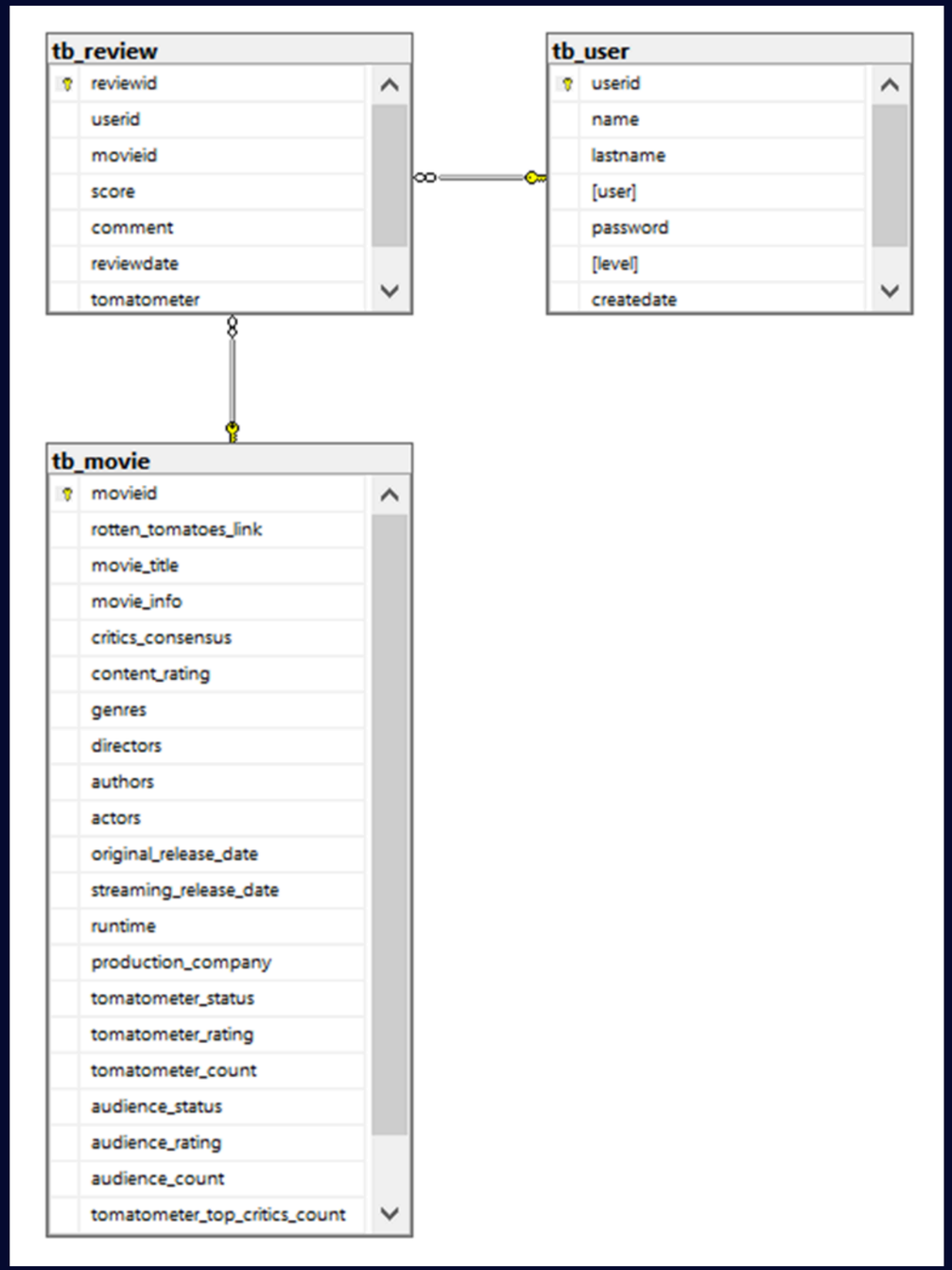
PROYECTO

Jose Giron - 1109419
Cristian Barrientos - 1114119
Melissa Mansilla - 1061719
Andrea Gudiel - 1115219

ENTIDAD RELACION PARA BD



IMPORTACION DE LIBRERIAS

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder, StandardScaler
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
import re
import nltk
```

Se importan diversas librerías que serán utilizadas para procesar y analizar datos. pandas y numpy son esenciales para manejar datos, sklearn para preprocesamiento, y nltk para el procesamiento de lenguaje natural.

DESCARGA DE RECURSOS NLTK

```
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
```

Se descargan recursos necesarios de la librería nltk como las stopwords, los tokenizadores y los lematizadores.

FUNCIÓN: CLEAN_REVIEW_SCORE

```python
def clean_review_score(value):
    try:
        if isinstance(value, str):
            if '/' in value:
                num, denom = value.split('/')
                denom_value = float(denom)
                if denom_value > 0:
                    return float(num) / denom_value
                else:
                    return 0
            elif re.match(r'^[A-F]$', value, re.IGNORECASE):
                return letter_grade_to_numeric(value)
            else:
                return float(value)
        return float(value)
    except ValueError:
        return np.nan
```

Esta función se encarga de limpiar y convertir las puntuaciones de reseñas a un formato numérico. Si el valor es una cadena que contiene una fracción, la convierte en un número decimal. Si el valor es una letra de calificación, lo convierte en un número usando la función letter_grade_to_numeric.

FUNCIÓN: LETTER_GRADE_TO_NUMERIC

Esta función convierte una letra de calificación a su equivalente numérico usando un diccionario de mapeo.

