto puede darse porque la biblioteca NLTK no se encuentre optimizada para el idioma español.

## EJERCICIO 4:

En el ejercicio 4 nos pidieron que realizáramos un modelo de embedding basado en Sentence-BERT y realizaramos una reflexión de los datos obtenidos.

Para la ejecucion del script ejecutamos el siguiente comando dentro del entorno de python.

```
python .\ejercicio4.py
```

El script se compone de las siguientes funciones:

- **generar_embeddings:** Toma un DataFrame de datos y un modelo de embedding de texto como entrada, itera a través de las categorías únicas en el DataFrame de datos, calcula los embeddings de los títulos en cada categoría y retorna un nuevo DataFrame que almacena las categorías, los títulos y sus correspondientes embeddings.
- **calcular_similitud:** Calcula la similitud de coseno entre dos tensores.
- **calcular_centro_vectores:** Dado un DataFrame y una categoría calcula el centroide de los vectores de una categoría especificada utilizando la función calcular_similitud.
- **similitudes_por_categoria:** Toma un df y un vector y calcula las similitudes entre un vector y los datos de todas las categorías del DataFrame.

El proceso de ejecución es el siguiente:

Cargamos las librerias necesarias.

```
import tensorflow_text
import tensorflow.compat.v2 as tf
import tensorflow_hub as hub
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Cargamos el modelo de embedding y los datos de las noticias de dataset.csv.

```
module_url =
"https://tfhub.dev/google/universal-sentence-encoder-multilingual/3'
model = hub.load(module_url)

df = pd.read_csv('dataset.csv')
```

Generamos un dataset con los tensores de cada título con generar_embeddings.

```
df_embeddings = generar_embeddings(df, model)
```

Dado que debemos realizar un análisis con una sola categoría escogimos la categoría “Policiales” por lo que calculamos su centroide.

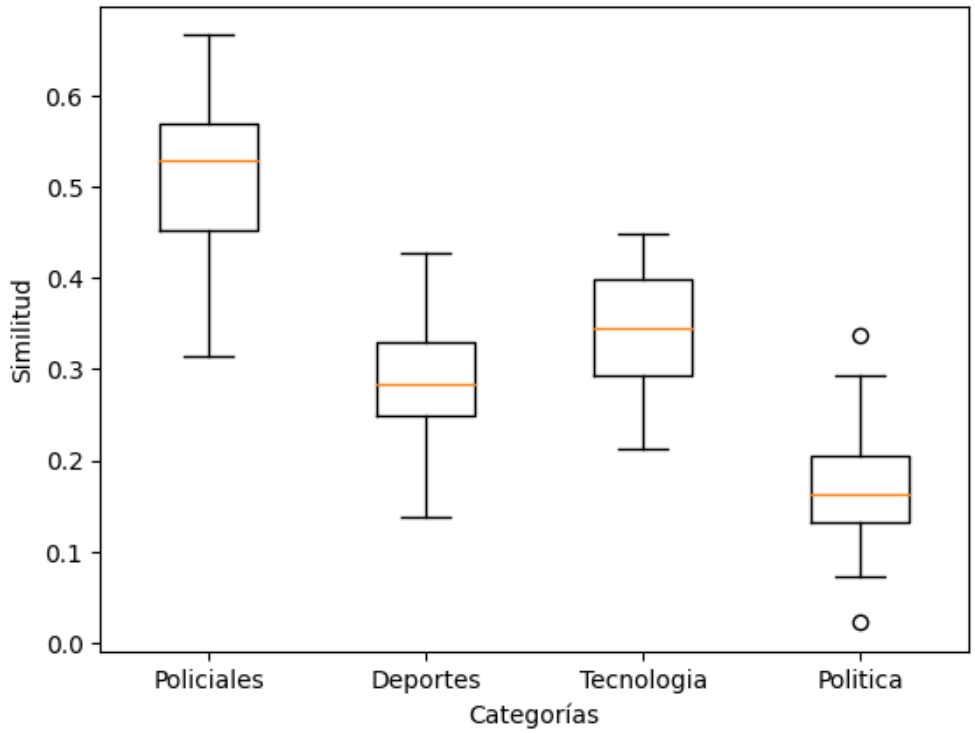
```
centro_policiales = calcular_centro_vectores(df_embeddings,
'Policiales')
```

Y analizamos las similitudes entre este centroide y todas las categorías disponibles realizando un gráfico de caja.

