Informe Del Trabajo Práctico 1 : Clasificador de Recomendaciones Recreativas utilizando NLP

Clasificación del Estado de Ánimo

Descripción General

La función `clasificar_estado_animo_con_reglas` se encarga de determinar el estado de ánimo del usuario en función de la frase que introduce. Para hacerlo, utiliza una combinación de reglas predefinidas (basadas en palabras clave) y el análisis de sentimientos usando un modelo de Procesamiento de Lenguaje Natural (NLP). Esto permite detectar si el usuario se encuentra en un estado "Alegre", "Melancólico" o "Neutral", lo cual será fundamental para recomendar actividades acordes a su estado emocional.

Código

```
def clasificar_estado_animo_con_reglas(frase, resultado=Non
e):
    # Lista de palabras clave para detectar estados negativ
os
    palabras_negativas = ["llorando", "triste", "deprimid
o", "desanimado", "solo", "mal", "angustiado", "preocupad
o", "sufrimiento", "desesperado"]

    # Si detecta palabras negativas, fuerza el estado de án
imo a "Melancólico"
    if any(palabra in frase.lower() for palabra in palabras
_negativas):
        return "Melancólico"

    # Verifica si `resultado` es válido antes de intentar a
cceder a él
```

```
    if resultado is not None and len(resultado) > 0:
        label = resultado[0]['label']
        score = resultado[0]['score']

        if label == '5 stars' or (label == '4 stars' and sc
ore > 0.8):
            return "Alegre"
        elif label == '3 stars' or (label == '4 stars' and
score <= 0.8):
            return "Neutral"
        else:
            return "Melancólico"
    else:
        # Retorna un valor predeterminado si `resultado` es
None o vacío
        return "Estado de ánimo desconocido"

sentiment_analyzer = pipeline('sentiment-analysis', model
='nlptown/bert-base-multilingual-uncased-sentiment')
```

Explicación de los Algoritmos y Funcionamiento

1. **Clasificación Basada en Reglas (Palabras Clave)**:
    - La función empieza con una lista de palabras clave negativas, como "triste", "desanimado" o "angustiado". Si alguna de estas palabras aparece en la frase ingresada por el usuario (sin distinguir mayúsculas o minúsculas), la función automáticamente clasifica el estado de ánimo como "Melancólico".
    - Esta técnica es una forma de clasificación por reglas, donde ciertas palabras actúan como indicadores de estados de ánimo específicos. Aunque no es tan precisa como otros modelos, es rápida y asegura que casos claramente negativos sean reconocidos como "Melancólicos".
2. **Clasificación Basada en Análisis de Sentimientos**:
    - Si no se encuentran palabras clave negativas, la función utiliza un análisis de sentimientos más avanzado mediante el modelo `nlptown/bert-base-multilingual-uncased-sentiment`, que se ha cargado con la función `pipeline` de Hugging Face. Este modelo clasifica los sentimientos en

categorías de "estrellas" que van de 1 a 5, donde 5 representa el sentimiento más positivo y 1 el más negativo.

- La función evalúa el `label` (cantidad de estrellas) y el `score` (confianza del modelo en la clasificación) de `resultado`, que es un diccionario de salida del análisis de sentimientos. Con estos datos:
  - Si la clasificación es "5 stars" o "4 stars" con alta confianza (>0.8), el estado de ánimo se clasifica como "Alegre".
  - Si es "3 stars" o "4 stars" con menor confianza (≤0.8), se clasifica como "Neutral".
  - Para las clasificaciones más bajas (1 o 2 estrellas), el estado de ánimo se considera "Melancólico".
- Esta parte del código permite una clasificación más matizada, utilizando el análisis de sentimientos para una mayor precisión y diferenciación entre estados de ánimo.

3. **Resultado Predeterminado**:
   - Si el análisis de sentimientos (`resultado`) no está disponible o es vacío, la función devuelve "Estado de ánimo desconocido". Esto permite manejar casos en los que no se puede realizar un análisis de sentimientos, asegurando una salida controlada.

