Universidad Nacional de Rosario

Facultad de Ciencias Exactas, Ingeniería y Agrimensura

Tecnicatura Universitaria en Inteligencia Artificial

Trabajo Práctico Parte 2 - Procesamiento del Lenguaje Natural

Docentes:

Juan Pablo Manson

Alan Geary

Constantino Ferruci

Dolores Sollberger

Arce Sofía - A4517/9

21 de mayo 2025

Resumen

En esta etapa del trabajo práctico, se consolidaron los datos del repositorio del juego Cascadia para realizar los ejercicios.

Desarrollo e implementación

Dentro del archivo main.ipynb se incluyeron los resultados de cada ejercicio
Para la resolución de los ejercicios creé una clase llamada TextoParaAnalizar con los siguientes métodos:

- __init__(self, text): El constructor de la clase. Recibe la ruta al archivo de texto como argumento. Realiza una validación para asegurarse de que el archivo existe.
- __str__(self): Permite que el objeto de la clase se comporte como la cadena de texto que contiene. Devuelve el contenido completo del archivo de texto.
- fragmentos(self, fragmento="palabra"): Divide el texto en fragmentos.
    - Si fragmento es "oracion", devuelve una lista de oraciones (líneas no vacías después de eliminar ciertos caracteres de puntuación).
    - Si fragmento es "palabra" (el valor por defecto), devuelve una lista de palabras.
- vectorizar(self, tipo: str = "sentencetransformer"): Vectoriza los fragmentos del texto (por defecto, oraciones) utilizando un modelo de embedding especificado.
    - tipo="CountVectorizer": Utiliza CountVectorizer para crear representaciones basadas en la frecuencia de palabras.
    - tipo="sentencetransformer" (valor por defecto): Utiliza un modelo de sentence-transformers (específicamente 'all-MiniLM-L6-v2') para generar embeddings contextuales de las oraciones.
    - Devuelve el DataFrame con los vectores y el modelo utilizado.
- extraer_sustantivos(self): Extrae sustantivos y entidades con nombre (NER) del texto utilizando el modelo de spaCy 'en_core_web_sm'.
    - Devuelve un diccionario donde las claves son los sustantivos/entidades encontrados y los valores son sus etiquetas NER (si se identificaron como entidad) o una cadena vacía.

Ejercicio 2

Cargo el manual del juego guardado en cascadia_manual.txt creando una instancia de TextoParaAnalizar. Luego vectorizo los fragmentos con el modelo de sentence transformer. Calculo la similitud del coseno entre cada frase del ejemplo que antes vectoricé y encuentro aquellas 3 primeras que están más cercanas en el espacio. Al generar los embeddings con el modelo de sentence transformer, el significado de las palabras se tiene en cuenta a la hora de generar el embedding y es por eso que la similitud del coseno funciona mejor que la similitud de Jaccard calculada en las celdas siguientes.

Ejercicio 3

Para la resolución de este ejercicio opté por usar el texto de las reviews y creé una instancia de la misma clase usada en el ejercicio anterior. Vectorizo los fragmentos y uso spaCy para identificar los sustantivos y hallar entidades del mundo real con nombre. Luego genero un dataframe con los sustantivos extraídos y si son sustantivos compuestos como por ejemplo “card’s games” se genera un vector para ambas palabras.

Ejercicio 4

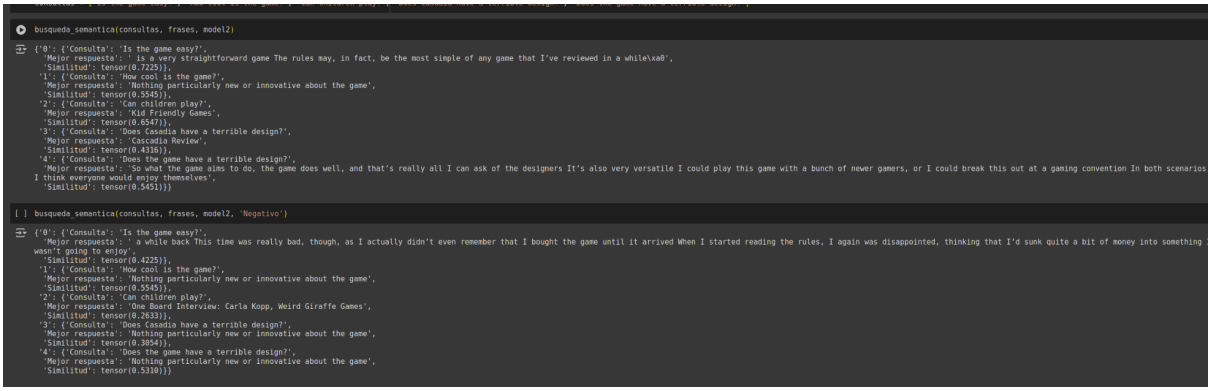
En este ejercicio creo la función detect_language que toma la ruta de un archivo de texto como entrada, lee su contenido y uso la biblioteca langdetect para determinar el idioma. Itero por cada archivo en el directorio y obtengo su idioma. Luego creo un DataFrame a partir de esta lista, con columnas "Nombre del archivo" e "Idioma"..