```
data = similitudes_por_categoria(df_embeddings, centro_policiales)

# Gráfico de caja
plt.boxplot(data, labels=['Policiales', 'Deportes', 'Tecnologia',
'Politica'])
plt.xlabel('Categorías')
plt.ylabel('Similitud')
plt.show()
```

Obtuvimos el siguiente gráfico con las similitudes de coseno.




Como era de esperar la categoría con más similitud con el centroide de “Policiales” es esta misma, también observamos que las similitudes se agrupan uniformemente entre cada categoría, sin embargo hay muchos datos donde tienen una similitud considerable al centroide de los títulos Policiales cuando uno esperaría que se diferencien considerablemente entre sí. Por estas razones podemos concluir que se podría utilizar la similitud del coseno para realizar una clasificación pero daría lugar a varias clasificaciones incorrectas si no se establece un umbral adecuado.

## EJERCICIO 5:

Para la realización del ejercicio 5, se nos pidió la creación de un programa interactivo que mediante la elección del usuario, el mismo realice un resumen de las noticias de la categoría.

Para esto lo primero que hicimos fue importar las librerías necesarias.

```
import pandas as pd
from transformers import T5Tokenizer, T5ForConditionalGeneration
import textwrap
```

Luego, posteriormente realizamos la carga del modelo desde hugging face, hugging face es un sitio que funciona como un github de los programadores de inteligencia artificial. En el programadores de todo el mundo utilizan hugging face para subir sus modelos open sources y compartirlos con la comunidad. Haciendo uso de esto nosotros tomamos y cargamos el modelo multipropósito T5. Este modelo considera todas las tareas de NLP como una conversión de texto a texto, lo que significa que tareas como traducción, resumen, y preguntas y respuestas se formulan de una manera en que la entrada y la salida son secuencias de texto. En este caso usamos t5-small, pero también existen otros tamaños de modelos (t5-base, t5-large, t5-3b, t5-11b) con diferentes cantidad de parámetros.

Cargamos este modelo para realizar un resumen abstractivo, o sea, que sea un resumen del contenido de la noticia generado por nuestro modelo. Al ser text-to-text genera un resumen propio del mismo y no por recortes de lo mencionado en la entrada de datos como sería el caso de un resumen extractivo.

También lo que hacemos en este paso es cargar los datos del dataset que obtuvimos en el ejercicio 1 y filtrarlos en su columna categoría por los valores únicos.

```
model_name = "t5-small"  # Modelo T5
model = T5ForConditionalGeneration.from_pretrained(model_name)
tokenizer = T5Tokenizer.from_pretrained(model_name)

df = pd.read_csv(r"C:\Users\tomas\OneDrive\Desktop\facultad\4to
cuatrimestre\Procesamiento del lenguaje natural\TP_1\dataset_tp2.csv")
# Reemplaza con tu ruta

categorias = df["Categoría"].unique()
```

Lo que hacemos aquí es crear una función donde para cada noticia de la categoría nos genere un resumen etiquetado. Recordemos como explicamos anteriormente, este modelo genera un resumen abstractivo por lo que genera el mismo modelo texto a texto la salida que obtendremos. Como parámetro del modelo le ponemos que el texto que genere no supere las 200 tokens (ósea palabras) y que sea mayor a 50 tokens y agregamos lo resumido a una lista que posteriormente imprimimos en consola

```
def resumir_noticias(noticias, tokenizer, model):
    resumenes = []
    for noticia in noticias:
        inputs = tokenizer.encode("summarize: " + noticia,
return_tensors="pt", truncation=True)
        summary_ids = model.generate(inputs, max_length=200,
min_length=50, length_penalty=2.0, num_beams=4, early_stopping=False)
        resumen = tokenizer.decode(summary_ids[0],
skip_special_tokens=True)
        resumenes.append(resumen)
    return resumenes
```

Posteriormente lo único que nos queda es generar el programa interactivo donde imprimimos por consola el resumen de las noticias para cada categoría que el usuario solicite.

```
while True:
    print("Categorías disponibles:", categorias)
    categoria_seleccionada = input("Selecciona una categoría (o 'salir'
para salir): ")
    if categoria_seleccionada.lower() == "salir":
        break
    if categoria_seleccionada in categorias:
        df_categoria = df[df["Categoría"] == categoria_seleccionada]
        noticias_categoria = df_categoria["Texto"].tolist()
        resumenes_noticias = resumir_noticias(noticias_categoria,
tokenizer, model)

        print(f"Resúmenes de la categoría {categoria_seleccionada}:")
        for i, resumen in enumerate(resumenes_noticias, start=1):
            print(f"Resumen de noticia {i} - {resumen}")
    else:
        print("Categoría no válida.")
```