Detalles de Implementación

- **Modelo de NLP**: Se utiliza el modelo preentrenado `nlptown/bert-base-multilingual-uncased-sentiment`, especializado en análisis de sentimientos y adecuado para trabajar en varios idiomas. Este modelo proporciona una clasificación rápida y precisa para interpretar la frase del usuario.
- **Función `pipeline` de Hugging Face**: Facilita la creación de un "pipeline" de NLP, simplificando la carga y el uso de modelos complejos sin necesidad de configuraciones extensas. Esto hace que el programa sea más accesible y fácil de modificar si se cambia el modelo de NLP en el futuro.

Ejemplos de Uso

Descripción General

Este bloque de código proporciona ejemplos de cómo utilizar el clasificador de estado de ánimo desarrollado. La idea es verificar el funcionamiento del clasificador con diferentes frases de ejemplo que reflejan diversos estados emocionales. Cada frase pasa por el análisis de sentimientos del modelo, y luego se utiliza la función `clasificar_estado_animo_con_reglas` para determinar el estado de ánimo correspondiente.

Código

```
frase_usuario = "ha nacido mi hija"
resultado = sentiment_analyzer(frase_usuario)
estado_animo = clasificar_estado_animo_con_reglas(frase_usu
ario, resultado)
print(f"Estado de ánimo clasificado: {estado_animo}")

frase_usuario = "me siento bien"
resultado = sentiment_analyzer(frase_usuario)
estado_animo = clasificar_estado_animo_con_reglas(frase_usu
ario, resultado)
print(f"Estado de ánimo clasificado: {estado_animo}")

frase_usuario = "estoy enfermo"
resultado = sentiment_analyzer(frase_usuario)
estado_animo = clasificar_estado_animo_con_reglas(frase_usu
ario, resultado)
print(f"Estado de ánimo clasificado: {estado_animo}")
```

Explicación de los Pasos y Funcionamiento

1. **Frase de Entrada**:
    - Cada frase (`frase_usuario`) representa una declaración o expresión emocional del usuario.
    - Estas frases varían en contenido y tono, simulando diversas situaciones emocionales para verificar si el clasificador asigna el estado de ánimo adecuado.
2. **Análisis de Sentimientos**:

- Para cada frase, se ejecuta `sentiment_analyzer(frase_usuario)`, que llama al modelo de análisis de sentimientos de Hugging Face (`nlptown/bert-base-multilingual-uncased-sentiment`). Este modelo evalúa la frase y devuelve un `resultado` que contiene la clasificación en estrellas y la puntuación de confianza.
- El modelo utiliza técnicas avanzadas de procesamiento de lenguaje natural para interpretar el sentimiento de la frase y devolver una valoración que el clasificador usará.

3. **Clasificación del Estado de Ánimo**:
   - Con el `resultado` del análisis de sentimientos, se llama a la función `clasificar_estado_animo_con_reglas(frase_usuario, resultado)`.
   - La función aplica reglas basadas en palabras clave y el análisis de sentimientos para categorizar el estado de ánimo en "Alegre", "Neutral", o "Melancólico".
   - La salida final (`estado_animo`) representa el estado emocional de la frase analizada.
4. **Impresión del Resultado**:
   - Finalmente, `print(f"Estado de ánimo clasificado: {estado_animo}")` muestra el estado de ánimo clasificado para cada ejemplo. Esto facilita la verificación visual de que la función de clasificación asigna el estado de ánimo correcto a cada frase de prueba.

Ejemplo de Resultados Esperados

- Para la frase **"ha nacido mi hija"**, el clasificador debería reconocerla como un evento positivo y clasificar el estado de ánimo como "Alegre".
- En el caso de **"me siento bien"**, el sentimiento positivo debería llevar a una clasificación "Alegre" o "Neutral" según la puntuación de confianza.
- Para **"estoy enfermo"**, que sugiere malestar, el clasificador debería clasificar el estado de ánimo como "Melancólico".

