



# Clasificador de Recomendaciones Recreativas

TUIA - Procesamiento del Lenguaje Natural

Trabajo Práctico 1

Autores:

- **Avecilla Tomás Valentino**
- **Calcía Franco Nicolás**

Profesores:

- **Alan Geary**
- **Juan Pablo Manson**

Fecha de Entrega: Miércoles 6 de Noviembre de 2024

Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

## Descripción del trabajo:

Este proyecto desarrolla un sistema de recomendaciones recreativas basado en el Procesamiento de Lenguaje Natural (NLP). El sistema tiene como objetivo identificar el estado de ánimo del usuario a través de una frase ingresada, clasificarlo en una de las categorías: "Alegre", "Melancólico" o "Ni fu ni fa", y luego, a partir de una frase de preferencia, recomendar libros, juegos de mesa o películas.

Para lograrlo, se utilizan embeddings generados con BERT, junto con técnicas (*similitud de coseno*) para medir la afinidad semántica entre las preferencias del usuario y las descripciones de cada opción de entretenimiento disponible. Los datos provienen de tres fuentes: una base de libros, otra de juegos de mesa y otra de películas, integradas en un sistema de Filtrado Global que permite ofrecer recomendaciones mixtas.

Los [datasets de juegos y películas](#) fueron proporcionados y el de libros fue generado a desde la siguiente [web](#), usando técnicas de web scraping.

## Introducción

Este proyecto se desarrolla en un contexto de vacaciones en la playa, donde el usuario prevé varios días de lluvia que limitarán las actividades al aire libre. Ante esta situación, surge la necesidad de un sistema de recomendaciones recreativas para esos días.

La justificación de este trabajo está en el potencial del Procesamiento de Lenguaje Natural para comprender las preferencias y emociones de los usuarios. Esto permite cosas como crear este sistema de recomendaciones personalizadas que supera las limitaciones de los enfoques tradicionales.

## Objetivos Específicos

1. Desarrollar un clasificador que identifique el estado de ánimo del usuario a partir de una frase ingresada, categorizando las emociones en "Alegre", "Melancólico" o "Ni fu ni fa".
2. Implementar un sistema de recomendación que relacione la preferencia del usuario con opciones de entretenimiento en tres categorías (libros, juegos y películas).

## Estructura del Informe

El informe está organizado en las siguientes secciones:

- Descripción del Problema y Justificación: Explicación detallada del contexto y la necesidad del proyecto.
- Metodología: Detalle de los métodos y técnicas de NLP utilizados.
- Implementación: Explicación de las partes clave del sistema.
- Resultados: Análisis de los resultados obtenidos con ejemplos específicos de recomendaciones.
- Conclusiones: Resumen de los hallazgos, limitaciones del sistema y posibles mejoras.

# Metodología

## Fuente y Datos Utilizados

El sistema de recomendaciones utiliza tres fuentes de datos para cubrir distintas opciones de entretenimiento:

1. Base de Datos de Libros: Se realizó web scraping del Proyecto Gutenberg, recopilando información sobre los 1000 libros más populares, incluyendo título, autor, resumen y géneros. Este conjunto de datos aporta contenido textual descriptivo que es ideal para recomendaciones literarias.
2. Base de Datos de Juegos de Mesa: Utilizamos una base de datos de Board Game Geek, que incluye datos sobre los 1000 juegos mejor valorados, con campos como nombre, descripción, mecánicas y categorías.

3. Base de Datos de Películas: La base de datos de IMDB se utilizó para recomendaciones de películas, proporcionando información sobre título, género, descripción, director y actores, además de calificaciones y puntajes.

### Métodos y Técnicas Utilizados

Se implementaron las siguientes técnicas de procesamiento de lenguaje natural:

1. Preprocesamiento:

- **Modelo de lenguaje:** Se cargó el modelo `en_core_web_md` de SpaCy para el procesamiento de texto.
- **Stopwords:** Se utilizaron las stopwords definidas en `spacy` para eliminar palabras comunes e irrelevantes en los géneros.
- **Limpieza de texto:**
  - Se convirtió todo el texto a minúsculas.
  - Se normaliza el texto eliminando acentos y caracteres especiales con `unicodedata`.
  - Se utilizaron expresiones regulares (`re.sub`) para eliminar caracteres no alfanuméricos.

Este proceso ayuda a mejorar la calidad del texto para las siguientes tareas.

2. Clasificación del Estado de Ánimo con BERT:

- Usamos el modelo `nlptown/bert-base-multilingual-uncased-sentiment`, basado en BERT, que clasifica el estado de ánimo en una escala de [1, 2, 3, 4, 5]. En esta escala, 1 y 2 indican sentimientos negativos, 3 es neutral, y 4 y 5 indican sentimientos positivos.

