



Universidad Nacional de Rosario  
Facultad de Ciencias Exactas, Ingeniería y Agrimensura

## **IA4.2 Procesamiento del Lenguaje Natural**

### **Trabajo Práctico N°1**

Tecnicatura Universitaria en Inteligencia Artificial

Integrantes:

Sergio Castells, C-7334/2

Gonzalo Asad, A-4595/1

Docentes:

Juan Pablo Manson

Alan Geary

Fecha: 06/10/2024

1.	RESUMEN	4
2.	INTRODUCCIÓN	5
3.	METODOLOGÍA	6
3.1	Entorno de trabajo	6
3.2	Fuentes de datos	7
3.3	Métodos y técnicas	8
3.3.1	Web-scraping	8
3.3.2	Embedding	8
3.3.3	Similitud del coseno	9
3.3.4	Regresión logística	9
3.3.5	Traducción	9
3.3.6	Reconocimiento de entidades nombradas (NER)	9
3.4	Pasos para la resolución	9
4.	DESARROLLO E IMPLEMENTACIÓN	11
4.1	Hito 1: Generación de base de datos de estados de ánimo	11
4.2	Hito 2: Generación de base de datos de libros	11
4.2.1	La función getValidation	12
4.2.2	Extracción del título y autor	13
4.2.3	Extracción del género	14
4.2.4	Extracción del idioma	14
4.2.5	Extracción del resumen	14
4.2.6	Resultado final	15
4.3	Hito 3: Clasificación del estado de ánimo	15
4.3.1	La función emotionClassifierModel	15
4.3.2	La función emotionClassifier	16
4.4	Hito 4: Sistema de recomendaciones	16
4.4.1	La función embeddingsGenerator	16
4.4.2	La función recommendations	16
4.4.3	La función labelsDetector	17
4.4.4	La función recommender	18
4.5	Hito 5: Programa principal (Main)	20
4.5.1	La función getDataFrame	20

4.5.2	La función onRecommendButtonClicked	20
4.5.3	Main	21
<b>5.</b>	<b>RESULTADOS</b>	<b>26</b>
5.1	Base de datos de estados de ánimo	26
5.2	Base de datos de libros	26
5.3	Clasificación del estado de ánimo	26
5.4	Sistema de recomendaciones	28
<b>6.</b>	<b>CONCLUSIONES</b>	<b>32</b>

## 1. RESUMEN

---

El presente trabajo consistió en el desarrollo de un programa de procesamiento del lenguaje natural (PLN) que, según el estado de ánimo del usuario, recomiende entre ver una película, jugar un juego de mesa, leer un libro o varias opciones para cada caso. Para ello, fue necesario construir un clasificador que categorizara el estado de ánimo del usuario y un modelo que pudiera sugerir un conjunto de recomendaciones basada en una frase de preferencia ingresada por el mismo.

Se observará que el algoritmo es capaz de clasificar correctamente la mayoría de los estados de ánimo, sin embargo, suele recomendar contenido que no siempre se encuentra del todo alineado a la frase ingresada por el usuario debido a la limitación que ofrecen las técnicas empleadas para desarrollarlo.

## 2. INTRODUCCIÓN

---

Este trabajo se desarrolló pensando en brindarle un servicio a una persona que, en un mes, se tomaría 15 días de vacaciones en la playa pero que desafortunadamente, las condiciones climáticas serían desfavorables para llevar a cabo las actividades previstas. Por ello, para entretener al individuo, se propuso una solución que facilitara su recreación durante el transcurso del día en función de su estado de ánimo.

El objetivo principal de este trabajo fue el de desarrollar un algoritmo capaz de comprender con exactitud el estado de ánimo de una persona y clasificarlo dentro de tres categorías positivo, neutro o negativo; para su uso posterior en un sistema de recomendaciones. Este último debía poder interpretar la solicitud del usuario y encontrar similitudes dentro de tres bases de datos (películas, libros y juegos de mesa), considerando el estado de ánimo en su decisión final.

El algoritmo se consideró satisfactorio luego de haber superado una serie de pruebas exhaustivas, basadas en aspectos objetivos y subjetivos de sus desarrolladores.

El presente informe comienza abordando la metodología empleada a lo largo del trabajo, continuando con los detalles sobre su desarrollo e implementación. Luego se muestran resultados de distintas evaluaciones y posteriormente se exponen las conclusiones alcanzadas. Por último, se incluyen referencias y anexos con los conjuntos de datos utilizados.

### 3. METODOLOGÍA

#### 3.1 ENTORNO DE TRABAJO

Para el desarrollo, se optó por utilizar el lenguaje de programación Python debido a su amplio uso en la carrera y a la abundante documentación disponible en la web. El código se escribió en el entorno de desarrollo integrado Google Colab, que ofrece herramientas no solo para la escritura de código, sino también para la depuración de problemas, ejecutándose en un servidor virtual en la nube con recursos de hardware proporcionados por Google.

En el espacio de trabajo se debieron instalar las siguientes librerías:

- *Gdown*: para cargar al entorno de trabajo datasets almacenados en un drive en la nube.
- *Beautifulsoup4*: para realizar trabajos automáticos de web-scraping en la construcción de uno de los datasets.
- *Deep\_translator*: que brinda funciones de traducción de idiomas.
- *Transformers*: que brinda funciones para generar tokens además de embeddings de palabras y oraciones usando modelos pre-entrenados.
- *Gliner*: para el reconocimiento de entidades nombradas (NER).

También, se hizo uso de las siguientes librerías que no necesitaron instalación adicional:

- *Pandas*: conocida librería para la manipulación de DataFrames.
- *Numpy*: librería muy difundida que permite operaciones matemáticas complejas sobre matrices.
- *Requests*: utilizada para hacer solicitudes http durante el proceso de web-scraping.
- *Time*: para medir tiempos de ejecución y generar tiempos de espera.
- *Re*: que permite trabajar con expresiones regulares.
- *Warnings*: para filtrar advertencias innecesarias que ensucian los mensajes del algoritmo.
- *Torch*: para hacer uso de sus muchos modelos.
- *Sklearn*: la librería por excelencia dentro del mundo del machine learning por su difusión y buena documentación, para hacer uso de sus funciones de cálculo de métricas, modelado con regresiones logísticas, splitters, clasificadores y vectorizadores.
- *Nltk*: para hacer uso de su reconocimiento de palabras de parada.
- *Ipywidgets*: que brinda una interfaz gráfica amigable con el usuario.
- *IPython.display*: para imprimir en pantalla objetos *html*.

### 3.2 FUENTES DE DATOS

Se utilizaron tres bases de datos de distintos orígenes que están disponibles al público:

- [Bgg\\_database.csv](#): base de datos con los mil juegos de mesa mejor puntuados según el sitio web BoardGameGeek. De cada uno de los juegos se tienen los siguientes datos:
  - Rank: posición dentro del ranking
  - Game\_name: nombre del juego
  - Game\_ref: enlace a los detalles del juego dentro del sitio de BGG
  - Geek\_rating: puntaje dado por reseñadores del sitio BGG
  - Avg\_rating: puntaje promedio dado por usuarios del sitio BGG
  - Num\_voters: número de votantes para la puntuación pública del juego
  - Description: contexto en el cual se desarrolla el juego junto con una breve descripción de sus reglas
  - Yearpublished: año de publicación del juego
  - Minplayers: mínimo número de jugadores
  - Maxplayers: máximo número de jugadores
  - Minplaytime: cantidad mínima de minutos de juego estimada
  - Maxplaytime: cantidad máxima de minutos de juego estimada
  - Minage: edad mínima recomendada para jugarlo
  - Avgweight: peso promedio en libras
  - Best\_num\_players: sugerencia sobre el número óptimo de jugadores basado en reseñas
  - Designers: creador o co-creadores del juego
  - Mechanics: etiquetas con las mecánicas principales del juego
  - Categories: etiquetas con las categorías principales del juego
- [IMDB-Movie-Data.csv](#): base de datos con las mil películas más populares entre los años 2008 y 2018 según el sitio web IMDb. De cada una de las películas se tienen los siguientes datos:
  - Rank: posición dentro del ranking
  - Title: nombre de la película
  - Genre: género o géneros de la película
  - Description: breve sinopsis de la trama
  - Director: persona que dirigió la película
  - Actors: actores principales dentro de la película
  - Year: año de lanzamiento de la película
  - Runtime: duración de la película en minutos
  - Rating: puntaje promedio de la película según usuarios de IMDb
  - Votes: número de votantes para la puntuación pública de la película
  - Revenue: ganancias que obtuvo la película medida en millones de dólares
  - Metascore: puntaje brindado por el sitio web Metascore