Conclusión

Este bloque de ejemplos es crucial para evaluar la precisión y fiabilidad de la función de clasificación. Probar frases variadas permite detectar si el algoritmo identifica adecuadamente distintos estados emocionales, proporcionando una

base sólida para recomendaciones personalizadas según el estado de ánimo del usuario.

Configuración del Modelo de Embeddings

Descripción General

Este código configura el modelo de embeddings utilizando la biblioteca `SentenceTransformer`, que permite convertir textos en representaciones numéricas (embeddings). Estas representaciones vectoriales se emplean para calcular similitudes entre frases y realizar recomendaciones en función de las preferencias del usuario.

Código

```
# Cargar el modelo de embeddings
modelo_embeddings = SentenceTransformer('sentence-transform
ers/paraphrase-multilingual-MiniLM-L12-v2')
```

Explicación de los Elementos

1. **Modelo de Embeddings**:
    - El modelo especificado, `'sentence-transformers/paraphrase-multilingual-MiniLM-L12-v2'`, es una versión multilingüe de `SentenceTransformer`, entrenada para captar el significado de frases en varios idiomas.
    - Este modelo transforma frases en vectores de dimensiones consistentes, donde cada vector representa una frase con sus matices semánticos. Frases con significados similares tendrán embeddings similares (es decir, sus vectores estarán cerca en el espacio vectorial).
2. **Uso de Embeddings en Recomendaciones**:
    - En este proyecto, los embeddings se usan para medir la similitud semántica entre la frase de preferencia del usuario y las descripciones de opciones recreativas en los datasets (películas, libros y juegos de mesa).
    - Este proceso de comparación se realiza mediante métricas de distancia entre vectores, como la similitud de coseno. Al seleccionar las opciones con la mayor similitud con la frase del usuario, el sistema puede

proporcionar recomendaciones alineadas con el contexto o el tema deseado.

Ventajas del Modelo `paraphrase-multilingual-MiniLM-L12-v2`

- **Multilingüismo**: Este modelo puede manejar textos en varios idiomas, lo que lo hace adaptable si los datos o las preferencias del usuario no están exclusivamente en español.
- **Precisión en Parafraseo**: Dado que el modelo está entrenado en parafraseo, puede identificar frases similares en significado aunque utilicen palabras diferentes, mejorando la relevancia de las recomendaciones.
- **Optimización para Similaridad Semántica**: Este modelo es especialmente adecuado para tareas donde la similitud semántica es clave, como en recomendaciones personalizadas.

Carga de Datasets

Descripción General

En este bloque de código se realiza la carga de datos de tres fuentes: películas, juegos de mesa y libros. Las primeras dos fuentes están disponibles en archivos CSV, mientras que los datos de libros se obtienen mediante web scraping desde el sitio del Proyecto Gutenberg. Una vez extraídos y procesados los datos de cada fuente, se almacenan en estructuras de datos adecuadas para su posterior uso en el sistema de recomendaciones.

Código

```
# Cargar datasets de películas y juegos de mesa
df_peliculas = pd.read_csv('IMDB-Movie-Data.csv')
df_juegos = pd.read_csv('bgg_database.csv')
```

Explicación de los Elementos

1. **Cargar Datasets de Películas y Juegos**:
    - Los datasets `IMDB-Movie-Data.csv` (películas) y `bgg_database.csv` (juegos de mesa) se cargan directamente usando la función `pd.read_csv()`, que los convierte en `DataFrame`s de pandas. Estos `DataFrame`s se usarán para

buscar recomendaciones basadas en la preferencia temática ingresada por el usuario.