```
def letter_grade_to_numeric(grade):
    grade = grade.upper()
    mapping = {
        'A': 4.0,
        'B': 3.0,
        'C': 2.0,
        'D': 1.0,
        'F': 0.0
    }
    return mapping.get(grade, np.nan)
```

FUNCIÓN: FILL_MISSING_VALUES

Esta función llena los valores faltantes en una columna específica de un DataFrame con un valor dado.

```
def fill_missing_values(df, column, fill_value):
    df[column] = df[column].fillna(fill_value)
    return df
```

CLASE: DATANORMALIZER

```python
class DataNormalizer:
    def __init__(self, review_file):
        self.review_file = review_file
        self.label_encoders = {}
        self.numerical_columns = ['review_score']
        self.categorical_columns = ['review_type', 'critic_name']
        self.vocabulario = {}
        self.stop_words = set(stopwords.words('english'))
        self.lemmatizer = WordNetLemmatizer()
```

Esta clase se encarga de normalizar datos de reseñas. En el constructor, inicializa las variables y recursos necesarios, incluyendo stopwords y un lematizador.

MÉTODO:
LOAD_AND_CLEAN_REVIEWS_DATA

```
def load_and_clean_reviews_data(self):
    reviews_df = pd.read_csv(self.review_file)
    reviews_df = self.clean_reviews_data(reviews_df)
    return reviews_df
```

Este método carga los datos de reseñas desde un archivo CSV y luego los limpia usando el método clean_reviews_data.

MÉTODO: CLEAN_REVIEWS_DATA

```python
def clean_reviews_data(self, df):
    df = fill_missing_values(df, 'critic_name', 'Unknown Critic')
    df['review_score'] = df['review_score'].apply(clean_review_score)
    df['review_score'] = df['review_score'].fillna(df['review_score'].mean())
    return df
```

Este método limpia los datos de reseñas rellenando los valores faltantes en la columna 'critic_name' y convirtiendo las puntuaciones de las reseñas a un formato numérico.

MÉTODO: ENCODE_CATEGORICAL

```
def encode_categorical(self, df):
    for column in self.categorical_columns:
        le = LabelEncoder()
        df[column] = le.fit_transform(df[column].astype(str))
        self.label_encoders[column] = le
    return df
```

Este método codifica las columnas categóricas en valores numéricos usando LabelEncoder.

MÉTODO: PREPROCESS_REVIEWS

```python
def preprocess_reviews(self, df):
    df['review_content'] = df['review_content'].apply(lambda x: self.clean_text(x))

    comentarios_preprocesados = [self.clean_text(comentario) for comentario in df['r

    vocabulario = {word for comentario in comentarios_preprocesados for word in come
    self.vocabulario = {word: idx for idx, word in enumerate(vocabulario)}

    def text_to_numbers(comentario, vocabulario):
        return [vocabulario[word] for word in comentario.split() if word in vocabula

    comentarios_numericos = [text_to_numbers(comentario, self.vocabulario) for coment

    max_length = max(len(comentario) for comentario in comentarios_numericos)
    comentarios_vectores = np.zeros((len(comentarios_numericos), max_length), dtype=

    for i, comentario in enumerate(comentarios_numericos):
        length = min(len(comentario), max_length)
        comentarios_vectores[i, :length] = comentario[:length]

    return df, comentarios_vectores
```

Este método preprocesa los textos de las reseñas limpiándolos, construyendo un vocabulario y convirtiéndolos en secuencias numéricas de longitud fija.

MÉTODO: CLEAN_TEXT

```
def clean_text(self, text):
    if isinstance(text, str):
        tokens = word_tokenize(text)
        tokens = [word.lower() for word in tokens if word.isalpha()]
        tokens = [self.lemmatizer.lemmatize(word) for word in tokens if word not in s
        clean_text = ' '.join(tokens)
        return clean_text
    else:
        return ''
```

Este método limpia un texto eliminando caracteres no alfabéticos, convirtiendo a minúsculas, eliminando stopwords y aplicando lematización.

MÉTODO: PREPARE_DATA

```
def prepare_data(self):
    df = self.load_and_clean_reviews_data()
    df = self.encode_categorical(df)
    df, review_sequences = self.preprocess_reviews(df)

