



Universidad Nacional de Rosario  
Facultad de Ciencias Exactas, Ingeniería y Agrimensura  
Tecnicatura Universitaria en Inteligencia Artificial  
Procesamiento de Imágenes I - IA 4.4

---

# Informe Trabajo Práctico 1:

“Clasificador de Recomendaciones Recreativas utilizando Procesamiento de Lenguaje Natural (NLP)”

## Integrantes:

- Longo, Gonzalo
- Yañez, Mirian

## Índice

1.	Introducción.....	3
2.	Datasets.....	4
3.	Dependencias.....	5
4.	Implementación.....	6
4.1.	Scraper de libros.....	6
4.2.	Clasificación del Estado de Ánimo.....	8
4.3.	Recomendación de actividades.....	13
4.4.	Aplicación.....	16
5.	Conclusión.....	10
6.	Anexo.....	21

## 1. Introducción

El presente trabajo tiene como objetivo desarrollar un programa de Procesamiento de Lenguaje Natural que recomiende actividades recreativas según una preferencia y estado de ánimo del usuario. Las recomendaciones pueden incluir ver una película, jugar un juego de mesa o leer un libro, proporcionando opciones personalizadas para maximizar la satisfacción del usuario.

Para alcanzar este objetivo, el trabajo se basa principalmente en dos modelos de inteligencia artificial: un **clasificador**, encargado de detectar y categorizar el estado de ánimo del usuario, y un **recomendador**, que sugiere actividades recreativas adecuadas. Además, se desarrolló un **scraper** que, mediante técnicas de web scraping, obtiene un conjunto de datos de libros para enriquecer las recomendaciones.

Finalmente, se implementó una **aplicación** que integra tanto el clasificador como el recomendador, facilitando la interacción del usuario con el sistema de una manera intuitiva y accesible.

## 2. Datasets

Para acceder a los datasets utilizados en este proyecto, se incluye en el **Anexo** un enlace que dirige a la carpeta de Google Drive correspondiente.

### 2.1 Dataset de Juegos de Mesa (bgg\_database.csv)

- **Fuente:** Archivo en formato CSV.
- **Descripción:** Contiene información sobre una variedad de juegos de mesa, incluyendo detalles como el título, descripción y otros atributos relevantes.
- **Cantidad de Datos:** 1,000 registros.
- **Uso en el Proyecto:** Proveen recomendaciones de juegos de mesa.

### 2.2 Dataset de Películas (IMDB-Movie-Data.csv)

- **Fuente:** Archivo en formato CSV.
- **Descripción:** Contiene información de películas, incluyendo título, descripción, género y otros detalles.
- **Cantidad de Datos:** 1000 registros.
- **Uso en el Proyecto:** Proveen recomendaciones de películas.

### 2.3 Dataset de Libros (Web Scraping de Proyecto Gutenberg)

- **Fuente:** Web scraping del [Proyecto Gutenberg](#).
- **Descripción:** Incluye información de los 1000 libros más populares del Proyecto Gutenberg.
- **Cantidad de Datos:** 1000 registros.
- **Uso en el Proyecto:** Generar recomendaciones de libros.

### 2.4 Dataset de Emociones Sintético (dataset\_emociones\_sintetico.csv)

- **Fuente:** Generación de datos sintéticos.
- **Descripción:** Contiene textos clasificados en tres emociones: *Alegre*, *Ni fu ni fa* (neutral) y *Melancólico*. Los textos fueron generados aleatoriamente con frases representativas de cada emoción.
- **Cantidad de Datos:** 10002 registros (3334 por categoría).
- **Uso en el Proyecto:** Entrenar un modelo para la detección de emociones en textos, lo cual permitirá clasificar el estado emocional del usuario.

En total, el proyecto dispone de 3000 registros sobre actividades, distribuidos entre los tres primeros datasets, lo que ofrece una base robusta para generar recomendaciones personalizadas y pertinentes.

Por otro lado, se tiene una base equilibrada de 10002 registros sintéticos que permiten entrenar un modelo para identificar y clasificar emociones en textos.

### 3. Dependencias

Para implementar el programa, es necesario instalar varias librerías de Python que facilitan diversas funcionalidades del sistema.

La instalación de todas ellas se realizó a través de **pip**, el gestor de paquetes de Python.