2. **Obtención de Datos de Libros mediante Web Scraping**:
   - Dado que el dataset de libros no se encuentra en un archivo CSV, se usa web scraping para extraer información directamente del sitio de Gutenberg.
3. **Extracción y Procesamiento de Datos del Proyecto Gutenberg**:
   - A través de `requests` y `BeautifulSoup`, el programa accede a la lista de los libros más populares y extrae la información detallada de cada uno, incluyendo título, autor, resumen y temas.
   - Para cada libro, el código sigue estos pasos:
     - Navega por la estructura HTML para acceder al listado de libros.
     - Para cada libro en la lista, construye la URL de la página individual del libro y realiza otra solicitud para obtener detalles adicionales (como el autor y el título).
     - Extrae la información relevante de las etiquetas HTML específicas y organiza esta información en un diccionario.
4. **Manejo de Datos Extraídos**:
   - La información de cada libro se guarda en una lista `libros`, donde cada elemento es un diccionario que representa un libro con su título, autor, resumen y temas.
   - Posteriormente, esta lista se convierte en un `DataFrame` de pandas `df_libros` para facilitar su manipulación en el proceso de recomendaciones.
5. **Guardado y Descarga del Dataset de Libros**:
   - El `DataFrame` resultante se guarda en un archivo `libros.csv` con `#` como separador, lo cual facilita la manipulación de datos textuales que puedan contener comas.
   - Se utiliza `files.download()` para descargar el archivo `libros.csv`, permitiendo que el usuario lo reutilice sin necesidad de ejecutar el proceso de scraping cada vez.

```
# Verificar y cargar el archivo libros.csv si ya existe
df_libros = pd.read_csv('libros.csv', sep='#')
```

Tratamiento de Datos Faltantes

1. **Verificación y Eliminación de Valores Nulos**:
   - Para evitar problemas en el proceso de recomendaciones, se verifica si existen valores nulos con `df_libros.isna().sum()`. Si se detectan valores nulos, se eliminan las filas correspondientes usando `df_libros.dropna(inplace=True)`.
   - La eliminación de filas con valores nulos asegura la consistencia y precisión de las recomendaciones.

Descripción del Código

El objetivo es calcular y almacenar embeddings de títulos para películas, juegos y libros, utilizando el modelo `all-mpnet-base-v2` de SentenceTransformer. Esto permite comparar las preferencias del usuario con recomendaciones, empleando la similitud de coseno entre los embeddings generados y la consulta del usuario.

Código Reorganizado y Correcciones

1. Cargar el Modelo de Embeddings

La línea de carga del modelo es correcta:

```
modelo_embeddings = SentenceTransformer('all-mpnet-base-v
2')
```

2. Definición de Funciones para Guardar y Cargar Embeddings

Las funciones `guardar_embeddings` y `cargar_embeddings` almacenan y recuperan embeddings en archivos `.pkl`, para que los datos ya procesados no se vuelvan a calcular innecesariamente.

3. Generar o Cargar Embeddings para Cada Dataset

La función `generar_o_cargar_embeddings` usa `guardar_embeddings` y `cargar_embeddings` para verificar si ya existen los embeddings en un archivo, en cuyo caso los carga; de lo contrario, los calcula y guarda.

Posible Error y Solución

Uno de los problemas frecuentes al ejecutar este código puede surgir de:

- **Estructura de Datos en `df_libros`**: Si `Subjects` o `Summary` contienen listas o valores `None`, podría fallar al tratar de convertir estos valores en strings concatenados.
- **Repetición Innecesaria de Embeddings**: Asegurar que solo se generan embeddings para los libros si la columna `Texto_completo_embedding` no existe.