- [Libros del Proyecto Gutenberg](#): base de datos con los mil libros más populares según el sitio web Proyecto Gutenberg. Se debió realizar un trabajo de web-scraping para conformar el dataset (incluido en los anexos del trabajo). De cada uno de los libros se tienen los siguientes datos:
  - Title: título del libro
  - Author: autor del libro
  - Subjects: género literario del libro
  - Language: idioma original en el que se publicó el libro
  - Summary: breve sinopsis de la historia narrada o descripción del contenido del libro

Adicionalmente, se debió generar una base de datos que tuviera diferentes palabras o frases que describieran estados de ánimo junto con etiquetas clasificatorias, para entrenar un modelo que sirviera como clasificador de estados de ánimo. Dicho dataset contiene únicamente dos variables:

- Label: etiqueta clasificatoria del estado de ánimo, puede asumir tres valores: positivo, neutro y negativo
- Estado\_animo: palabras o frases que describen un estado de ánimo. Los datos vienen del español común y del lunfardo

Esta base de datos fue generada con la ayuda de ChatGPT, que tras muchas iteraciones de consiguió brindar una base de datos de más de 250 elementos.

### 3.3 MÉTODOS Y TÉCNICAS

Si bien el detalle de la implementación se describe en la unidad siguiente, aquí describimos los métodos y técnicas de PLN principales utilizados por el algoritmo.

#### 3.3.1 Web-scraping

Es un conjunto de técnicas para la extracción de datos contenidos en el código *html* de una página web. Apoyados en la librería de Python *Beautiful Soup*, se navegó el portal de *Proyecto Gutenberg* para extraer el código *html* de su página con el contenido de los mil libros más famosos de la historia. Luego, se utilizaron las funciones de *find* y *select* para filtrar contenido dentro del código, extrayendo las porciones de texto que serían de interés a la hora de armar la base de datos. Para afinar detalles finales en las cadenas de texto, usaron expresiones regulares junto con la función *split*, limpiando pedazos de texto que podrían dificultar la interpretación de los datos por parte del modelo de lenguaje.

#### 3.3.2 Embedding

Consiste en generar representaciones vectoriales dentro de un espacio multidimensional de frases o palabras. Haciendo uso de la librería *Transformers*, puntualmente de su función *Sentence\_Transformers*, se obtuvieron incrustaciones (embeddings) de palabras y frases, usando como base un modelo pre-entrenado, en distintos puntos del algoritmo:



- Para la clasificación de estados de ánimo, se calcularon las incrustaciones de los estados de ánimo que luego serían utilizadas en un modelo de regresión logística.
- Para el pedido del usuario, se calcularon las incrustaciones del prompt y de los distintos elementos de las bases de datos que serían utilizados para comparar con la solicitud de la persona.

### 3.3.3 Similitud del coseno

Técnica que evalúa la cercanía de vectores dentro de un espacio multivariable. Ésta fue utilizada para encontrar similitudes entre la incrustación del prompt ingresado por el usuario y las incrustaciones de los datos más relevantes de las tres bases de datos. Dichas similitudes serían luego ordenadas dentro de un ranking para posteriormente obtener los mejores resultados (los más próximos al prompt) y presentarlos al usuario.

### 3.3.4 Regresión logística

Modelo que permite clasificar dentro de distintas categorías a los datos, habiendo sido entrenado previamente con un dataset etiquetado (aprendizaje supervisado). Se utilizó para generar el modelo que clasifica los estados de ánimo del usuario dentro de tres categorías: positivo, neutro y negativo.

### 3.3.5 Traducción

Consiste en traducir palabras o frases de un idioma a otro seleccionado por el usuario o desarrollador. Dentro del algoritmo, se utiliza para traducir al inglés el prompt del usuario previo a generar su incrustación, ya que las bases de datos de películas, libros y juegos de mesa se encuentran en ese idioma. De no traducir el prompt, se podrían obtener incrustaciones incomparables con la realidad del problema.

### 3.3.6 Reconocimiento de entidades nombradas (NER)

Es un etiquetado inteligente de elementos dentro de una frase o conjunto de oraciones. Dentro del algoritmo, se utiliza para etiquetar elementos del prompt del usuario y así orientar mejor la búsqueda y recomendación de contenido para el usuario.

## 3.4 PASOS PARA LA RESOLUCIÓN

El trabajo fue abordado por etapas o hitos. Se dividió el problema en subproblemas a resolver con la generación de distintas funciones para atacarlos.

- *Hito 1:* generación de base de datos de estados de ánimo utilizando generadores de texto basados en NLP.
- *Hito 2:* web-scraping del sitio web de *Proyecto Gutenberg* para la generación de base de datos de libros.
- *Hito 3:* desarrollo de una función que clasifique correctamente emociones.
- *Hito 4:* desarrollo de una función que tome el prompt de un usuario y obtenga los resultados más cercanos de las distintas bases de datos, de acuerdo al estado de ánimo de la persona.

- *Hito 5*: desarrollo del programa principal e interfaz del usuario.

## 4. DESARROLLO E IMPLEMENTACIÓN

Tal como fue descripto en la sección anterior, la resolución del algoritmo fue abordada en muchas etapas o hitos, descomponiendo el problema en varios subproblemas. Durante esta sección, se describe la resolución de cada uno de ellos.

### 4.1 HITO 1: GENERACIÓN DE BASE DE DATOS DE ESTADOS DE ÁNIMO

Se utilizó el chatbot de inteligencia artificial *ChatGPT* por su fuerte potencial de procesamiento dentro del mercado de procesadores del lenguaje natural y su capacidad de generar conjuntos de datos como tablas o incluso archivos *csv*. Fue necesario realizar muchas iteraciones de generación de texto hasta dar con un resultado conforme a las expectativas. En un principio, se intentó generar un dataset que clasificara tan solo palabras que representaran estados de ánimo en cinco categorías distintas: positivo-fuerte, positivo-débil, neutro, negativo-débil y negativo-fuerte. Aquí, el chatbot no sólo consiguió entregar datasets demasiado pequeños, sino que las clasificaciones no siempre fueron correctas.

Para solucionar el primer problema, se le pidió al chatbot generar más palabras. Sin embargo, no consiguió encontrar palabras nuevas que representaran estados de ánimo, por el contrario, comenzó a repetirlas dentro del dataset. Esto último dio a entender que con palabras no sería suficiente para clasificar estados de ánimo, por lo que se le pidió al chatbot agregar frases pequeñas que los representaran. Esto amplió en una gran magnitud el dataset brindando mejores resultados de clasificación más adelante. Finalmente, considerando que el usuario principal sería argentino, se le solicitó al chatbot agregar palabras y frases del lunfardo clasificadas, ampliando aún más el dataset.

Para solventar el segundo problema, se apuntó a reducir la cantidad de categorías clasificatorias a tan solo tres: positivo, neutro y negativo, lo que ayudó a obtener clasificaciones de palabras mucho más precisas.

El chatbot generó múltiples archivos *CSV*, los cuales fueron consolidados en un único archivo para simplificar su uso antes de cargarlos en el entorno de trabajo.

### 4.2 HITO 2: GENERACIÓN DE BASE DE DATOS DE LIBROS

El sitio web del *Proyecto Gutenberg* contiene un repositorio de datos de libros de toda la historia global y los clasifica en diferentes rankings. Para este trabajo, se debió conformar un dataset basado en el ranking de los mil libros agrupados en “Top 1000 EBooks yesterday” según los clasifica este sitio web, sin embargo, el mismo no provee una versión descargable del mencionado listado. Por este motivo, fue necesario realizar un trabajo de web-scraping para extraer los datos y conformar el dataset en cuestión.