A continuación se detallan las principales dependencias y su instalación:

#### Instalación de Dependencias

Se utilizarán las siguientes librerías:

- **gdown**: Permite descargar archivos de Google Drive fácilmente.
- **beautifulsoup4**: Herramienta para analizar y extraer datos de documentos HTML y XML.
- **lxml**: Librería de procesamiento de XML y HTML con rendimiento superior.
- **faiss-cpu**: Librería para la búsqueda y agrupación de vectores en grandes conjuntos de datos.
- **sentence-transformers**: Proporciona modelos para transformar oraciones en vectores, útil para comparación de textos.
- **mtranslate**: Permite traducir texto entre diferentes idiomas usando la API de Google Translate.
- **ipywidgets**: Herramientas para crear widgets interactivos en Jupyter notebooks.
- **imbalanced-learn**: Librería para el manejo de datos desbalanceados en aprendizaje automático, proporcionando técnicas para balancear conjuntos de datos.

## 4. Implementación

Para una comprensión detallada del código fuente utilizado en este proyecto, se remite al **Anexo**, donde se proporciona un enlace al documento de Google Colab con el código fuente desarrollado.

### 4.1 Scraper de Libros

El primer paso en la implementación del sistema de recomendaciones fue el desarrollo de un scraper para obtener un conjunto de datos de libros que enriquecería las sugerencias ofrecidas al usuario. A continuación, se detalla el proceso de creación y funcionamiento del scraper.

#### 4.1.1. Librerías

Para llevar a cabo el scraping, se utilizaron las siguientes librerías de Python:

- **ssl**: Se empleó para configurar el contexto HTTPS y evitar problemas de verificación de certificados SSL/TLS al realizar solicitudes.
- **requests**: Esta librería se utilizó para realizar solicitudes HTTP a la página web de Gutenberg y obtener el contenido de las páginas que contienen la lista de libros.
- **BeautifulSoup**: Esta librería se utilizó para parsear el contenido HTML de la página web, facilitando la extracción de datos específicos como títulos, autores y resúmenes de los libros.
- **csv**: Se empleó para guardar la información extraída en un archivo CSV, que puede ser fácilmente manipulado y consultado más adelante.
- **concurrent.futures**: Esta librería se utilizó para implementar la programación concurrente, permitiendo la extracción de datos de múltiples libros de forma paralela, lo que mejora la eficiencia del scraper.

#### 4.1.2. Estructura de la Clase **ScraperBooks**

La clase **ScraperBooks** encapsula toda la funcionalidad del scraper y se compone de varios métodos, cada uno con una función específica:

- **\_\_init\_\_(self, url)**: Método constructor que inicializa la clase con la URL de la página de libros en la que se hará el web scraping.
- **get\_books(self)**: Este método envía una solicitud a la URL especificada, recupera el contenido HTML de la página y utiliza **BeautifulSoup** para localizar y extraer las URLs de los libros listados. Devuelve una lista de URLs completas que apuntan a cada libro.

- **get\_book\_details(self, book\_url)**: A partir de la URL de un libro, este método realiza otra solicitud HTTP para obtener el contenido de la página del libro específico. Extrae detalles como el autor, el título y un resumen utilizando **BeautifulSoup**, y devuelve estos datos en forma de diccionario.
- **save\_to\_csv(self, data, filename='data/books.csv')**: Este método recibe una lista de diccionarios con los detalles de los libros y los guarda en un archivo CSV, lo que facilita su posterior análisis y uso.
- **run(self)**: Método principal que ejecuta el scraper utilizando los 3 métodos previos. Coordina el proceso de extracción de URLs de libros, almacena en una lista los detalles de cada libro utilizando múltiples hilos para optimizar el tiempo de ejecución y finalmente guarda los resultados en un archivo CSV.

#### 4.1.3. Ejecución del Scraper

El scraper se ejecuta instanciando un objeto de la clase **ScraperBooks** con la URL correspondiente a la lista de libros. Al invocar el método **run**, se inicia todo el proceso de scraping, desde la obtención de las URLs hasta el almacenamiento de los datos en el archivo CSV. Además, durante la ejecución, se mide el tiempo total del proceso, lo que permite evaluar la eficiencia del scraper.