Aquí está el código corregido y simplificado:

```
import os
import pickle
from sentence_transformers import SentenceTransformer

# Modelo de embeddings
modelo_embeddings = SentenceTransformer('all-mpnet-base-v
2')

# Función para guardar los vectores embeddings en un archiv
o .pkl
def guardar_embeddings(titulos, filename):
    vectores = [modelo_embeddings.encode(titulo) for titulo
in titulos]
    with open(filename, 'wb') as f:
        pickle.dump(vectores, f)

# Función para cargar los vectores embeddings desde un arch
ivo .pkl
def cargar_embeddings(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)

def generar_o_cargar_embeddings(df, columnas, filename):
    # Verifica si el archivo ya existe
```

```
    if not os.path.exists(filename):
        # Combina valores de las columnas especificadas
        titulos = df[columnas].astype(str).agg(' '.join, ax
is=1).tolist()
        guardar_embeddings(titulos, filename)
    return cargar_embeddings(filename)

# Cargar o crear embeddings para películas, juegos y libros
vector_peliculas = generar_o_cargar_embeddings(df_pelicula
s, ['Title', 'Genre', 'Description', 'Director'], 'vector_p
eliculas.pkl')
vector_juegos = generar_o_cargar_embeddings(df_juegos, ['ga
me_name', 'categories'], 'vector_juegos.pkl')
vector_libros = generar_o_cargar_embeddings(df_libros, ['Au
thor', 'Title', 'Summary', 'Subjects'], 'vector_libros.pk
l')

# Si la columna de embeddings completos no existe, la crea
if 'Texto_completo_embedding' not in df_libros.columns:
    df_libros['Texto_completo'] = df_libros[['Summary', 'Su
bjects']].fillna('').applymap(lambda x: ' '.join(x) if isin
stance(x, list) else str(x)).agg(' '.join, axis=1)
    df_libros['Texto_completo_embedding'] = df_libros['Text
o_completo'].apply(lambda x: modelo_embeddings.encode(x))

# Imprimir tamaños de los vectores para verificar
print("Tamaño de vector_peliculas:", len(vector_peliculas))
print("Tamaño de vector_juegos:", len(vector_juegos))
print("Tamaño de vector_libros:", len(vector_libros))
```

Explicación de Cambios

1. **Transformación de Datos en `df_libros`**:
    - Para `Summary` y `Subjects`, se aplica `fillna('')` para reemplazar valores `None` y asegurar que todos los valores se concatenen correctamente como strings.
    - `applymap(lambda x: ' '.join(x) if isinstance(x, list) else str(x))` se asegura de que los datos en formato de lista se conviertan en una cadena de

texto, especialmente en `Subjects`, que podría contener listas de temas.

2. **Uso Condicional de `Texto_completo_embedding`**:
    - Este bloque asegura que los embeddings para `Texto_completo` solo se calculen si no existen, evitando duplicación.

Resultado

Al ejecutar este código, deberías obtener los tamaños de `vector_peliculas`, `vector_juegos`, y `vector_libros`, asegurando que los embeddings se han calculado y cargado correctamente. Esta estructura optimiza tanto la eficiencia como la precisión en la generación de recomendaciones.

Modelo de Embeddings

Para la generación de embeddings, se utilizó el modelo `all-mpnet-base-v2` de la biblioteca **SentenceTransformers**. Este modelo está preentrenado para transformar texto en vectores de alta dimensión, manteniendo relaciones semánticas; es decir, textos con significados similares tienen embeddings cercanos en el espacio vectorial.

```
# Modelo de embeddings
modelo_embeddings = SentenceTransformer('all-mpnet-base-v
2')
```

Función para Guardar Embeddings

La función `guardar_embeddings` se encarga de crear embeddings a partir de una lista de títulos y guardarlos en un archivo `.pkl`. El propósito de esta función es evitar el cómputo repetitivo de embeddings para el mismo contenido, almacenándolos para que puedan ser reutilizados.

```
def guardar_embeddings(titulos, filename):
    vectores = [modelo_embeddings.encode(titulo) for titulo
in titulos]
    with open(filename, 'wb') as f:
        pickle.dump(vectores, f)
```

