Universidad Nacional de Rosario
Facultad de Ciencias Exactas, Ingeniería y Agrimensura
T.U.I.A

Trabajo Práctico 1 - Segunda Parte

Procesamiento del Lenguaje Natural

Integrante:
Masciangelo, Lucía

2025

Informe del Trabajo Práctico 1 - Segunda Parte

Introducción

Este trabajo práctico se centra en la aplicación de diversas técnicas de procesamiento del lenguaje natural (NLP) para resolver tareas reales sobre un corpus textual, extraído de un repositorio que contiene información acerca de un juego de mesa llamado “Cascadia” generado por compañeros. A lo largo de los ejercicios se aplicaron métodos de embeddings, análisis de similitud, POS y NER tagging, detección de idioma, análisis de sentimientos, y clasificación semántica de preguntas.
El desarrollo fue realizado íntegramente en Google Colab, con apoyo en bibliotecas como spaCy, scikit-learn, transformers, nltk, gensim, y pandas.

A continuación procederé a explicar ejercicio por ejercicio, decisiones tomadas y conclusiones:

Ejercicio 2 – Vectorización y Búsqueda Semántica

Objetivo: Dividir un texto largo en fragmentos, vectorizarlos, y luego realizar búsquedas semánticas sobre esos fragmentos comparando diversas técnicas de similitud.

Desarrollo:

1) Para comenzar, se hizo un análisis exploratorio de los archivos ubicados en la carpeta de información. Se detectó cuántos caracteres tenía cada uno para elegir el texto más largo como base del análisis. El resultado fue el siguiente:
    - cascadia_manual.txt: 39301 caracteres
    - boardgamereview_review.txt: 17695 caracteres
    - whatboardgame_review.txt: 13071 caracteres
    - oneboardfamily_review.txt: 10430 caracteres
    - bluehighwaygames_description.txt: 9276 caracteres
    - flatout_games.txt: 2499 caracteres

- board_game_co_uk_guide.txt: 909 caracteres

Como era de esperarse, el manual del juego (cascadia_manual.txt) resultó ser el más extenso, así que se lo utilizó como fuente principal para este ejercicio.

2) El texto se segmentó en fragmentos utilizando **spaCy**. La lógica de segmentación fue por bloques de 4 oraciones con un solapamiento de 1 oración entre bloques. Esta decisión se tomó para mantener el contexto sin necesidad de un segmentador más complejo, ya que el documento es un manual técnico que trata siempre sobre el mismo tema.

3) Cada fragmento fue vectorizado utilizando dos modelos distintos evaluando el desempeño de cada uno:
   - **BERT** (desde transformers): genera embeddings palabra por palabra teniendo en cuenta el contexto cercano, lo que puede resultar útil en algunas tareas específicas.
   - **Sentence-BERT** (sentence-transformers): optimizado para capturar relaciones semánticas a nivel de oración o texto completo. Considera toda la frase como una unidad, lo que resulta clave en tareas de búsqueda semántica.

4) Se definieron cinco frases de consulta diseñadas manualmente para evaluar la calidad semántica de los fragmentos devueltos por cada modelo. Luego, se aplicó la búsqueda semántica usando las representaciones vectoriales de cada modelo mencionado en el punto 3.

   Resultados con BERT:

	Query	Fragmento más similar	Coseno (TF-IDF)	Jaccard (sets)	Dice	Levenshtein	Jaro-Winkler
0	The wildlife scoring system awards points base...	Whenever you fill in a shape, fill in the next...	0.106652	0.046154	0.088235	398	0.579876
1	Tile placement must follow adjacency rules acc...	Tiles are included in a contiguous habitat cor...	0.059540	0.055556	0.105263	492	0.555024
2	The game progresses over a fixed number of tur...	Scenarios can be played in multi-player or sol...	0.070639	0.044776	0.085714	334	0.582945
3	Players are not allowed to exchange tiles or t...	10\n\n \n▲WILDLIFE SCORING CARDS\n\nA\n\nC\n\n...	0.080014	0.060606	0.114286	409	0.557346
4	Final points are tallied by summing terrain bo...	10\n\n \n▲WILDLIFE SCORING CARDS\n\nA\n\nC\n\n...	0.050144	0.030769	0.059701	408	0.551471

   Los resultados con BERT mostraron limitaciones a la hora de capturar el significado global del fragmento. Al operar a nivel de palabra, pierde contexto cuando se combinan los embeddings. De hecho en las dos últimas frases devuelve como fragmento más similar el mismo, cuando las oraciones tienen temas diferentes.