El sitio web incluye una página de ranking de libros que muestra los títulos y autores. Cada libro tiene un hipervínculo que dirige a una página dedicada con información detallada sobre la obra:

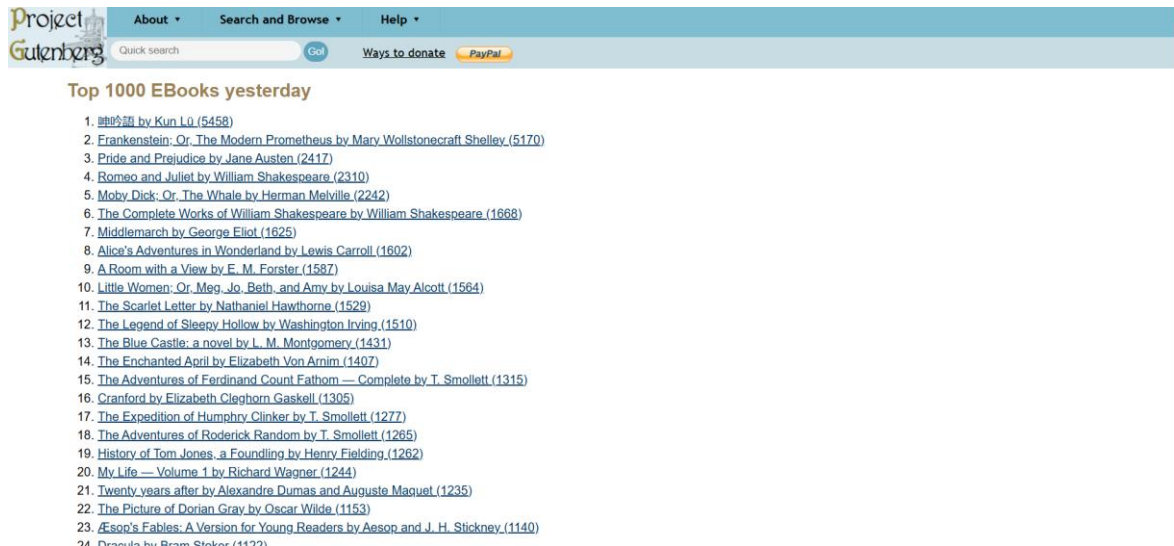


Figura 1: Ranking de libros más populares de la historia global.

Por lo que, en una primera instancia, fue necesario extraer del código *html* del sitio las porciones que contenían exclusivamente datos de los libros del ranking en cuestión, con la dificultad adicional que en el mismo sitio se encuentran muchos rankings uno bajo el otro. El ranking de interés fue el primero en el listado.

## 4.2.1 Función `getValidation`

Se generó una función llamada `getValidation` que valida la petición GET a una URL y retorna la respuesta si el caso es exitoso. Caso contrario, implementa reintentos con backoff exponenciales para manejar los errores de timeout. La función recibe como argumentos:

- `url(str)`: URL a la cual se le quiere realizar la petición.
- `retries(int)`: Cantidad de reintentos de reconexión.
- `Backoff_factor(float)`: Factor que afecta el tiempo de espera antes de reintentar una reconexión.

La función `getValidation` realiza un *try-except* extrayendo el estado de la transacción durante el pedido GET. En caso de recibir una excepción, devuelve el código de error y realiza la cantidad de reintentos especificados como argumento de la función, aumentando el tiempo de espera entre reintentos de acuerdo al factor también ingresado como argumento, para no sobrecargar de solicitudes a la página. De superarse la cantidad de reintentos devuelve un mensaje de error de solicitud. Si la solicitud es exitosa, retorna el mensaje *html* completo recibido como respuesta.

## 4.2.2 Extracción del título y autor

Haciendo uso de la función arriba descrita, se consulta al sitio web de *Proyecto Gutenberg* obteniendo como respuesta el código *html* completo de la página. Dentro de su estructura se encuentran varias listas ordenadas `<ol>` cada una de las cuales agrupa mil ítems `<li>`. El ranking de interés se encuentra contenido en la primera lista ordenada, por lo tanto, usando la función *find* de *BeautifulSoup* se extrae el listado completo y se lo almacena en una estructura de tipo lista.

Cada ítem de la lista ordenada tiene el siguiente formato:

```
<li><a href="/ebooks/84">Frankenstein; Or, The Modern Prometheus by
Mary Wollstonecraft Shelley (8508)</a></li>
```

por ende, de allí se pueden obtener el nombre del libro y el autor (cadena de texto con separador "by"), además del enlace a la página web donde hay más información del mismo.

Para filtrar el nombre del libro y el autor se utiliza la función *get\_text* para obtener la cadena de texto y luego se aplica la función *split* utilizando la palabra "by" como separador. En el caso de que dicha palabra no se encuentre contenida en el texto se considera que el autor es desconocido ("Unknow"). Además, el nombre del autor viene acompañado por un número entre paréntesis. Eso se elimina implementando una función con una expresión regular (RegEx).

La extensión al enlace de la web asociada al libro se obtiene aplicando la función *find*

A continuación, se conforma la URL con los datos del libro concatenando la URL base con el dato extraído del elemento `<href>` y se le envía una solicitud mediante la función *getValidation* obteniendo una respuesta. Las páginas web de los libros en particular contienen mucha información de utilidad para agregar al dataset.

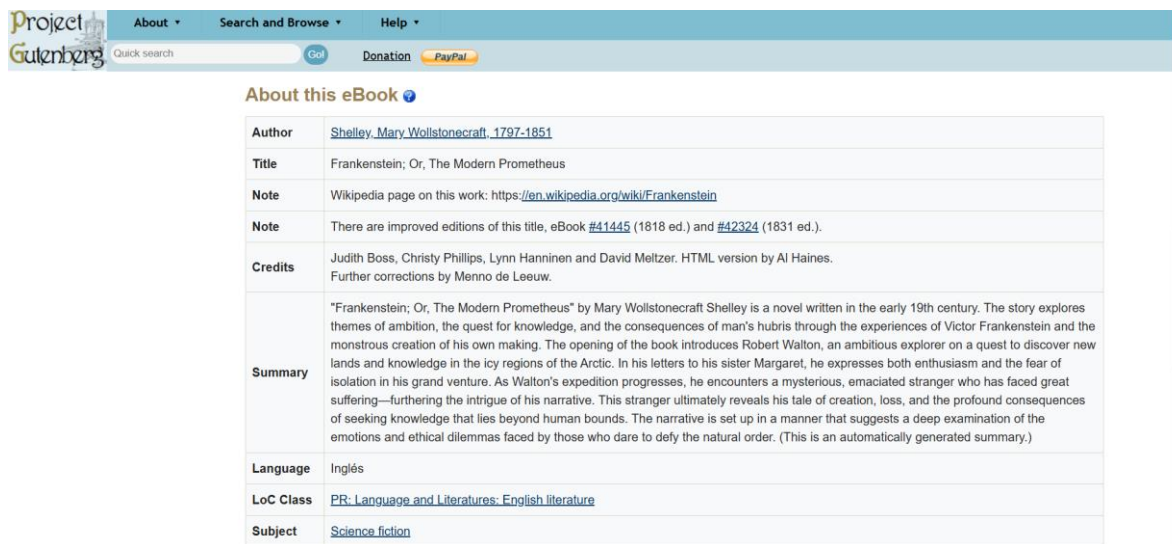


Figura 2: Página web de ejemplo de un libro.

Inspeccionando el documento *html* se observa que la información acerca del libro se encuentra en una estructura de tipo tabla donde por cada fila `<tr>` existe una columna a modo de encabezado `<th>` y otra columna donde se almacenan los datos `<td>`.

## 4.2.3 Extracción del género

La información de género literario se encuentra en una estructura como la siguiente:

```
<tr>
  <th>Subject</th>
  <td property="dcterms:subject" datatype="dcterms:LCSH">
    <a class="block" href="/ebooks/subject/36"> Science fiction
  </a>
  </td>
</tr>
```

Sin embargo, hay libros que tienen más de un género y libros que no tienen ninguno. Se extrae entonces el texto de la etiqueta `<a>` correspondiente a la primera fila que contiene el encabezado "Subject" y que en el campo de datos posea al atributo "Property" con el valor "dcterms:subject". En caso de no haberse detectado ninguno, se completa con *None*. Estas validaciones se agregan porque, caso contrario, se genera un error al no encontrar elementos.