3. Embeddings.:

- Las descripciones de los elementos en los tres datasets (libros, juegos y películas) se transformaron en embeddings usando el modelo BERT, lo que permitió una representación vectorial de cada descripción.

#### 4. NER:

- El modelo utilizado es `en_core_web_trf` de spaCy, que es un modelo preentrenado basado en transformadores, como BERT. Este modelo es muy preciso para tareas de NER.

#### 5. Filtrado Semántico Global:

- En lugar de asignar una categoría específica de entretenimiento a cada estado de ánimo, implementamos un sistema de Filtrado Semántico Global que combina las recomendaciones de los tres datasets según la similitud semántica. Esto permite sugerir opciones de distintas categorías (libros, juegos o películas) en función de la relevancia para la preferencia ingresada.

#### 6. Incrementos:

- Se buscó darle un valor agregado (incrementando la similitud) a aquellas descripciones que hayan sido identificadas con el mismo sentimiento que el ingresado por el usuario, esto permite darle más relevancia a este input.
- El mismo enfoque fue utilizado en caso de encontrar coincidencia de entidades lo cual potencia el sistema con NER y hace la diferencia sobre todo en casos específicos..

### Herramientas y Tecnologías Empleadas

- Librerías de Python:
  - Transformers: Para la implementación de BERT y generación de embeddings.
  - Spacy: Para limpieza y NER.
  - Scikit-learn: Para cálculo de similitud de coseno y procesamiento de datos.
  - Pandas: Para manipulación y análisis de los dataframes de libros, juegos y películas.
  - Ipywidgets: Para la creación de una interfaz interactiva en Google Colab.
- Google Colab.

## Desarrollo/Implementación

En esta subsección vamos a explicar detalles de las partes clave del trabajo.

➤ Generación de embeddings:

```
def obtener_embedding(texto):
    inputs = tokenizer(texto, return_tensors='pt', padding=True, truncation=True, max_length=512)
    inputs = inputs.to(device)

    with torch.no_grad():
        outputs = model(**inputs)

    return outputs.logits.cpu().numpy()
```

Con esta función obtenemos las representaciones numéricas de las descripciones.

- El primer paso es tokenizar el texto con el **tokenizador de BERT**. El texto se convierte en tokens (sub-palabras o palabras) que el modelo BERT puede entender. El parámetro `max_length=512` asegura que no haya secuencias de texto más largas de lo que el modelo puede procesar (BERT tiene un límite de 512 tokens por entrada).
- Después los tokens se pasan a través del modelo **BERT**, que genera una serie de **logits** (predicciones no normalizadas). Estos logits representan las características del texto en forma de vectores numéricos. Se devuelven las representaciones numéricas o embeddings.
- Los **logits** son retornados después de ser movidos a la CPU desde la GPU (si se usó GPU) y se convierten en un array de **NumPy** para su posterior procesamiento.

➤ NER en descripciones:

```
# Aplicar NER y generar embeddings
def procesar_descripcion(descripcion):
    # Aplicar NER
    doc = nlp_ner(descripcion)
    entidades = [(ent.text, ent.label_) for ent in doc.ents]

    # Obtener embedding de la descripción
    embedding = obtener_embedding(descripcion)
```

- El texto de la descripción se procesa utilizando el modelo `nlp_ner` de SpaCy. Identificará las **entidades** dentro del texto.

- El objeto `doc` resultante contiene las entidades identificadas dentro del texto.
- Finalmente, devuelve tanto el embedding como las entidades extraídas.

➤ Modelos de traducción:

```
# Cargar el modelo de traducción inglés a español
device = 0
traductor_en_es = pipeline("translation", model="Helsinki-NLP/opus-mt-en-es", device=device)

# Cargar el modelo de traducción español a inglés
traductor_es_en = pipeline("translation", model="Helsinki-NLP/opus-mt-es-en", device=device)

# Función para traducir descripciones
def traducir_texto(texto):
    traduccion = traductor_en_es(texto, max_length=512)
    return traduccion[0]['translation_text']

# Función para traducir la preferencia del usuario al inglés
def traducir_preferencia(preferencia):
    traduccion = traductor_es_en(preferencia, max_length=512)
    return traduccion[0]['translation_text']
```

- Se cargaron dos pipelines de traducción, uno para traducir del inglés al español y otro para hacerlo del español al inglés. Las funciones creadas toman un texto y lo traducen entre estos dos idiomas según sea necesario, asegurándose de que el texto no exceda el límite de longitud de los modelos de traducción, lo cual permite realizar la traducción de manera eficiente para los modelos de NLP.