#### 4.1.4. Resultados Obtenidos

El scraper logró extraer con éxito los datos de un conjunto significativo de libros, almacenando información crucial que sería utilizada posteriormente en el sistema de recomendación. Se generó un archivo **books.csv** que contiene columnas para el título, autor, resumen y enlace del libro, sirviendo como base para las recomendaciones personalizadas. Para su descarga remitirse al **Anexo**.

```
1 libros.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 4 columns):
#   Column      Non-Null Count  Dtype
---  -
0   Title       1000 non-null   object
1   Author      951 non-null    object
2   Summary     814 non-null    object
3   Link        1000 non-null   object
dtypes: object(4)
memory usage: 31.4+ KB
```

**Figura 4.1.1:** Dataset generado a partir del web scraping

## 4.2 Clasificación del Estado de Ánimo

El siguiente paso en el trabajo consiste en clasificar el estado de ánimo del usuario, un componente esencial del sistema, ya que permite personalizar las recomendaciones de actividades recreativas según las emociones detectadas. Para llevar a cabo esta tarea, se implementó un clasificador de emociones utilizando un modelo pre-entrenado en español.

### 4.2.1. Librerías

Para llevar a cabo la clasificación de emociones, se utilizaron las siguientes librerías de Python:

- **scikit-learn**: Esta librería se utilizó para proporcionar herramientas de aprendizaje automático, incluyendo modelos de clasificación como SVM, regresión logística y MLP.
- **transformers**: Se utilizó para implementar modelos pre entrenados de análisis de sentimientos y traducción.
- **mtranslate**: Esta librería se utilizó para traducir texto entre idiomas, lo cual resultó útil para procesar entradas en diferentes lenguas.
- **numpy**: Se utilizó para realizar operaciones numéricas y manipular datos en forma de matrices.
- **matplotlib**: Esta librería se utilizó para visualizar datos y resultados, incluyendo gráficas de comparación y pérdidas.
- **pandas**: Facilitó la manipulación y análisis de datos en estructuras de datos tipo DataFrame.
- **sentence-transformers**: Se utilizó para obtener embeddings de oraciones, que fueron necesarios para el procesamiento del texto.
- **imblearn**: Esta librería se utilizó para el sobremuestreo de clases en desequilibrio, ayudando a mejorar la precisión del modelo.
- **warnings**: Se utilizó para manejar y suprimir advertencias de código.
- **joblib**: Se empleó para guardar y cargar los modelos entrenados, facilitando su reutilización en el futuro sin necesidad de reentrenar desde cero.



### 4.2.2 Estructura de la Clase ClasificadorEmocion

La clase `ClasificadorEmocion` encapsula toda la funcionalidad relacionada con el análisis y clasificación de emociones. A continuación se detallan los principales métodos de la clase:

- `__init__(self)`: Método constructor que inicializa la clase con dos modelos preentrenados: uno para el análisis de sentimientos y otro para la clasificación de emociones. Además, se define un diccionario para mapear las etiquetas numéricas a las categorías emocionales.
- `predict_emotion(self, text)`: Este método recibe un texto y utiliza el modelo preentrenado para analizar el sentimiento del texto, devolviendo una clasificación de la emoción (Alegre, Melancólico, Ni fu ni fa o Desconocido) según el mapeo de las emociones.
- `predict_emotion_sentence_transformer(self, text)`: Este método primero traduce el texto a inglés y luego utiliza un modelo de clasificación de emociones entrenado en inglés para predecir la emoción del texto. La emoción predicha se mapea a las categorías "Alegre", "Melancólico" y "Ni fu ni fa".
- `detectar_emocion_entrenada(self, data, model, show_metrics=True)`: Utiliza un modelo de regresión logística entrenado con un conjunto de datos etiquetado para predecir la emoción de los textos. Se emplea un enfoque de validación cruzada K-Fold para optimizar el modelo y evaluar su rendimiento. Además, se calcula la precisión y la pérdida durante el entrenamiento y la prueba.
- `mostrar_grafica_perdidas(self, train_losses, test_losses)`: Este método muestra una gráfica que ilustra la evolución de la pérdida (log loss) durante el proceso de validación cruzada, lo que permite evaluar el comportamiento del modelo durante el entrenamiento y la prueba.
- `preprocess_texts(self, texts)`: Este método realiza el preprocesamiento necesario para convertir los textos en un formato adecuado para ser utilizados por el modelo de clasificación.
- `get_bert_embeddings(self, texts, model)`: Convierte los textos proporcionados en representaciones vectoriales utilizando el modelo de embeddings BERT.
- `guardar_modelo(self, modelo, nombre_archivo)`: Guarda el modelo entrenado en un archivo para su uso posterior, facilitando la reutilización del modelo sin necesidad de reentrenar cada vez.
- `cargar_modelo(self, nombre_archivo)`: Carga un modelo previamente guardado desde un archivo para hacer predicciones o continuar su entrenamiento.

- **predecir\_modelo\_entrenado(self, modelo, texto, model):**  
Realiza una predicción utilizando un modelo previamente entrenado. El texto ingresado es vectorizado y el modelo devuelve la emoción predicha.