## 4.2.4 Extracción del idioma

La información del idioma se encuentra en una estructura como la siguiente:

```
<tr property="dcterms:language" datatype="dcterms:RFC4646"
itemprop="inLanguage" content="en">
  <th>Language</th>
  <td>Inglés</td>
</tr>
```

Se selecciona la primera celda de datos `<td>` correspondiente al encabezado "Language" utilizando la función *select*. Se agrega también una validación, caso contrario, se genera un error al no encontrar elementos.

## 4.2.5 Extracción del resumen

La información del resumen se encuentra en una estructura como la siguiente:

```
<tr>
  <th>Summary</th>
  <td>
    "Frankenstein; Or, The Modern Prometheus" by Mary Wollstonecraft
    Shelley is a novel written in the early 19th century.
  </td>
</tr>
```

The story explores themes of ambition, the quest for knowledge, and the consequences of man's hubris through the experiences

of Victor Frankenstein and the monstrous creation of his own making...

</td>

</tr>

Se selecciona la primera celda de datos <td> correspondiente al encabezado "Summary". Nuevamente, se agregó una etapa de validación para evitar la generación de errores al no encontrar elementos.

Al final de cada resumen se encuentra el texto "(This is an automatically generated summary.)", el cual es eliminado.

## 4.2.6 Resultado final

La información del libro es almacenada en un diccionario que contiene los datos de: título, autor, género, idioma y resumen. El diccionario es luego agregado a una lista de libros que se transforma a un DataFrame y posteriormente se exporta como un archivo csv.

## 4.3 HITO 3: CLASIFICACIÓN DEL ESTADO DE ÁNIMO

Al ejecutar el algoritmo, se solicita al usuario que indique su estado de ánimo, información que se utilizará posteriormente para decidir el tipo de contenido recomendado. El estado de ánimo del usuario se clasifica en tres categorías: positivo, neutro y negativo, mediante un modelo de regresión logística entrenado con un conjunto de datos de estados de ánimo recopilados a través de un chatbot. A continuación, se describen las funciones principales utilizadas para la clasificación de estos estados de ánimo.

### 4.3.1 Función `emotionClassifierModel`

Esta función genera y retorna un modelo de regresión logística para la clasificación de estados de ánimo. Recibe como argumentos:

- `df(pd.DataFrame)`: Base de datos de estados de ánimo clasificados.
- `production(bool)`: Si es Falsa, se calculan y muestran las métricas de prueba.

La función `emotionClassifierModel` carga un modelo pre-entrenado, específicamente el modelo "distiluse-base-multilingual-cased-v1" usando *SentenceTransformer*. A continuación, recorre la base de datos de estados de ánimo y la almacena en una lista de tuplas para su procesamiento, separando las variables de entrada (estado de ánimo) y de salida (etiqueta). A continuación, divide el dataset en datos de entrenamiento y de prueba, en relación 80-20 y calcula las incrustaciones de los datos de entrenamiento usando el modelo de *SentenceTransformer*. Finalmente, crea un modelo de regresión logística y lo entrena con los datos de entrenamiento, retornándolo como salida. En el caso donde la variable *production* es Falsa, se calculan las incrustaciones de los datos de prueba y se

muestran en pantalla la exactitud del modelo junto con el reporte de clasificación de la regresión logística.

#### 4.3.2 Función `emotionClassifier`

Esta función realiza la clasificación de un estado de ánimo ingresado por el usuario.

Recibe como argumentos:

- `Modelo_LR(LogisticRegression)`: Modelo de regresión logística entrenado para clasificación de estados de ánimo.
- `Estado_animo(str)`: Cadena de texto con el estado de ánimo del usuario.

La función `emotionClassifier` carga un modelo pre-entrenado, específicamente el modelo “*distiluse-base-multilingual-cased-v1*” usando *SentenceTransformer*. A continuación, adapta la cadena de texto correspondiente al estado de ánimo de la persona convirtiéndola a minúsculas y calcula su incrustación usando el modelo de *SentenceTransformer*. Finalmente, utiliza el modelo de regresión logística para predecir la salida.

### 4.4 HITO 4: SISTEMA DE RECOMENDACIONES

Después de solicitar al usuario su estado de ánimo, el algoritmo le pide que ingrese una frase que resuma la temática que desea explorar. Esta frase se procesa mediante varias funciones, generando una serie de sugerencias alineadas con la solicitud del usuario y su estado anímico. Para lograr este resultado, se desarrollaron diversas funciones específicas.

#### 4.4.1 Función `embeddingsGenerator`

Esta función genera y retorna las incrustaciones de las distintas bases de datos.

Recibe como argumentos:

- `Df_imdb(pd.DataFrame)`: Base de datos de películas.
- `Df_bgg(pd.DataFrame)`: Base de datos de juegos de mesa.
- `Df_gutenberg(pd.DataFrame)`: Base de datos de libros.

La función `embeddingsGenerator` carga un modelo pre-entrenado, específicamente el modelo *distiluse-base-multilingual-cased-v1*, utilizando *SentenceTransformer*. Luego, genera las incrustaciones de las descripciones de películas, juegos de mesa y libros por separado, y las devuelve como salida.

#### 4.4.2 Función `recommendations`

Esta función muestra en pantalla recomendaciones de contenido.

Recibe como argumentos:

- `User_prompt(str)`: Frase de solicitud de contenido por parte del usuario.



- `Embeddings (Any)`: Las incrustaciones de una base de datos.
- `Top (int)`: Cantidad de recomendaciones a retornar.
- `Category (str)`: Categoría de la base de datos (“películas”, “juegos de mesa”, “libros”).
- `df (pd.DataFrame)`: `DataFrame` de la base de datos usada en las recomendaciones.

La función `recommendations` carga un modelo pre-entrenado, específicamente *distiluse-base-multilingual-cased-v1*, utilizando *SentenceTransformer*. Luego, emplea la función *GoogleTranslator* de la librería *Deep-Translator* para traducir el *prompt* del usuario al inglés antes de calcular sus incrustaciones mediante *SentenceTransformer*.

Se calculan las puntuaciones de similitud entre las incrustaciones del *prompt* y las de la base de datos, utilizando la similitud de coseno mediante la función *cos\_sim* de *util*. Cada puntuación y su índice correspondiente (que indica el dato específico de la base de datos) se guardan en un diccionario que resume la similitud del *prompt* con cada entrada de la base de datos seleccionada. Todos los diccionarios se almacenan en una lista, la cual se ordena de mayor a menor según las puntuaciones, de modo que los primeros elementos de la lista representen los datos con mayor similitud al *prompt* del usuario.

Finalmente, se crea una lista que muestra la cantidad de elementos especificada como parámetro de la función. Esto se consigue iterando los elementos de la lista de similitudes, usando el índice y la categoría definida como argumento, para buscar en el `DataFrame` ingresado los datos de los contenidos recomendados y agregarlos a la tabla. La lista es convertida a un `DataFrame`, al cual se le modifican diversas propiedades gráficas para que tenga una interfaz más amigable con el usuario. La tabla de recomendaciones se muestra en pantalla mediante la función *display*, que la imprime como un objeto HTML, conforme a los requisitos del código principal (*Main*).

#### 4.4.3 Función `labelsDetector`

Esta función detecta etiquetas específicas contenidas en un *prompt* del usuario y las retorna como salida.

Recibe como argumento:

- `User_prompt (str)`: La frase de solicitud de contenido por parte del usuario.

La función `labelsDetector` carga un modelo pre-entrenado, específicamente “urchade/gliner\_multi-v2.1” usando *GLiNER*. A continuación, cambia el modelo a modo de evaluación para desactivar ciertas características como el “dropout” durante la inferencia. Luego, genera una lista de etiquetas que serán usadas para reconocer elementos clave que serán usados durante la recomendación: actor, director, genre (género) y author (autor).

Se predicen las entidades del *prompt* del usuario utilizando las etiquetas definidas previamente, y se almacenan los resultados. A continuación, se inicializa un diccionario en

el que las etiquetas detectadas actúan como claves, mientras que las entidades correspondientes se almacenan como valores en forma de listas (para gestionar múltiples instancias de la misma etiqueta). A medida que se recorren las entidades reconocidas en el texto, si la etiqueta ya existe en el diccionario, se añade el texto a la lista correspondiente; si no existe, se crea una nueva lista con el primer texto de la entidad.