➤ Clasificación del Estado de Ánimo

- Para capturar el estado de ánimo del usuario, se implementó un pipeline de análisis de sentimientos utilizando el modelo **BERT Multilingüe**. Este modelo procesa la frase ingresada y clasifica el sentimiento en cinco niveles (1-5), que luego se agrupan en tres categorías principales: *Alegre*, *Melancólico* y *Ni fu ni fa*.
- Se utilizó la función `pipeline` de la librería **Transformers**, que facilita la implementación.
- Esto se hizo en las descripciones de cada dato y luego en la entrada del usuario.

## ➤ Problema de clasificación de estados de ánimo en descripciones.

```
def clasificar_sentimiento_descripcion(descripcion):
    # Tokenizar la descripción con truncamiento
    inputs = tokenizer(descripcion, return_tensors='pt', padding=True, truncation=True, max_length=512).to(device)

    # Clasificar el sentimiento
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
        predicted_class = logits.argmax().item()

    # Clasificar el sentimiento basado en la clase predicha
    # Mapeamos el índice de clase a la etiqueta correspondiente
    if predicted_class in [0, 1]: # Clases 1 y 2
        return "Melancólico"
    elif predicted_class == 2: # Clase 3
        return "Ni fu ni fa"
    elif predicted_class in [3, 4]: # Clases 4 y 5
        return "Alegre"

    # Opción para manejar el caso en que no se encuentre una clasificación
    return "Clasificación no reconocida"

# Aplicar la clasificación de sentimiento a cada dataframe
df_libros['sentiment'] = df_libros['Description'].apply(clasificar_sentimiento_descripcion)
df_juegos['sentiment'] = df_juegos['Description'].apply(clasificar_sentimiento_descripcion)
df_movies['sentiment'] = df_movies['Description'].apply(clasificar_sentimiento_descripcion)
```

- En el caso de las descripciones, se optó por no usar el `sentiment_pipeline` de Hugging Face debido a que creemos que las descripciones de los elementos pueden superar el límite de 512 tokens, lo que genera errores. En lugar de esto, se utilizó una función personalizada para manejar las descripciones largas, procesando el texto de manera más adecuada.
- Sin embargo, en los inputs del usuario, donde las longitudes suelen ser más controladas y breves, sí se utiliza el `sentiment_pipeline`, ya que estos textos no superan el límite de tokens y se adaptan mejor al modelo.

## ➤ Filtrado Semántico Global

Corresponde a la última celda del notebook

### ➤ Flujo de Entrada (Con ipwidgets):

- El usuario proporciona dos entradas: una que representa su estado de ánimo (emocional) y otra con una preferencia de contenido, como un tipo de libro, juego o película. Estas entradas se traducen (si es necesario) y se utilizan para obtener embeddings de la preferencia del usuario.
- **Análisis de Sentimiento:** El estado de ánimo del usuario se clasifica utilizando un modelo de análisis de sentimiento. El sentimiento resultante se usa para ajustar las recomendaciones, alineando las opciones con el estado emocional detectado.



➤ 2. Cálculo de Similitud:

```
# Calcular la similitud en cada dataframe
def calcular_similitud(df, preferencia_embedding):
    # Calcular la similitud entre la preferencia y los embeddings del dataframe
    df['similarity'] = df['embedding'].apply(lambda x: cosine_similarity(preferencia_embedding, x)[0][0])
    return df

def verificar_coincidencias(df, preferencia_texto, sentimiento_usuario):
    # Verifica que 'entidades' esté en el DataFrame
    if 'entidades' not in df.columns:
        df['entidades'] = [[] for _ in range(len(df))] # Columna vacía si falta

    # Extraer entidades de la preferencia del usuario
    entidades_preferencia = [ent.strip() for ent in nlp_ner(preferencia_texto).ents]

    # Crear una lista para almacenar las similitudes ajustadas
    adjusted_similarities = []

    # Recorrer cada fila del DataFrame
    for _, row in df.iterrows():
        # Verificar si hay coincidencias entre las entidades de la descripción y las de la preferencia
        matching_entities = any(ent in row['entidades'] for ent in entidades_preferencia)
        similarity = row['similarity'] # Obtener similitud antes del ajuste

        # Ajustar similitud en función de las coincidencias de entidades
        if matching_entities:
            similarity += 0.1 # Incremento si hay coincidencias

        # Incrementar similitud si el sentimiento coincide
        if row['sentiment'] == sentimiento_usuario:
            similarity += 0.1 # Ajuste por sentimiento

        # Asegurarse de que similarity no exceda 1
        similarity = min(similarity, .9999)

        # Agregar la similitud ajustada a la lista
        adjusted_similarities.append(similarity)

    # Asignar las similitudes ajustadas al DataFrame
    df['similarity'] = adjusted_similarities

    return df
```