Resultados con S-BERT:

	Query	Fragmento más similar	Coseno (TF-IDF)	Jaccard (sets)	Dice	Levenshtein	Jaro-Winkler
0	The wildlife scoring system awards points base...	End Game & Scoring: Follow scoring on pages 8-...	0.215013	0.076923	0.142857	232	0.603113
1	Tile placement must follow adjacency rules acc...	If you do not have any \nremaining Nature Toke...	0.249619	0.104478	0.189189	601	0.548628
2	The game progresses over a fixed number of tur...	The game ends when there are no more Habitat T...	0.159098	0.085366	0.157303	596	0.555673
3	Players are not allowed to exchange tiles or t...	Note: when replacing tiles and tokens in the p...	0.220242	0.128205	0.227273	145	0.628344
4	Final points are tallied by summing terrain bo...	End Game & Scoring: Follow scoring on pages 8-...	0.111849	0.037736	0.072727	232	0.588899

En cambio, S-BERT obtuvo mejores resultados en todos los casos. Las frases recuperadas fueron más relevantes semánticamente y mejor alineadas con el estilo y el contenido del texto original.

Comparar ambos métodos dejó claro que vectorizar palabra por palabra (como en el caso de BERT clásico) y luego combinar no solo es menos eficiente, sino que también es más costoso en términos de cómputo. Además, no alcanza la precisión que ofrece un modelo entrenado directamente para la tarea, como es el caso de S-BERT.

5) A nivel de evaluación de distancias, se utilizaron cinco métricas clásicas para comparar similitudes entre la query y cada fragmento: Coseno, Jaccard, Dice, Levenshtein y Jaro-Winkler.

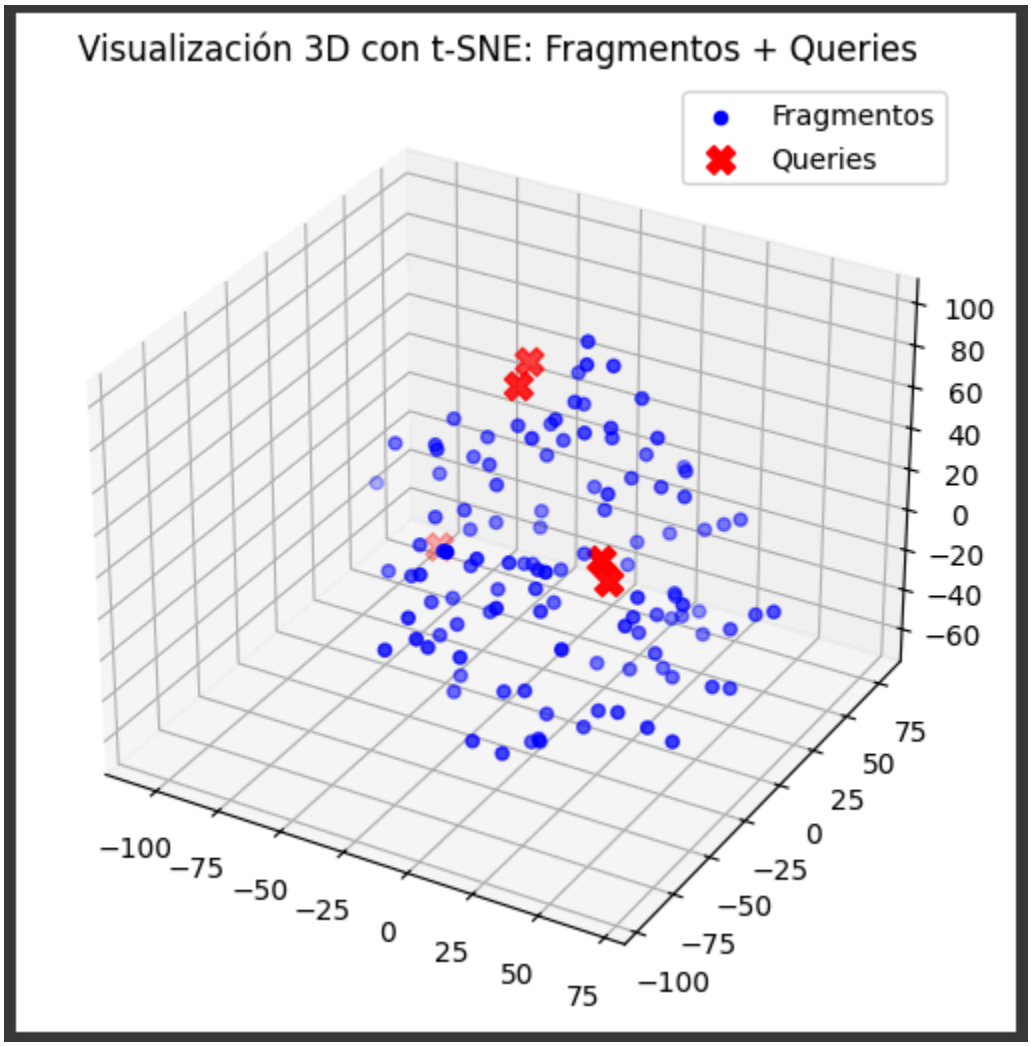
   - Dentro del análisis con S-BERT, se destacó especialmente la métrica **Jaro-Winkler**, que mostró valores sistemáticamente más altos. Esto se debe a que esta métrica pondera los prefijos comunes entre cadenas, por lo que funciona muy bien cuando las frases comparten estructura gramatical o palabras iniciales similares.