La función retorna un diccionario donde cada clave es una etiqueta y su valor es una lista de todas las entidades detectadas para dicha etiqueta.

#### 4.4.4 Función recommender

Esta función genera recomendaciones de películas, libros y/o juegos de mesa en función de un estado de ánimo y una frase temática suministrada por un usuario. La función recibe como argumentos:

- `moodState(str)`: Frase o palabra asociada al estado de ánimo.
- `moodModel(Any)`: Modelo clasificador de estados de ánimo.
- `topicPhrase(str)`: Frase o palabra asociada una temática (prompt).
- `embeddings(Any)`: Incrustaciones de las bases de datos.

La función `recommender` comienza llamando a la función `emotionClassifier` enviando como parámetros el estado de ánimo y el modelo clasificador recibidos en sus argumentos. Obtenida la clasificación del estado de ánimo, define dos conjuntos de etiquetas (que más adelante obtendrá del prompt usando NER) que le servirán para interpretar si el usuario desea explorar una película o un libro:

- Películas: actor, director y genre (género)
- Libros: author (autor)

A continuación, divide el arreglo de incrustaciones en tres, una por cada tipo de base de datos. El último paso previo a la recomendación es obtener las etiquetas contenidas en la frase temática suministrada por el usuario mediante la función `labelsDetector`.

El sistema de recomendación toma en cuenta muchos factores a la hora de recomendar que se resumen en la siguiente tabla:

Etiquetas detectadas (NER)	Clasif. estado de ánimo	Cant. películas	Cant. libros	Cant. juegos de mesa
Sólo de películas	Positivo	3	-	-
	Neutro	4	-	-
	Negativo	5	-	-
De películas y libros	Positivo	3	3	-
	Neutro	4	4	-
	Negativo	5	5	-
Sólo de libros	Positivo	-	3	-

Ninguna	Neutro	-	4	-
	Negativo	-	5	-
	Positivo	2	1	3
	Neutro	1	3	2
	Negativo	3	2	1

Tabla 1: Sistema de recomendación.

El criterio utilizado para definir el sistema de recomendaciones se basa en los siguientes puntos:

- Si el usuario especifica en su *prompt* detalles como actores, géneros o directores, se interpreta que está buscando ver una película, por lo que solo se le recomendarán películas. La cantidad de recomendaciones varía según su estado de ánimo, incrementando a medida que éste es más negativo. Esta decisión busca ofrecerle un mayor conjunto de opciones para brindarle apoyo en su estado emocional.
- Si el usuario incluye en su *prompt* detalles como el nombre de un autor, se aplica la misma lógica que en el caso anterior, pero con recomendaciones de libros.
- Si el usuario incluye detalles que se corresponderían tanto a libros como películas, se le sugieren de ambas categorías. El estado de ánimo influye de la misma manera que en los puntos anteriores.
- Si el usuario no incluye detalles que puedan clasificarse dentro de las etiquetas definidas, el criterio de recomendación difiere considerablemente de los puntos anteriores. En este caso, se ofrecen contenidos de todas las categorías, y la cantidad de recomendaciones en cada una está determinada por el estado de ánimo del usuario. Se considera que una persona con un estado de ánimo positivo puede participar más activamente, por lo que los juegos de mesa son la opción más recomendada. Con un estado de ánimo neutro, es probable que el usuario esté en un humor más reflexivo, por lo que los libros son la recomendación prioritaria. Finalmente, si el estado de ánimo es negativo, las películas se convierten en el contenido principal recomendado, ya que requieren menos esfuerzo físico y participación activa.

Lo anterior se implementa en el código mediante cuatro condiciones `if`, cada una correspondiente al grupo de etiquetas detectadas mediante NER. Dentro de cada condición, se utiliza una estructura `match-case` para determinar la cantidad de libros, películas o juegos de mesa que se deben recomendar en función de cada caso. Finalmente, se llama a la función `recommendations` tantas veces como tipos de contenido se requieran, pasándole como argumentos el *prompt*, las incrustaciones, la cantidad de recomendaciones, la categoría y los *datasets* correspondientes, generando un conjunto de recomendaciones que se imprime en pantalla.

## 4.5 HITO 5: PROGRAMA PRINCIPAL (MAIN)

El programa principal integra todas las funciones y bases de datos descritas anteriormente, conectando los componentes individuales para formar el algoritmo completo. Este programa incluye algunas funciones adicionales, junto con el código principal (*main*), que coordina su ejecución.

### 4.5.1 Función `getDataFrame`

Esta función crea un `DataFrame` a partir de un archivo CSV alojado en Google Drive, retornándolo a su salida. Recibe como argumentos:

- `File_id(str)`: ID del archivo (codificado según Google Drive).
- `Encoding(str)`: Codificación de caracteres que se utilizará para leer el archivo.
- `Delimiter(str)`: Especifica el carácter que separa los valores en el archivo.
- `Show_head(bool)`: Si es `True`, muestra las primeras filas del `DataFrame`.

La función `getDataFrame` comienza construyendo la URL de descarga del archivo, concatenando la URL base de Google Drive con el ID proporcionado como argumento. Luego, utiliza la función `gdown.download` para cargar el archivo en el entorno de trabajo, empleando la URL generada y un nombre para el archivo de salida. Finalmente, crea un `DataFrame` a partir del archivo CSV descargado, aplicando las configuraciones de codificación y delimitadores recibidos como argumentos, y lo retorna como salida.

### 4.5.2 Función `onRecommendButtonClicked`

Verifica si el estado de ánimo y la frase descriptiva ingresados por el usuario son válidos. Si alguno no lo es, se solicita al usuario que vuelva a ingresar la información correspondiente; de lo contrario, se genera la recomendación. La función recibe como parámetros:

- `moodInput(str)`: Widget de entrada de texto para el estado de ánimo.
- `topicInput(str)`: Widget de entrada de texto para la frase descriptiva.
- `moodModel(Any)`: Modelo de clasificación de estados de ánimo.
- `embeddings(Any)`: Incrustaciones de las bases de datos.
- `outputMessage(widgets.Output)`: Widget de salida que muestra mensajes al usuario.

La función `onRecommendButtonClicked` interactúa con los widgets de `ipywidgets` según las acciones del código principal (*main*). Primero, limpia los mensajes de salida y extrae las cadenas de texto de los argumentos de entrada, correspondientes al estado de ánimo y a la frase descriptiva de contenido. Luego, evalúa la longitud de ambos y, si no alcanzan el valor mínimo requerido, muestra un mensaje de error en pantalla solicitando al usuario que ajuste los parámetros ingresados. Si las validaciones son correctas, se llama a la función `recommender` parametrizándola con el estado de ánimo, el modelo de clasificación, el *prompt* del usuario y las incrustaciones de las bases de datos, iniciando así el proceso de recomendación.

#### 4.5.3 Main

El programa principal organiza y coordina todas las funciones definidas previamente, estructurando el algoritmo de manera ordenada. Además, proporciona una interfaz gráfica amigable para el usuario con mensajes informativos durante el proceso, utilizando la librería `ipywidgets`. En primer lugar, se crean el título y subtítulo de la interfaz, ambos configurados como elementos HTML.

A continuación, se crea un widget de salida que contiene los mensajes informativos relacionados con la validación de la existencia de los datasets en el entorno de trabajo. Este proceso da paso a la carga de las bases de datos disponibles en Google Drive. La operación se lleva a cabo mediante cuatro bloques `try-except`, uno para cada dataset. En el bloque `try`, se evalúa la existencia de los datasets llamándolos directamente. Si no se logra acceder a ellos, el bloque `except` ejecuta la función `getDataFrame`, pasando como argumento el ID del archivo codificado según Google Drive. Para los datasets de películas y libros, se realiza un procesamiento adicional en los DataFrames, que incluye la gestión de valores nulos y la conversión de datos de tipo `object` a cadenas de texto, ya que estos serán utilizados posteriormente para generar las incrustaciones. Durante cada validación, se imprime un mensaje que indica el proceso que se está llevando a cabo con la base de datos correspondiente. Estos mensajes se almacenan en el widget de salida para ser mostrados al usuario en pantalla.

```
Verificando existencia de DataFrames necesarios...
⚠ Cargando DataFrame de Estados de Ánimo.
⚠ Cargando DataFrame de Películas.
⚠ Cargando DataFrame de Juegos de Mesa.
⚠ Cargando DataFrame de Libros cargado.
```

Figura 3: Mensajes mostrados si los datasets no están en el entorno de trabajo.

```
Verificando existencia de DataFrames necesarios...
✓ DataFrame de Estados de Ánimo cargado.
✓ DataFrame de Películas cargado.
✓ DataFrame de Juegos de Mesa cargado.
✓ DataFrame de Libros cargado.
```

Figura 4: Mensajes mostrados si los datasets sí están en el entorno de trabajo.