Ejercicio 5

Comienzo cargando un modelo pre-entrenado de análisis de sentimiento (nlptown/bert-base-multilingual-uncased-sentiment) y su tokenizador asociado utilizando la biblioteca transformers. El motivo de la elección del modelo es que es compatible con las demas versiones de las librerías utilizadas en los ejercicios anteriores.

Creo un pipeline de análisis de sentimiento con el modelo y tokenizador cargados con dos funciones:

- analisis_de_sentimiento: Descarto aquellos fragmentos que tienen una longitud menor a 5 palabras porque hay algunos fragmentos que no tienen información relevante, debido a que la fragmentación del texto no tiene en cuenta donde empieza y donde termina una “review”. Luego con el modelo predigo el sentimiento del fragmento y clasifico los fragmentos en categorías (Negativo, Neutral, Positivo) basándose en las "estrellas" predichas por el modelo.
- busqueda_semantica: Para cumplir con la consigna de filtrar por sentimiento se hace una búsqueda de similitud del coseno entre consultas y respuestas (fragmentos) y si se le pasa como argumento el sentimiento a buscar se llama la función de analisis_de_sentimiento y busca solo dentro de fragmentos con un sentimiento específico (Negativo, Neutral o Positivo).



En este ejemplo se puede observar como con la primer oración sin filtrar por sentimiento se obtiene una con una similitud de 0.7225 clasificada como positiva, pero al filtrar solamente por sentimientos negativos la primer respuesta cambia totalmente y ahora se obtiene una con 0.4225. Otra observación es que cuando los fragmentos de consulta contienen el nombre del juego (Cascadia) los vectores a responder no son tan informativos como cuando se formula la pregunta con “el juego” en lugar del nombre.

Ejercicio 6

Este ejercicio consiste de 3 partes:

1. **Creación del Dataset:**
    - Generé un conjunto de datos con más de 300 preguntas.
    - Estas preguntas se obtuvieron con la ayuda de ChatGPT, solicitando preguntas relacionadas con diferentes aspectos de juegos (información general, estadísticas y personas/relaciones).
    - Me aseguré de que el dataset estuviera balanceado, con aproximadamente 110 preguntas por cada una de las tres categorías ("Estadísticas", "Información", "Relaciones").
    - El dataset fue almacenado en un archivo CSV (tp-nlp-preguntas - preguntas_personas_110.csv).
2. **Preparación de Datos:**
    - Creé los embeddings con el modelo de sentence transformers usado en los ejercicios anteriores.
    - Cargué el archivo en un DataFrame.
    - Dividí el dataset en conjuntos de entrenamiento (80%) y prueba (20%) utilizando train_test_split de scikit-learn.
3. **Entrenamiento y Evaluación de Modelos de Clasificación:**
    - Eegí dos modelos de clasificación:
        - **Regresión Logística (LogisticRegression):** U
        - **Máquinas de Vectores de Soporte (Support Vector Machines - SVM):**

Resultados y Análisis:

- **Regresión Logística:** Se obtuvo un *score* (accuracy) de 0.68 y analicé la matriz de confusión. La observación inicial sugiere que el modelo tiene cierta capacidad para clasificar las consultas, pero puede presentar confusiones entre las categorías, particularmente entre "Información" y "Relaciones".

- **Support Vector Machines (SVM):** Los resultados mostraron un *Accuracy* significativamente mayor (80%) en comparación con la Regresión Logística, lo que indica que SVM es más efectivo para este problema de clasificación. La matriz de confusión para SVM reveló que, si bien aún existen algunas confusiones entre "Información" y "Relaciones", las confusiones con las otras clases son menores que en el modelo de Regresión Logística.

  Cabe resaltar que al generar el dataset con Chat GPT algunas preguntas tienen cierta ambigüedad inherente, es decir, que podrían ser respondidas por más de una categoría de fuente de datos, podría ser un factor que contribuye a los errores de clasificación, especialmente entre las categorías "Información" y "Relaciones".