Función para Cargar Embeddings

La función `cargar_embeddings` permite recuperar los embeddings previamente guardados en un archivo `.pkl`. Esto hace que el programa sea más eficiente al evitar generar los mismos embeddings cada vez que se ejecuta.

```
def cargar_embeddings(filename):
    with open(filename, 'rb') as f:
        return pickle.load(f)
```

Función para Generar o Cargar Embeddings

La función `generar_o_cargar_embeddings` decide si es necesario crear nuevos embeddings o cargar los existentes. Primero, verifica si el archivo de embeddings existe; si no, concatena los valores de las columnas seleccionadas (por ejemplo, título, género, descripción) y genera los embeddings, que luego se guardan en el archivo especificado.

```
def generar_o_cargar_embeddings(df, columnas, filename):
    if not os.path.exists(filename):
        titulos = df[columnas].astype(str).agg(' '.join, ax
is=1).tolist()
        guardar_embeddings(titulos, filename)
    return cargar_embeddings(filename)
```

Carga o Creación de Embeddings para Películas, Juegos y Libros

Estas líneas de código llaman a `generar_o_cargar_embeddings` para crear o cargar los embeddings de películas, juegos, y libros. Los embeddings generados se almacenan en variables como `vector_peliculas`, `vector_juegos`, y `vector_libros` para su posterior uso en las recomendaciones.

```
vector_peliculas = generar_o_cargar_embeddings(df_pelicula
s, ['Title', 'Genre', 'Description', 'Director'], 'vector_p
eliculas.pkl')
vector_juegos = generar_o_cargar_embeddings(df_juegos, ['ga
me_name', 'categories'], 'vector_juegos.pkl')
vector_libros = generar_o_cargar_embeddings(df_libros, ['Au
```

```
thor', 'Title', 'Summary', 'Subjects'], 'vector_libros.pk
l')
```

Cálculo y Almacenamiento de Embeddings de Libros

Para los libros, además de los campos mencionados, se crea una columna de `Texto_completo` en el `DataFrame` combinando los campos `Summary` y `Subjects`. Esto asegura que todos los detalles relevantes de cada libro estén incluidos en la generación de embeddings. Solo si no se ha calculado el embedding previamente, se ejecuta la generación y almacenamiento.

```
if 'Texto_completo_embedding' not in df_libros.columns:
    df_libros['Texto_completo'] = df_libros['Summary'].fill
na('').apply(lambda x: ' '.join(x) if isinstance(x, list) e
lse str(x)) + " " + \\
                                df_libros['Subjects'].fil
lna('').apply(lambda x: ' '.join(x) if isinstance(x, list)
else str(x))
    df_libros['Texto_completo_embedding'] = df_libros['Text
o_completo'].apply(lambda x: modelo_embeddings.encode(x))
```

Verificación de los Tamaños de los Embeddings

Finalmente, se imprimen los tamaños de los vectores de embeddings para películas, juegos, y libros. Esto sirve para verificar que se han generado correctamente los embeddings de cada tipo de contenido.

```
print("Tamaño de vector_peliculas:", len(vector_peliculas))
print("Tamaño de vector_juegos:", len(vector_juegos))
print("Tamaño de vector_libros:", len(vector_libros))
```

Conclusión

El sistema de recomendación implementado utiliza embeddings de frases para representar de forma numérica la información semántica de cada tipo de contenido, lo que permite realizar recomendaciones basadas en similitudes. Este enfoque es eficaz en la búsqueda de elementos con significado similar al de una consulta del usuario, mejorando la precisión y eficiencia en las recomendaciones.

Identificación del Tipo de Contenido

La función `identificar_tipo_contenido` tiene el propósito de determinar el tipo de contenido que el usuario está buscando (película, libro, o juego de mesa) basándose en palabras clave presentes en la frase proporcionada.

1. Primero, la frase se convierte a minúsculas y se eliminan los acentos para evitar problemas de comparación.
2. Luego, se verifica la presencia de palabras clave como "pelicula", "libro", o "juego de mesa".
3. Si alguna palabra clave coincide, se devuelve el tipo correspondiente; en caso contrario, devuelve `None`.