Luego, se crea un widget de salida que contiene mensajes informativos relacionados con la validación de la existencia del modelo de clasificación y las incrustaciones de las bases de datos en el entorno de trabajo. Para llevar a cabo el proceso de validación, se utiliza nuevamente la estructura `try-except`. En el bloque `try`, se intenta acceder al modelo de clasificación. Si el modelo no existe, el bloque `except` ejecuta la función `emotionClassifierModel`, pasando como parámetro el `DataFrame` de los estados de ánimo, y guarda el modelo resultante en una variable. Se imprime un mensaje informativo sobre el proceso que se está llevando a cabo con el modelo de clasificación de estados de ánimo. Estos mensajes se almacenan en el widget de salida para ser mostrados al usuario en pantalla.

La última etapa de preparación del entorno consiste en generar las incrustaciones de las bases de datos. Nuevamente, se utiliza la estructura `try-except` para verificar la existencia de las incrustaciones. En el bloque `try`, se evalúa si ya existen, mientras que en el bloque `except`, se llama a la función `embeddingsGenerator`, pasándole como parámetros los `DataFrames` de las bases de datos de películas, libros y juegos de mesa. Las incrustaciones generadas se almacenan en una variable, y los mensajes informativos sobre el proceso se guardan en el widget de salida para ser mostrados al usuario.

```

Verificando existencia de Modelos y Embeddings necesarios...
⚠ Cargando Modelo Clasificador de Emociones.
modules.json: 100% ██████████ 341/341 [00:00<00:00, 6.77kB/s]
config_sentence_transformers.json: 100% ██████████ 122/122 [00:00<00:00, 3.71kB/s]
README.md: 100% ██████████ 2.47k/2.47k [00:00<00:00, 69.5kB/s]
sentence_bert_config.json: 100% ██████████ 53.0/53.0 [00:00<00:00, 898B/s]
config.json: 100% ██████████ 556/556 [00:00<00:00, 9.14kB/s]
model.safetensors: 100% ██████████ 539M/539M [00:02<00:00, 259MB/s]
tokenizer_config.json: 100% ██████████ 452/452 [00:00<00:00, 12.7kB/s]
vocab.txt: 100% ██████████ 996k/996k [00:00<00:00, 2.33MB/s]
tokenizer.json: 100% ██████████ 1.96M/1.96M [00:00<00:00, 3.04MB/s]
special_tokens_map.json: 100% ██████████ 112/112 [00:00<00:00, 1.75kB/s]
1_Pooling/config.json: 100% ██████████ 190/190 [00:00<00:00, 12.3kB/s]
2_Dense/config.json: 100% ██████████ 114/114 [00:00<00:00, 6.75kB/s]
pytorch_model.bin: 100% ██████████ 1.58M/1.58M [00:00<00:00, 20.5MB/s]
model.safetensors: 100% ██████████ 1.58M/1.58M [00:00<00:00, 21.6MB/s]
⚠ Cargando Embeddings.

```

Figura 5: Mensajes mostrados si no existen el modelo de clasificación ni las incrustaciones.

```

Verificando existencia de Modelos y Embeddings necesarios...
✓ Modelo Clasificador de Emociones cargado.
✓ Embeddings cargadas.

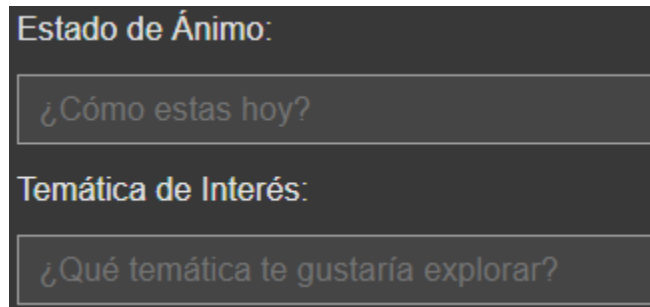
```

Figura 6: Mensajes mostrados si existen el modelo de clasificación y las incrustaciones.

Los pasos descritos anteriormente tienen un tiempo de ejecución muy alto, ya que se deben descargar múltiples modelos pre-entrenados de gran tamaño y generar incrustaciones (el proceso más lento de todos), sumado a que las descargas de las bases de

datos también consumen tiempo. Es por esto que el programa se asegura que las tareas sean realizadas una única vez y no siempre que el usuario necesite utilizar la aplicación.

Se crea a continuación un widget con campos de entrada donde el usuario pueda ingresar su estado de ánimo y una temática de contenido que le gustaría explorar. Los valores ingresados por el usuario son almacenados en dos variables.

El formulario tiene un fondo gris oscuro. En la parte superior, el texto "Estado de Ánimo:" está en color naranja. Debajo de él hay un campo de entrada rectangular con un borde gris claro y el texto de marcador de posición "¿Cómo estas hoy?" en gris claro. Más abajo, el texto "Temática de Interés:" está en color naranja. Debajo de él hay otro campo de entrada rectangular con un borde gris claro y el texto de marcador de posición "¿Qué temática te gustaría explorar?" en gris claro.

*Figura 7: Campos de entrada para el usuario.*

Se crea un último widget de salida para mostrar los mensajes generados por la función `onRecommendButtonClicked`. Esta función es invocada a través de un nuevo widget de botón, el cual debe ser presionado por el usuario para obtener las recomendaciones. La función se parametriza con el estado de ánimo del usuario, el prompt, el modelo de clasificación, las incrustaciones y el widget de salida para gestionar los mensajes

Un botón rectangular con un fondo azul brillante y un borde negro. El texto "Obtener Recomendación" está en blanco y está centrado dentro del botón.

*Figura 8: Botón de recomendación.*

Finalmente, se crea la interfaz gráfica completa al combinar todos los widgets generados durante el proceso. La interfaz se muestra en pantalla, permitiendo al usuario interactuar con ella de manera intuitiva.



## Clasificador de Recomendaciones Recreativas

Este sistema te recomendará opciones de entretenimiento según tu estado de ánimo y preferencias temáticas.

Verificando existencia de DataFrames necesarios...

- ✓ DataFrame de Estados de Ánimo cargado.
- ✓ DataFrame de Películas cargado.
- ✓ DataFrame de Juegos de Mesa cargado.
- ✓ DataFrame de Libros cargado.

Verificando existencia de Modelos y Embeddings necesarios...

- ✓ Modelo Clasificador de Emociones cargado.
- ✓ Embeddings cargadas.

Estado de Ánimo:

Temática de Interés:

Obtener Recomendación

Figura 9: Interfaz de usuario completa.

## 5. RESULTADOS

Debido a la complejidad del proceso y las múltiples etapas involucradas, se considera adecuado analizar los resultados de manera segmentada, evaluando el desempeño de cada componente individualmente antes de realizar una evaluación global.

### 5.1 BASE DE DATOS DE ESTADOS DE ÁNIMO

Se consiguió generar un dataset de 254 elementos que representan estados de ánimo. El dataset comprende no únicamente palabras, sino también frases del español tradicional y del lunfardo, por lo que comprende un gran rango de estados de ánimo del vocabulario de usuarios argentinos. Las clasificaciones de sus categorías son correctas y no dejan lugar a ambigüedades.

### 5.2 BASE DE DATOS DE LIBROS

Cada libro del *Proyecto Gutenberg* incluye una gran cantidad de datos descriptivos, aunque no todos presentan las mismas categorías de información. Para evitar complicaciones durante el proceso de web scraping, se optó por limitar los datos cargados en la base de datos generada a aquellos atributos comunes a todos los libros. Además, se descartaron datos considerados irrelevantes, como el copyright o la cantidad de descargas del ebook.