#### 4.2.3. Entrenamiento del Clasificador

El proceso de entrenamiento del clasificador de emociones se llevó a cabo utilizando un conjunto de datos sintético con tres categorías emocionales: Alegre, Ni fu ni fa y Melancólico. El flujo de trabajo incluye los siguientes pasos:

1. **Preprocesamiento de Datos:** Los textos se limpiaron y preprocesaron para su vectorización mediante embeddings BERT. Estos embeddings transforman los textos en vectores numéricos que pueden ser utilizados por el modelo de clasificación.
2. **Balanceo de Clases:** Dado que las categorías pueden estar desbalanceadas, se aplicó un **RandomOverSampler** para igualar el número de ejemplos en cada clase, mejorando la precisión del modelo.
3. **Entrenamiento y Validación:** El modelo se entrenó utilizando un enfoque de validación cruzada K-Fold para evaluar su desempeño en diferentes subconjuntos de datos. Además, se calcularon las métricas de precisión y la pérdida, y se presentó una gráfica de las pérdidas en el conjunto de entrenamiento y prueba.

#### 4.2.4. Evaluación y Resultados

El clasificador de emociones fue evaluado utilizando el conjunto de datos de prueba, logrando un buen desempeño en la clasificación de las emociones predichas. Sin embargo, tras comparar el rendimiento de los modelos, se determinó que el método **predict\_emotion\_sentence\_transformer** supera al modelo de regresión logística entrenado en términos de precisión y capacidad de clasificación de emociones.

De todos modos, se optó por usar el modelo de regresión logística entrenado conseguido por nosotros como el modelo principal para las predicciones de emociones, debido a su pedido en el enunciado.

#### 4.2.5 Ejecución del Clasificador

El clasificador se ejecuta al instanciar un objeto de la clase **ClasificadorEmocion** y al invocar los métodos de predicción y entrenamiento. Primero, se cargan los datos del conjunto sintético generado previamente y se entrenan los modelos. Durante la ejecución, el clasificador realiza predicciones

sobre textos de ejemplo, clasificándolos en las categorías de emociones: "Alegre", "Melancólico" o "Ni fu ni fa".

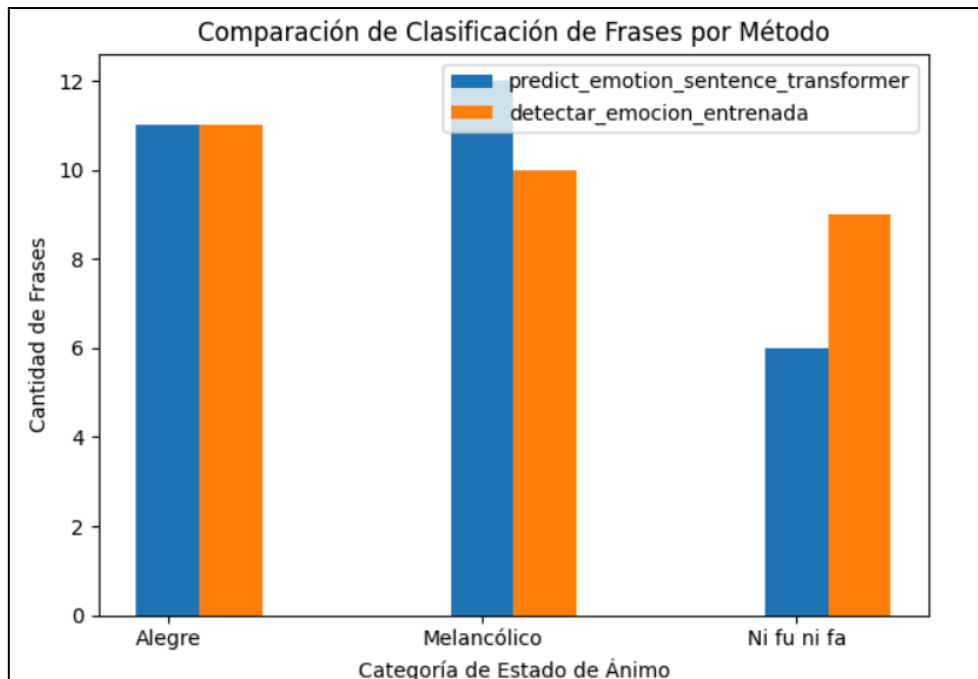
#### 4.2.6 Resultados Obtenidos

El clasificador logró categorizar correctamente las emociones de las frases de prueba, permitiendo al sistema realizar recomendaciones más personalizadas basadas en el estado de ánimo del usuario. Se generaron gráficas de comparación que ilustran la efectividad de los distintos métodos de clasificación implementados, proporcionando una evaluación visual de los resultados. En las figuras siguientes mostramos resultados obtenidos con el clasificador entrenado.