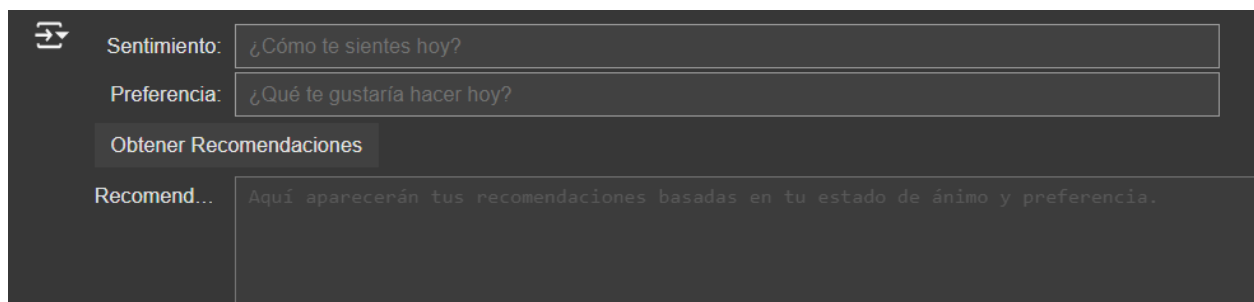
- Se calcula la similitud entre el embedding de la preferencia del usuario y los embeddings de las descripciones de los elementos en los dataframes de libros, juegos y películas. Esta similitud se mide utilizando la **distancia coseno**.
- Se realiza un **ajuste de la similitud** en dos casos:
  - **Coincidencia de Entidades:** Si las entidades (por ejemplo, nombres de personajes, lugares) de la preferencia del usuario coinciden con las de las descripciones, se incrementa la similitud.
  - **Coincidencia de Sentimiento:** Si el sentimiento del usuario coincide con el sentimiento de la descripción del producto, también se incrementa la similitud.
- Generación de Recomendaciones:
  - Después se generan las recomendaciones tomando en cuenta las descripciones y los títulos más cercanos a las preferencias del usuario.

# Resultados

## Presentación de los Resultados

Los resultados muestran cómo el sistema de recomendaciones identifica y adapta las opciones de entretenimiento a las preferencias y el estado emocional del usuario. Algunos resultados destacados incluyen:

1. **Gráficos de Similitud:** Se muestran gráficos que reflejan la similitud promedio entre la preferencia del usuario y las descripciones de cada categoría, lo que demuestra la precisión en la elección de recomendaciones.
2. **Ejemplos de Recomendaciones:** En una tabla se presentan ejemplos de frases ingresadas por el usuario, el estado de ánimo detectado y las recomendaciones obtenidas.



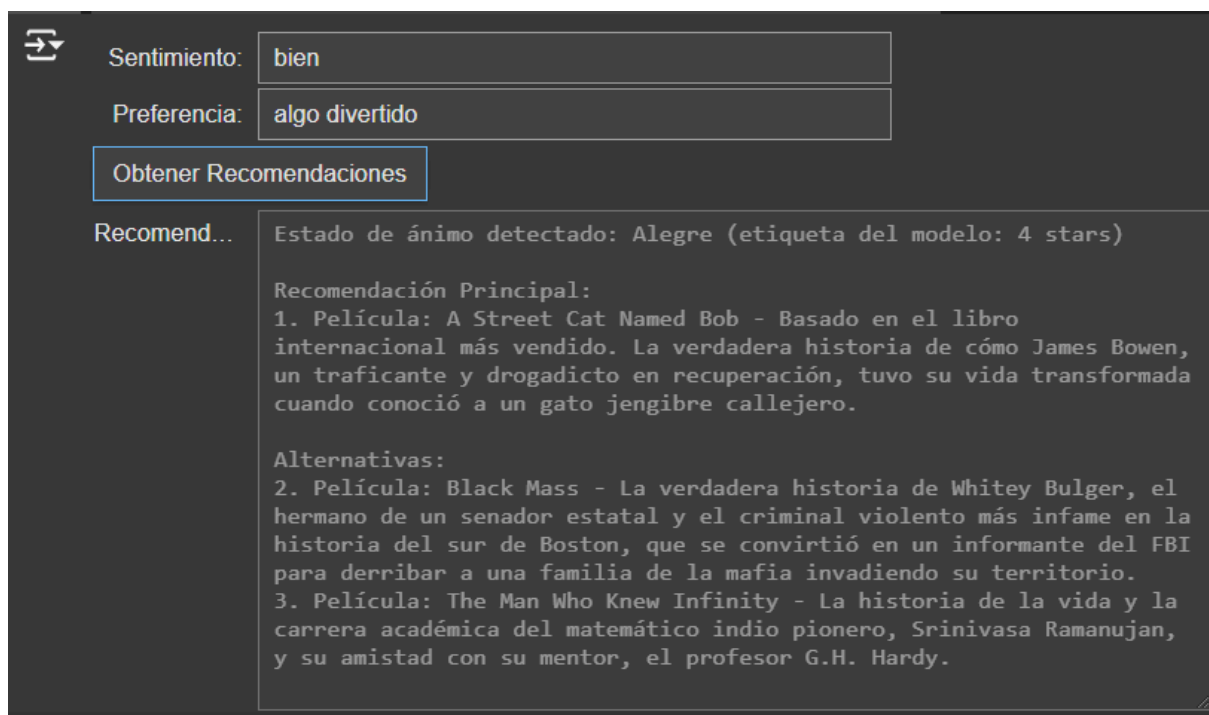
Interface showing input fields for mood and preference, a button to get recommendations, and a placeholder for the results.