Esta función es esencial para definir el tipo de contenido en base al cual se realizarán las recomendaciones personalizadas.

```
def identificar_tipo_contenido(frase_usuario):
    frase_sin_acentos = unidecode(frase_usuario.lower().str
ip())
    if "pelicula" in frase_sin_acentos:
        return "pelicula"
    elif "libro" in frase_sin_acentos:
        return "libro"
    elif "juego de mesa" in frase_sin_acentos:
        return "juego"
    return None
```

Búsqueda de Opciones Usando Embeddings

La función `buscar_opciones_embeddings` toma el tipo de contenido detectado y utiliza embeddings para encontrar las opciones más similares a la consulta del usuario. La función sigue los siguientes pasos:

1. **Concatenación de Campos de Interés**: Según el tipo de contenido, concatena en una columna (`Texto_completo`) los campos más relevantes para formar una descripción completa. Para:
    - **Películas**: usa título, descripción y género.
    - **Libros**: usa resumen y temas (subjects).

- **Juegos de mesa**: usa nombre del juego y categorías.

2. **Generación de Embeddings del DataFrame**: El contenido de la columna `Texto_completo` se transforma en embeddings utilizando el modelo de `SentenceTransformer`, generando un vector de características para cada fila del DataFrame.
3. **Generación de Embedding para la Frase del Usuario**: La frase proporcionada por el usuario también se convierte en un vector embedding, de modo que se pueda comparar semánticamente con las opciones del DataFrame.
4. **Cálculo de la Similitud del Coseno**: Se calcula la **similitud del coseno** entre el vector de la frase del usuario y cada vector del DataFrame. Este cálculo mide qué tan cercanos son dos vectores en términos de dirección, lo cual indica la similitud entre la consulta del usuario y cada opción disponible.
5. **Selección de las Opciones Más Similares**: Finalmente, se ordenan las opciones en función de su similitud con la frase del usuario y se devuelven las cinco opciones con mayor similitud. Estas son las recomendaciones que más se aproximan a la intención de búsqueda del usuario.

```
def buscar_opciones_embeddings(df, frase, tipo):
    # Concatenar los campos de interés según el tipo de con
tenido
    if tipo == "pelicula":
        df['Texto_completo'] = (
            df['Title'].fillna('') + " " +
            df['Description'].fillna('') + " " +
            df['Genre'].fillna('')
        )
    elif tipo == "libro":
        df['Texto_completo'] = (
            df['Summary'].fillna('') + " " +
            df['Subjects'].apply(lambda x: ' '.join(x) if i
sinstance(x, list) else x).fillna('')
        )
    elif tipo == "juego":
        df['Texto_completo'] = (
            df['game_name'].fillna('') + " " +
            df['categories'].apply(lambda x: ' '.join(x) if
```

```
isinstance(x, list) else str(x)).fillna('')
        )

    # Generar incrustaciones para el texto completo del Dat
aFrame
    incrustaciones_texto = modelo_embeddings.encode(df['Tex
to_completo'].tolist(), convert_to_tensor=True)

    # Generar incrustación para la frase ingresada
    frase_vector = modelo_embeddings.encode(frase, convert_
to_tensor=True)

    # Calcular la similitud del coseno entre la frase y las
opciones del DataFrame
    similitudes = util.cos_sim(frase_vector, incrustaciones
_texto)[0]

    # Devolver las 5 opciones más similares
    df['Similaridad'] = similitudes.cpu().numpy()
    return df.nlargest(5, 'Similaridad')
```

El uso de embeddings y la similitud del coseno son clave en este algoritmo, ya que permiten medir la afinidad semántica entre el texto ingresado por el usuario y las opciones almacenadas en el sistema, mejorando la precisión de las recomendaciones.