En cuanto al género ("subject" en el sitio web), cada libro poseía una cantidad variable de categorías. Para evitar conflictos durante el proceso de web scraping, se decidió tomar únicamente el primer género de cada libro, consciente de que esto implicaba sacrificar parte de la información descriptiva. A pesar de ello, el dataset resultante mantuvo una amplia gama de características descriptivas, ya que la mayoría de la información contenida en el resumen ("summary" en el sitio web) estaba disponible para todos los libros.

### 5.3 CLASIFICACIÓN DEL ESTADO DE ÁNIMO

La efectividad de la clasificación del estado de ánimo, afortunadamente, puede ser medida. Como mencionamos anteriormente, para el entrenamiento del modelo se dividió el dataset de estados de ánimo en proporción 80-20, es decir, un 80% de los datos dedicados al entrenamiento del modelo y un 20% de los datos dedicados a la prueba del mismo. Los resultados obtenidos de la prueba del modelo se observan en el reporte de clasificación a continuación:

	Precision	Recall	F1-Score	Support
Negativo	0,61	0,79	0,69	14
Neutro	0,67	0,62	0,65	16
Positivo	0,78	0,67	0,72	21

Accuracy			0,69	51
Macro Avg.	0,69	0,69	0,68	51
Weighted Avg.	0,70	0,69	0,69	51

Tabla 2: Reporte de clasificación de modelo de regresión logística.

Siendo que el dataset es balanceado (cantidad de datos por clase similares), se podría usar la métrica de exactitud (accuracy) para determinar cuánto se adapta el modelo a los datos de prueba. En este caso, la exactitud es de casi el 70%, por lo que acierta la clasificación del estado de ánimo en la mayoría de los casos.

Los F1-Score, que resumen la precisión (precisión) y la exhaustividad (recall), son bastante similares en todas las clases. Se observa que quizá la clase “Neutro” sea la que más dificultades tenga a la hora de clasificarse correctamente.

Desde el punto de vista de la precisión es la clase “Negativo” la que posee más falsos positivos, dicho de otra manera, la que más clasifica estados de ánimo de otras clases erróneamente como “Negativo”. La clase “Positivo” es la que menos falsos positivos tiene, con un acierto del casi 80%.

Finalmente, desde el punto de vista de la exhaustividad, se observa que la clase “Neutro” es la que tiene más falsos negativos, es decir, la que más falla en clasificar un estado de ánimo “Neutro” como tal. La clase “Negativo” es la que menos falsos negativos tiene, con un acierto del casi 80%.

Los resultados, en resumen, si bien no son los mejores son considerados “aceptables” dentro de las herramientas que manejamos.

Sobre este modelo se realizaron muchos ensayos, a continuación, presentamos algunos ejemplos:

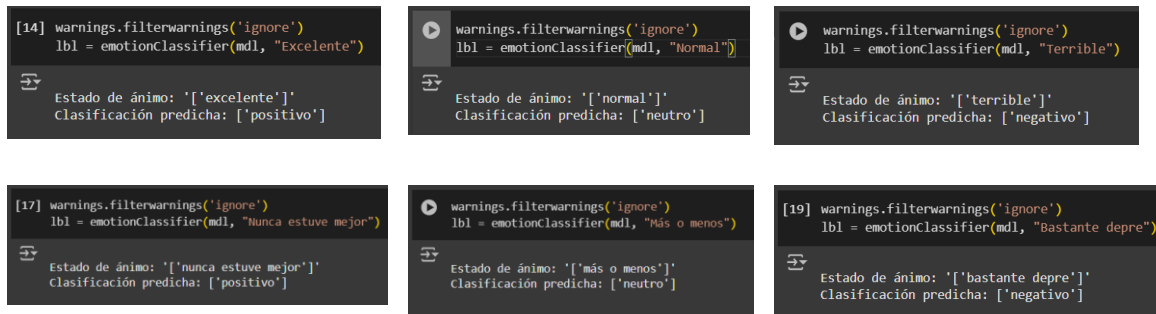
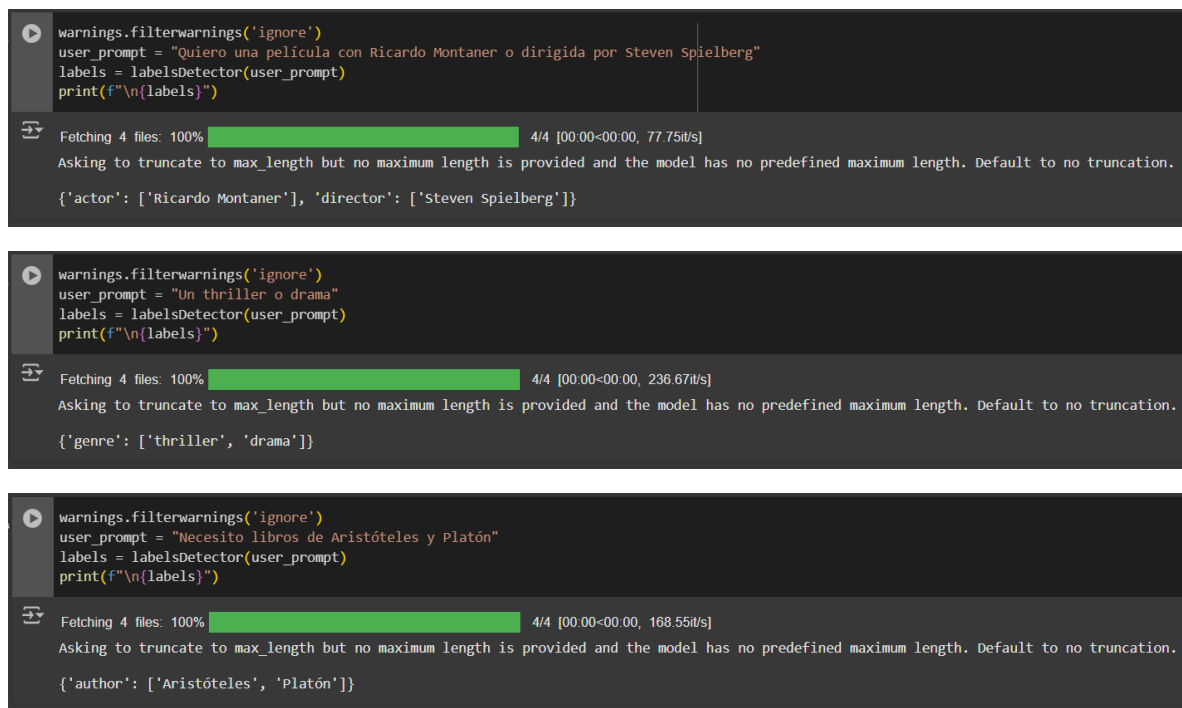


Figura 10: Ejemplos de resultados obtenidos sobre palabras y frases para todas las categorías.

Se ha probado utilizando todo tipo de palabras y frases en el español clásico o lunfardo y la tasa de éxito en la clasificación resultó enorme. Incluso habiendo visto que la tasa de éxito teórica es de aproximadamente el 70%, no se consiguió encontrar frases simples que no pudiera clasificar correctamente. Se considera entonces que el modelo de clasificación de estado de ánimo cumple con su objetivo.