Sentimiento: ¿Cómo te sientes hoy?

Preferencia: ¿Qué te gustaría hacer hoy?

Obtener Recomendaciones

Recomend... Aquí aparecerán tus recomendaciones basadas en tu estado de ánimo y preferencia.



Interface showing the results of a recommendation based on the user's mood and preference.

Sentimiento: bien

Preferencia: algo divertido

Obtener Recomendaciones

Recomend... Estado de ánimo detectado: Alegre (etiqueta del modelo: 4 stars)

Recomendación Principal:

1. Película: A Street Cat Named Bob - Basado en el libro internacional más vendido. La verdadera historia de cómo James Bowen, un traficante y drogadicto en recuperación, tuvo su vida transformada cuando conoció a un gato jengibre callejero.

Alternativas:

2. Película: Black Mass - La verdadera historia de Whitey Bulger, el hermano de un senador estatal y el criminal violento más infame en la historia del sur de Boston, que se convirtió en un informante del FBI para derribar a una familia de la mafia invadiendo su territorio.

3. Película: The Man Who Knew Infinity - La historia de la vida y la carrera académica del matemático indio pionero, Srinivasa Ramanujan, y su amistad con su mentor, el profesor G.H. Hardy.

## Análisis y Discusión de los Resultados

El sistema de recomendaciones logró una alta afinidad con las preferencias del usuario en múltiples pruebas, destacándose en los casos donde las descripciones de los elementos coincidían en temas específicos (por ejemplo, “aventuras” o “romance”).

**Limitaciones** observadas:

- **Sensibilidad de los Embeddings:** BERT responde bien en contextos generales, pero puede no capturar todas las variaciones temáticas o culturales en español.
- **Cálculo de Similitud Semántica:** El sistema es dependiente de la precisión de los embeddings; términos similares pueden reducir la exactitud de la recomendación en casos específicos.

## Conclusiones

### Resumen de Resultados

Este proyecto demostró que un sistema de recomendaciones basado en NLP puede adaptarse al estado de ánimo y las preferencias temáticas del usuario, proporcionando opciones relevantes en función del contexto emocional. La combinación de análisis de sentimientos y filtrado semántico global se mostró eficaz para personalizar recomendaciones entre diversas categorías de entretenimiento.

### Evaluación de Objetivos

Los objetivos fueron alcanzados satisfactoriamente:

- Se logró implementar un clasificador de emociones con BERT.
- Se desarrolló un sistema de similitud semántica que permite la selección de recomendaciones variadas y coherentes.
- Se implementó un sistema de recomendaciones mixtas, integrando opciones de entretenimiento sin limitarse a un único tipo.

### Recomendaciones para Trabajos Futuros

- **Fine-tuning del Modelo:** Ajustar el modelo BERT con datos específicos en español podría mejorar la detección de emociones y afinidad temática.
- **Expansión de Categorías de Entretenimiento:** Incluir otras opciones recreativas, como actividades físicas o contenido digital, ampliará las recomendaciones.
- **Mejoras en la Similitud Semántica:** Experimentar con modelos como RoBERTa o BETO podría optimizar aún más la precisión en español.