Flujo Principal del Programa

El flujo principal del programa, implementado en la función `flujo_principal`, guía al usuario a través de los pasos de clasificación de su estado de ánimo y selección de recomendaciones de contenido según sus preferencias. Este flujo integra las funcionalidades previas de análisis de sentimientos, identificación de tipo de contenido, y recomendación mediante embeddings. A continuación, se explica cada paso:

```
def flujo_principal():
    frase_usuario = input("¿Cómo te sientes hoy? ")
    resultado = sentiment_analyzer(frase_usuario)  # Analiz
ar la frase del usuario
```

```
    estado_animo = clasificar_estado_animo_con_reglas(frase
_usuario, resultado)  # Clasificar el estado de ánimo
    print(f"Estado de ánimo clasificado: {estado_animo}")

    # Ingreso de Preferencias
    frase_usuario = input("Ingresa una frase que describa l
a temática que te gustaría explorar: ")
    tipo_contenido = identificar_tipo_contenido(frase_usuar
io)

    # Búsqueda de Opciones
    if tipo_contenido == "pelicula":
        opciones = buscar_opciones_embeddings(df_peliculas,
frase_usuario, tipo_contenido)
        print("\\nOpciones de Películas:", opciones[['Titl
e', 'Description', 'Genre', 'Similaridad']])
    elif tipo_contenido == "libro":
        opciones = buscar_opciones_embeddings(df_libros, fr
ase_usuario, tipo_contenido)
        print("\\nOpciones de Libros:", opciones[['Title',
'Summary', 'Subjects', 'Similaridad']])
    elif tipo_contenido == "juego":
        opciones = buscar_opciones_embeddings(df_juegos, fr
ase_usuario, tipo_contenido)
        print("\\nOpciones de Juegos de Mesa:", opciones
[['game_name', 'categories', 'Similaridad']])
    else:
        print("No se reconoció el tipo de contenido. Por fa
vor, incluye 'película', 'libro' o 'juego de mesa' en tu fr
ase.")
```

1. **Análisis de Sentimientos**:
    - El flujo comienza solicitando al usuario que describa cómo se siente.
    - La frase ingresada se envía a la función `sentiment_analyzer`, que evalúa el tono emocional de la entrada y devuelve un valor que representa el sentimiento detectado.

- Este valor es luego usado por `clasificar_estado_animo_con_reglas` para asignar una clasificación más general de estado de ánimo, la cual es mostrada al usuario.

2. **Ingreso de Preferencias y Tipo de Contenido**:
   - El flujo solicita una segunda entrada en la que el usuario describe la temática o tipo de contenido que desea explorar.
   - Esta frase se procesa en `identificar_tipo_contenido`, que analiza palabras clave para definir si el usuario busca una **película**, **libro**, o **juego de mesa**.
3. **Búsqueda y Presentación de Opciones**:
   - Dependiendo del tipo de contenido, el flujo principal llama a la función `buscar_opciones_embeddings`, que realiza la búsqueda de recomendaciones basándose en embeddings (vectores de características semánticas) para identificar opciones similares a la descripción proporcionada.
   - La función devuelve una lista de las cinco recomendaciones más relevantes. La información de cada recomendación se presenta en un formato estructurado y ordenado según su similitud con la frase del usuario.
   - Si no se detecta un tipo de contenido, se le pide al usuario incluir una palabra clave específica en su frase para mejorar la identificación.

Conclusión Final del Flujo Principal

La función `flujo_principal` asegura una interacción fluida con el usuario, integrando diferentes componentes y algoritmos en un flujo continuo. La estructura permite:

- Un análisis de estado de ánimo inicial para captar el estado emocional del usuario,
- La identificación del tipo de contenido deseado, y
- La búsqueda personalizada de recomendaciones basadas en embeddings, las cuales proveen un resultado adaptado a las preferencias expresadas.

Este flujo permite personalizar las recomendaciones, ofreciendo una experiencia de usuario más precisa y relevante al integrar análisis semántico de preferencias y sentimientos.