## 5.4 SISTEMA DE RECOMENDACIONES

El primer componente dentro del sistema de recomendaciones es el reconocimiento de entidades nombradas (NER). Utilizando un modelo pre-entrenado, se solicita que etiquete entidades dentro del prompt del usuario, asignándolas a las categorías: actor, director, género (genre) y autor (author). Aunque el rendimiento del etiquetado está limitado por las capacidades del modelo, en general, el sistema muestra buenos resultados en la mayoría de los casos. El modelo identifica entidades principalmente por contexto, por lo que su precisión dependerá de la claridad y redacción del prompt. Además, es sensible a errores ortográficos, lo que podría dificultar el reconocimiento de un género o entidad si está mal escrito. Abajo, se presentan algunos ejemplos de reconocimiento:



```
warnings.filterwarnings('ignore')
user_prompt = "Quiero una película con Ricardo Montaner o dirigida por Steven Spielberg"
labels = LabelsDetector(user_prompt)
print(f"\n{labels}")
```

```
Fetching 4 files: 100% ██████████ 4/4 [00:00<00:00, 77.75it/s]
Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length. Default to no truncation.
{'actor': ['Ricardo Montaner'], 'director': ['Steven Spielberg']}
```

```
warnings.filterwarnings('ignore')
user_prompt = "Un thriller o drama"
labels = LabelsDetector(user_prompt)
print(f"\n{labels}")
```

```
Fetching 4 files: 100% ██████████ 4/4 [00:00<00:00, 236.67it/s]
Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length. Default to no truncation.
{'genre': ['thriller', 'drama']}
```

```
warnings.filterwarnings('ignore')
user_prompt = "Necesito libros de Aristóteles y Platón"
labels = LabelsDetector(user_prompt)
print(f"\n{labels}")
```

```
Fetching 4 files: 100% ██████████ 4/4 [00:00<00:00, 168.55it/s]
Asking to truncate to max_length but no maximum length is provided and the model has no predefined maximum length. Default to no truncation.
{'author': ['Aristóteles', 'Platón']}
```

Figura 11: Ejemplos de reconocimiento de entidades nombradas.

Finalmente, se ensaya el programa completo con diversos estados de ánimo y frases con temáticas de interés. El algoritmo responde correctamente de acuerdo a las condiciones mencionadas en la sección 4.4.4, generando recomendaciones en base a las similitudes entre el prompt del usuario y las incrustaciones de las bases de datos en las cantidades y categorías correspondientes. Mientras más detallado sea el prompt del usuario, mayor tasa de éxito habrá en las recomendaciones.

## Clasificador de Recomendaciones Recreativas

Este sistema te recomendará opciones de entretenimiento según tu estado de ánimo y preferencias temáticas.

Verificando existencia de DataFrames necesarios...

- ✓ DataFrame de Estados de Ánimo cargado.
- ✓ DataFrame de Películas cargado.
- ✓ DataFrame de Juegos de Mesa cargado.
- ✓ DataFrame de Libros cargado.

Verificando existencia de Modelos y Embeddings necesarios...

- ✓ Modelo Clasificador de Emociones cargado.
- ✓ Embeddings cargadas.

Estado de Ánimo:

depre

Temática de Interés:

romántico o de suspenso

Obtener Recomendación

Generando recomendaciones...

Fetching 4 files: 100% 4/4 [00:00<00:00, 139.12it/s]

Asking to truncate to max\_length but no maximum length is provided and the model has no predefined ma

### Recomendaciones de Películas

	Puesto	Sugerencias de Películas	Puntuación de Similitud
0	1	The Lobster	0.2120
1	2	The Ugly Truth	0.2114
2	3	(500) Days of Summer	0.2043
3	4	To Rome with Love	0.1999
4	5	He's Just Not That Into You	0.1915

Figura 12: Ejemplos de recomendaciones para estado de ánimo negativo y prompt con géneros de películas.

Sin embargo, encontramos algunos comportamientos indeseados. Algunas veces, se detectan etiquetas incorrectas, lo que produce resultados inesperados como el del siguiente ejemplo, donde se le pidió un juego de ciertas características, pero el algoritmo interpretó que el prompt contenía el género de una película.

## Clasificador de Recomendaciones Recreativas

Este sistema te recomendará opciones de entretenimiento según tu estado de ánimo y preferencias temáticas.

Verificando existencia de DataFrames necesarios...

- ✓ DataFrame de Estados de Ánimo cargado.
- ✓ DataFrame de Películas cargado.
- ✓ DataFrame de Juegos de Mesa cargado.
- ✓ DataFrame de Libros cargado.

Verificando existencia de Modelos y Embeddings necesarios...

- ✓ Modelo Clasificador de Emociones cargado.
- ✓ Embeddings cargadas.

Estado de Ánimo:

Eufórico

Temática de Interés:

juego de rompecabezas épico

Obtener Recomendación

Generando recomendaciones...

Fetching 4 files: 100% 4/4 [00:00<00:00, 259.55it/s]

Asking to truncate to max\_length but no maximum length is provided and the model has no predefined max

### Recomendaciones de Películas

	Puesto	Sugerencias de Películas	Puntuación de Similitud
0	1	Wreck-It Ralph	0.1563
1	2	The Imitation Game	0.1521
2	3	Casino Royale	0.1337

Figura 13: Ejemplo de recomendación indeseada.

Por otro lado, al ser los juegos de mesa algo tan distinto a los libros o películas, los prompts de la primera categoría se conforman muy diferentes a las otras categorías. Es por eso que cuando el usuario ingresa un prompt orientado a historias, el algoritmo recomendará acertadamente películas y libros, pero recomendará juegos no muy relacionados al mismo. Ocurre algo similar en el caso opuesto, cuando el usuario ingresa un prompt asociado a juegos de mesa el algoritmo no resuelve bien las recomendaciones de películas o libros.

Estado de Ánimo:

más o menos

Temática de Interés:

historia de amor y venganza

Obtener Recomendación

Generando recomendaciones...

Fetching 4 files: 100% 4/4 [00:00<00:00, 261.78it/s]

Asking to truncate to max\_length but no maximum length is provided and the model has no predefined m

**Recomendaciones de Películas**

	Puesto	Sugerencias de Películas	Puntuación de Similitud
0	1	Endless Love	0.2289

**Recomendaciones de Libros**

	Puesto	Sugerencias de Libros	Puntuación de Similitud
0	1	Cuentos de Amor de Locura y de Muerte	0.3279
1	2	The Legend of Sleepy Hollow	0.2980
2	3	Romeo and Juliet	0.2971

**Recomendaciones de Juegos de mesa**

	Puesto	Sugerencias de Juegos de mesa	Puntuación de Similitud
0	1	Betrayal Legacy	0.3069
1	2	Near and Far	0.2637

Figura 14: Ejemplo de rendimiento medio.

## 6. CONCLUSIONES

Entre todos los componentes del algoritmo, se pueden considerar a dos de ellas como sus principales: la clasificación de estados de ánimo y la recomendación de contenido. En el apartado de clasificación de estados de ánimo el algoritmo presenta un comportamiento muy bueno. Posee pocos errores a la hora de clasificar y la cantidad de clases de clasificación son suficientes para los objetivos planteados. Los modelos de *SentenceTransformer* se basan en BERT, cuyo uso es muy difundido debido a su efectividad a pesar de ser, dentro de todo, antiguo.

Por otra parte, el apartado de recomendación de contenido tiene mucho lugar a mejoras. El algoritmo es bueno encontrando similitudes con libros y películas si en el prompt se incluyen autores, actores o directores (apoyado en NER para identificarlos), sin embargo, no se puede decir lo mismo para el caso de los juegos de mesa. La realidad es que tanto los libros como las películas cuentan historias, por lo que un prompt del usuario que describe una situación podría combinar muy bien con sus premisas, pero usar ese mismo prompt para sugerir juegos de mesa no sirve completamente. El entretenimiento de los juegos no se centra (tanto) en las historias, sino en la jugabilidad que es lo que se describe para cada uno ellos. Lo opuesto podría ocurrir si el prompt del usuario apunta a la jugabilidad de un juego de mesa, allí las recomendaciones de películas y libros no serán precisas.

El algoritmo podría ser mejorado incorporando un reconocimiento de la *intención* del usuario. Detectar su deseo, si se trata de consumir historias o de entretenerse jugando. Sin embargo, con los recursos vistos hasta el momento del presente trabajo no se le ha podido encontrar una solución. Por otro lado, se podría desarrollar un chatbot que realice preguntas específicas basadas en las interacciones previas, con el objetivo de refinar y orientar mejor las recomendaciones. De esta forma, el sistema no se limitaría a un único prompt inicial, sino que podría ajustarse dinámicamente según las respuestas y preferencias del usuario a lo largo de la conversación.

Habiendo considerado todos los aspectos, el algoritmo ha logrado cumplir con los objetivos establecidos, proporcionando recomendaciones de contenido basadas en el estado de ánimo del usuario y una descripción de sus preferencias. Al mismo tiempo, hay lugar a mejoras de rendimiento lo que permitiría que las recomendaciones sean más precisas y ajustadas a las expectativas del usuario.