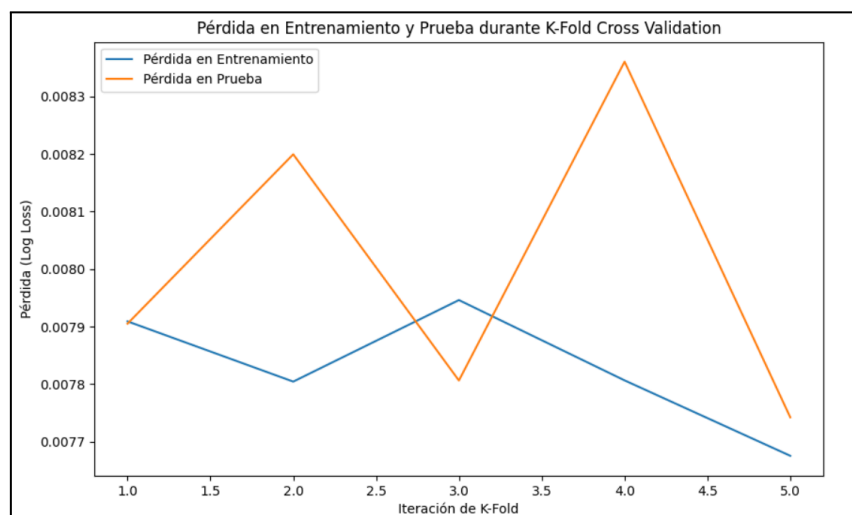
```
2 df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10002 entries, 0 to 10001
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype  
---  -
 0   Texto     10002 non-null  object 
 1   Emocion   10002 non-null  object 
dtypes: object(2)
memory usage: 156.4+ KB
```

**Figura 4.2.1:** Dataset sintético de emociones generado



**Figura 4.2.2:** Comparación de modelos del clasificador



**Figura 4.2.3:** Error de entrenamiento y prueba de modelo supervisado

Precisión Regresión Logística (final): 1.0				
Reporte de clasificación Regresión Logística:				
	precision	recall	f1-score	support
0	1.00	1.00	1.00	652
1	1.00	1.00	1.00	669
2	1.00	1.00	1.00	680
accuracy			1.00	2001
macro avg	1.00	1.00	1.00	2001
weighted avg	1.00	1.00	1.00	2001

**Figura 4.2.4:** Métricas de modelo supervisado

```

|-----|
Predicción de emociones:

Estoy muy feliz hoy --> Alegre
Fue un mal día --> Melancólico
Nada fuera de lo común --> Ni fu ni fa
|-----|

```

**Figura 4.2.5:** Ejemplo de salida de modelo supervisado

### 4.3 Recomendación de actividades:

La búsqueda de opciones constituye el núcleo del sistema, ya que permite generar recomendaciones personalizadas de actividades recreativas basadas en las preferencias y el estado emocional del usuario. Este proceso implica la integración de diferentes fuentes de datos, el uso del clasificador y el empleo de técnicas de procesamiento de lenguaje natural para ofrecer sugerencias relevantes.

#### 4.3.1. Librerías

Se utilizaron diversas librerías para implementar esta funcionalidad, incluyendo:

- **pandas:** Para la manipulación y análisis de datos, permitiendo la carga y procesamiento de los datasets en formato CSV.
- **faiss:** Un sistema de búsqueda eficiente que permite realizar búsquedas de similitud entre embeddings.
- **sentenceTransformers:** Para la generación de embeddings a partir de textos, facilitando la conversión de descripciones de actividades en vectores semánticos.

- **transformers**: Para el procesamiento de lenguaje natural, en particular, para resumir y traducir textos.
- **numpy** y **pickle**: Para operaciones matemáticas y la gestión de archivos de estado del modelo.