    X = review_sequences
    y = df['review_type'].values
    return X, y
```

Este método prepara los datos cargándolos, limpiándolos, codificando las columnas categóricas y preprocesando las reseñas para convertirlas en secuencias numéricas.

MÉTODO: PREPROCESS_SINGLE_REVIEW

```python
def preprocess_single_review(self, review_text):
    clean_review = self.clean_text(review_text)
    review_numbers = [self.vocabulario.get(word, 0) for word in clean_review.split()]
    max_length = max(len(review_numbers), 256)
    review_vector = np.zeros((1, max_length), dtype=int)
    review_vector[0, :len(review_numbers)] = review_numbers[:max_length]
    return review_vector
```

Este método preprocesa una única reseña convirtiéndola en una secuencia numérica de longitud fija, utilizando el vocabulario construido previamente.

IMPORTACION DE LIBRERIAS

Se importan librerías esenciales para construir y entrenar el modelo de sentimiento. numpy se utiliza para operaciones numéricas, tensorflow y keras para construir el modelo de red neuronal, matplotlib para graficar, y DataNormalizer para normalizar los datos.

```
import numpy as np
import tensorflow as tf
from tensorflow import keras
import matplotlib.pyplot as plt
from DataNormalizer import DataNo
```

CLASE: SENTIMENTMODEL

El constructor de la clase SentimentModel inicializa el tamaño del vocabulario, crea un modelo secuencial de keras y establece el normalizador de datos en None.

```
class SentimentModel:
    def __init__(self, vocab_size):
        self.vocab_legth = vocab_size
        self.model = keras.Sequential()
        self.data_normalizer = None
```

MÉTODO: TRAIN

```
def train(self, review_file):
    vocab_size = self.vocab_legth
    self.model.add(keras.layers.Embedding(vocab_size, 16))
    self.model.add(keras.layers.GlobalAveragePooling1D())
    self.model.add(keras.layers.Dense(16, activation=tf.nn.relu))
    self.model.add(keras.layers.Dense(1, activation=tf.nn.sigmoid))
    self.model.summary()
    self.model.compile(optimizer='adam',
                loss='binary_crossentropy',
                metrics=['acc'])

    self.data_normalizer = DataNormalizer(review_file)
    X, y = self.data_normalizer.prepare_data()

    X = np.clip(X, 0, vocab_size - 1)

    x_val = X[:vocab_size]
    partial_x_train = X[vocab_size:]
    y_val = y[:vocab_size]
    partial_y_train = y[vocab_size:]

    history = self.model.fit(partial_x_train,
                        partial_y_train,
                        epochs=20,
                        batch_size=512,
                        validation_data=(x_val, y_val),
                        verbose=1)
    return history
```

Este método entrena el modelo de sentimientos. Primero, configura el modelo agregando capas de embedding, pooling, y densas con activaciones relu y sigmoid. Luego, compila el modelo con el optimizador adam y la función de pérdida binary_crossentropy.

A continuación, crea un objeto DataNormalizer y prepara los datos de reseñas. Los datos de entrada se ajustan al tamaño del vocabulario.

Se divide el conjunto de datos en un conjunto de validación y un conjunto de entrenamiento parcial. Finalmente, el modelo se entrena durante 20 épocas con un tamaño de lote de 512, usando los datos de entrenamiento y validación.

MÉTODO: PREDICT_REVIEW

```
def predict_review(self, review_text):
    if not self.data_normalizer:
        raise ValueError("The model must be trained before making predictions.")
    review_vector = self.data_normalizer.preprocess_single_review(review_text)
    prediction = self.model.predict(review_vector)
    return 'FRESH' if prediction < 0.5 else 'ROTTEN'
```

Este método predice el sentimiento de una reseña. Si el modelo no ha sido entrenado, lanza un error. Primero, normaliza la reseña usando el método preprocess_single_review del DataNormalizer. Luego, realiza la predicción y devuelve 'FRESH' si la predicción es menor a 0.5, y 'ROTTEN' en caso contrario.

CONCLUSIÓN

La aplicación comienza con la importación de librerías esenciales como Flask, pyodbc y TensorFlow, que proporcionan la base para el desarrollo web, la conexión a bases de datos y la implementación de modelos de aprendizaje profundo. Se define y entrena un modelo de análisis de sentimiento utilizando datos de reseñas de películas, lo que permite predecir la polaridad de los comentarios (positivos o negativos) y agrega valor a las funcionalidades de la aplicación.

La aplicación Flask se estructura con múltiples rutas para manejar el registro y autenticación de usuarios, la creación y obtención de reseñas, y la administración de datos de usuarios y películas. Utilizando pyodbc, la aplicación se conecta a una base de datos SQL Server para gestionar eficientemente la información. Diversas funciones de utilidad se implementan para limpiar y normalizar datos, manteniendo el código organizado y modular, y asegurando una experiencia de usuario integrada y fluida.