#### 4.3.2. Estructura de la Clase RecomendadorActividades

La clase `RecomendadorActividades` se encarga de gestionar la lógica de recomendación y se compone de los siguientes métodos:

- **`__init__(self, datasets, nombre_archivo='estado_recomendador.pkl')`**: Este método constructor inicializa la instancia de la clase, configurando los modelos de embeddings y de resumen. Verifica si existe un archivo de estado previo; si no, carga los datasets y genera los embeddings.
- **`_cargar_e_indexar_datasets(self, datasets)`**: Carga los datasets y genera los embeddings correspondientes. Cada dataset se lee utilizando `pandas`, y los textos se convierten en embeddings a través de `SentenceTransformers`. Estos embeddings se almacenan en un índice `FAISS` para búsquedas posteriores.
- **`_crear_indice(self, embeddings)`**: Este método crea un índice de similitud utilizando `FAISS`, permitiendo una búsqueda eficiente entre los embeddings generados.
- **`recomendar(self, preferencia, emocion)`**: Genera recomendaciones basadas en la preferencia del usuario y su estado emocional. Convierte la preferencia a inglés, genera su embedding, y busca en el índice `FAISS` para encontrar las tres actividades más similares.
- **`_formatear_recomendacion(self, item, emocion)`**: Formatea la recomendación ajustando el mensaje según la emoción del usuario, incluyendo el título, un resumen traducido y la fuente del contenido.
- **`_resumir_y_traducir(self, texto)`**: Resume un texto si supera las 50 palabras y lo traduce al español, mejorando la legibilidad de la información proporcionada al usuario.
- **`guardar_estado(self, nombre_archivo)`**: Guarda el índice `FAISS`, los embeddings y el estado del modelo en un archivo para su uso posterior.
- **`cargar_estado(self, nombre_archivo)`**: Carga el estado del modelo desde un archivo, restaurando el índice `FAISS`, los embeddings y los modelos utilizados.

#### 4.3.3. Ejecución del Recomendador de Actividades

Para probar la funcionalidad del recomendador, se creó un objeto de la clase `RecomendadorActividades`, que inicializa los datasets y los modelos. Luego, se solicita al usuario que describa su día en una frase, que es analizada por el

**ClasificadorEmocion** para detectar el estado emocional. A continuación, el usuario especifica una preferencia temática, y se generan las recomendaciones.

#### 4.3.4. Resultados Obtenidos

El sistema devuelve tres recomendaciones personalizadas, acompañadas de un mensaje que refleja el estado emocional del usuario. Por ejemplo, si se detecta una emoción positiva, el mensaje será alentador, mientras que si la emoción es negativa, el mensaje ofrecerá sugerencias para mejorar el estado de ánimo. Estas recomendaciones se presentan de forma clara y atractiva, facilitando al usuario la exploración de nuevas actividades recreativas.

```
¿Qué temática te gustaría explorar? Una historia de amor en la selva
Estado emocional detectado: Melancólico
Recomendaciones:

-----
Interesante opción: La Comedia Humana - Volumen 02 (Fuente: Libro)
"La Comédie Humaine - Volumen 02" de Honoré de Balzac es una colección de narra
-----
Interesante opción: Cuentos de Amor de Locura y de Muerte (Fuente: Libro)
"Cuentos de amor de locura y de muerte" de Horacio Quiroga es una colección de
-----
Interesante opción: El romance de la rosa - Volumen I (Fuente: Libro)
"Le roman de la rose - Tome I" de Guillaume de Lorris y Jean de Meung es un poe
-----
```

**Figura 4.3.1:** Ejemplo de salida del recomendador

**Observacion sobre uso de NER:** En el contexto de la clase RecomendadorActividades, el uso de NER (modelo Gliner) no es necesario porque los embeddings de SentenceTransformer ya capturan de manera integral el contexto y la semántica del texto. Estos embeddings permiten comparar la similitud semántica entre las preferencias del usuario y el contenido, proporcionando una representación global y rica del significado del texto.

Nuestro enfoque basado en embeddings es mejor que el uso de NER en este caso porque los embeddings no solo identifican palabras clave, sino que también capturan relaciones complejas y matices semánticos que son esenciales para recomendaciones precisas. Mientras que NER se enfoca únicamente en extraer entidades específicas (nombres, lugares, etc.), los embeddings permiten una comparación más profunda y contextual del significado de los textos completos, lo cual es crucial para encontrar recomendaciones relevantes y alineadas con las preferencias del usuario. Además, el uso de NER agregaría una capa de complejidad y procesamiento adicional que no aporta un valor proporcional al costo computacional, ya que no mejora la calidad de la similitud semántica.

## 4.4 Aplicación:

El objetivo principal de la aplicación es permitir que los usuarios ingresen su estado de ánimo y preferencias temáticas, para luego recibir recomendaciones personalizadas. La implementación de esta aplicación se realizó utilizando la librería `ipywidgets` para la creación de una interfaz gráfica interactiva en Jupyter Notebook.

### 4.4.1. Librerías

Para desarrollar la aplicación se emplearon las siguientes librerías de Python:

- **ipywidgets**: Esta librería proporciona herramientas para construir interfaces gráficas interactivas en entornos Jupyter. Permite crear widgets como botones, cuadros de texto y salidas de texto.
- **IPython.display**: Utilizada para mostrar HTML y limpiar la salida en la interfaz de usuario.
- **pandas**: Esta librería se utiliza para manejar y procesar datos, en este caso, para cargar el conjunto de datos sintético de emociones.



#### 4.4.2. Estructura de la Aplicación

La aplicación se organiza en varios componentes, cada uno responsable de una funcionalidad específica:

**Nota:** La interfaz grafica de la aplicación se puede ver en la sección 4.4.4 de resultados, figura ?

##### Clasificación:

- **Entrada de Frase de Estado de Ánimo:** Un campo de texto donde el usuario puede escribir una frase que describa su día. Este campo está etiquetado como "Estado de Ánimo" y tiene un texto de marcador de posición que indica al usuario que debe describir su día.
- **Botón para Detectar Ánimo:** Un botón que, al ser clicado, activa el proceso de detección de emociones.
- **Salida de Emoción Detectada:** Un área de salida donde se muestra la emoción detectada a partir de la frase ingresada por el usuario.

##### Recomendación:

- **Entrada de Preferencia de Temática:** Un segundo campo de texto donde el usuario puede especificar el tipo de actividad que le gustaría explorar. Este campo se habilita automáticamente una vez que se ha detectado una emoción.
- **Botón para Recomendar:** Otro botón que permite al usuario solicitar recomendaciones basadas en su emoción y preferencia. Este botón también está inicialmente deshabilitado hasta que el usuario ingresa su preferencia.
- **Salida de Recomendaciones:** Un área donde se despliegan las recomendaciones de actividades recreativas basadas en la emoción detectada y la preferencia del usuario.

#### 4.3.3 Funciones

Las funciones callback son esenciales para gestionar la interactividad de la aplicación, ya que responden a las acciones del usuario. En este caso, se implementan dos funciones callback: una para detectar el estado de ánimo y otra para recomendar actividades.

1. **Función `detectar_animo_callback`:** Esta función se activa cuando el usuario hace clic en el botón "Detectar Ánimo". Realiza las siguientes acciones:
  - Limpia la salida anterior en la interfaz.
  - Recupera la frase ingresada por el usuario y genera los embeddings utilizando el modelo BERT.
  - Utiliza el clasificador entrenado para predecir la emoción correspondiente a la frase ingresada.
  - Muestra la emoción detectada en la interfaz.
  - Habilita el campo de entrada para la preferencia del usuario y el botón de recomendar, permitiendo que el usuario continúe con el proceso.
2. **Función `recomendar_callback`:** Esta función se activa cuando el usuario hace clic en el botón "Recomendar". Realiza las siguientes acciones:
  - Limpia la salida anterior en la interfaz.
  - Recupera la preferencia ingresada por el usuario.
  - Utiliza el recomendador para obtener una lista de actividades basadas en la emoción detectada y la preferencia del usuario.
  - Muestra la emoción detectada, la preferencia del usuario y las recomendaciones en la interfaz. Si no se encuentran recomendaciones, informa al usuario de esta situación.

#### 4.4.3. Ejecución de la Aplicación

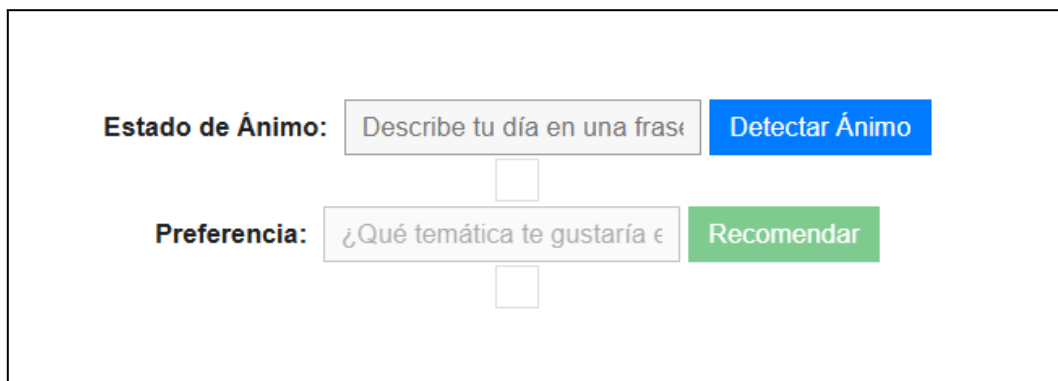
A través de los widgets, los usuarios pueden interactuar con la aplicación de la siguiente manera:

1. **Descripción del Estado de Ánimo:** El usuario ingresa una frase que describe su estado emocional en un campo de texto dedicado.
2. **Detección del Estado de Ánimo:** Al hacer clic en el botón "Detectar Ánimo", se activa la función `detectar_animo_callback`, que procesa la frase ingresada. Esta función genera los embeddings utilizando el modelo BERT, clasifica la emoción mediante el clasificador y muestra la emoción detectada en la interfaz.
3. **Preferencia temática:** El usuario especifica una temática de interés en otro cuadro de texto habilitado tras la detección del estado de ánimo.
4. **Recomendación de Actividades:** Finalmente, al hacer clic en el botón "Recomendar", se activa la función `recomendar_callback`, que procesa la preferencia ingresada y la emoción previamente detectada. Esta función genera y muestra las recomendaciones personalizadas en la interfaz,

permitiendo al usuario explorar actividades alineadas con su estado emocional y preferencias.

#### 4.4.4. Resultados Obtenidos

La aplicación interactiva permite a los usuarios recibir retroalimentación inmediata sobre su estado emocional y obtener recomendaciones de actividades de manera intuitiva. La integración de la lógica del clasificador y el recomendador asegura que las sugerencias sean relevantes y alineadas con las emociones del usuario. Esto no solo mejora la experiencia del usuario, sino que también fomenta un uso efectivo del sistema de recomendaciones en función del estado de ánimo.



The image shows a user interface for a recommendation application. It consists of two main sections, each with a label, an input field, and a button. The first section is labeled "Estado de Ánimo:" and contains a text input field with the placeholder "Describe tu día en una frase" and a blue button labeled "Detectar Ánimo". Below the input field is a small square checkbox. The second section is labeled "Preferencia:" and contains a text input field with the placeholder "¿Qué temática te gustaría €" and a green button labeled "Recomendar". Below this input field is another small square checkbox. The entire interface is enclosed in a thin black rectangular border.

**Figura 4.4.1:** Interfaz de usuario de la aplicación de recomendaciones

## **5. Conclusión**

El desarrollo de un sistema de recomendaciones de actividades recreativas basado en el estado de ánimo del usuario ha demostrado ser un proceso efectivo e integrador, combinando técnicas de Procesamiento de Lenguaje Natural y diversas herramientas de software.

A través de un scraper, se recopiló una base de datos sólida de libros, que, junto con los datasets de juegos de mesa y películas, respaldan las recomendaciones personalizadas. Se implementó un clasificador de emociones a través de un modelo de clasificación supervisado, lo que permitió detectar el estado de ánimo del usuario con precisión. El recomendador de actividades utilizó embeddings y búsqueda de similitud junto con el clasificador para ofrecer sugerencias relevantes, mientras que la aplicación construida facilitó la interacción del usuario. En conjunto, este sistema enriquece la experiencia del usuario al ofrecer opciones de entretenimiento personalizadas.

Este proyecto no solo demuestra la efectividad de personalizar las recomendaciones según el estado emocional del usuario, sino que también se obtuvieron buenos

resultados en términos de precisión y relevancia de las sugerencias, lo que valida el enfoque adoptado. Además, abre la puerta a futuras mejoras, como la incorporación de más fuentes de datos y la implementación de modelos de aprendizaje profundo.

## 6. Anexos

- **Código Fuente:** En este enlace se puede acceder al código fuente desarrollado para el sistema de recomendaciones en Google Colab.

[https://colab.research.google.com/drive/1p2UqctWDKZ1L52wBDxS-ljzUjkifA3mh#scrollTo=Eh9sn\\_tWn-Pd](https://colab.research.google.com/drive/1p2UqctWDKZ1L52wBDxS-ljzUjkifA3mh#scrollTo=Eh9sn_tWn-Pd)

- **Datasets:** Este enlace proporciona acceso a los conjuntos de datos utilizados en el proyecto, incluyendo juegos de mesa, películas y libros.

<https://drive.google.com/drive/folders/1u-q-c8m-J6lGvfm5K3XvZCHhGlvvFHeh?usp=sharing>