   - **Levenshtein** también se comportó bien, capturando errores tipográficos o variantes leves en las palabras.

Visualización:

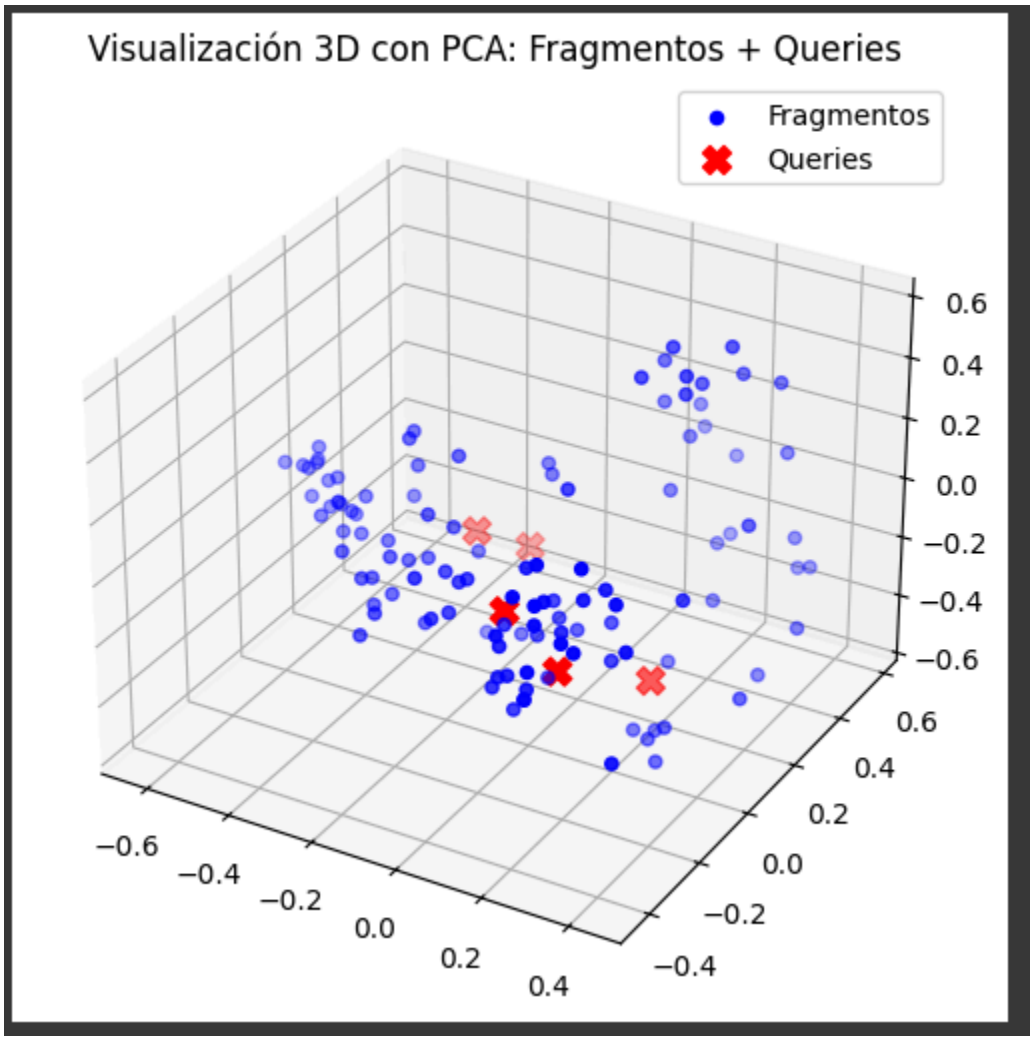
Para analizar si las representaciones vectoriales generadas por S-BERT realmente agrupan de forma semántica los fragmentos del manual, se aplicaron dos técnicas de reducción de dimensionalidad: **t-SNE** y **PCA**, ambas proyectando los embeddings a un espacio tridimensional.

- Visualización con t-SNE:



Se observa que las queries (en rojo) se ubican cerca de varios grupos de fragmentos (en azul). Esto es esperable, ya que t-SNE prioriza preservar las vecindades locales, lo que significa que si dos puntos están cerca en el espacio original de alta dimensión, deberían mantenerse cerca en la proyección. Esto refuerza que las consultas y los fragmentos están efectivamente alineados semánticamente dentro del espacio vectorial generado por S-BERT.

- Proyección con PCA:



Sí preserva la estructura lineal de los datos, también se ve que las queries caen dentro de zonas densas del espacio de fragmentos. Aunque PCA no captura relaciones no lineales como t-SNE, sigue siendo útil para validar que las queries comparten componentes principales con fragmentos relevantes.

Ambas visualizaciones, aunque desde enfoques distintos, muestran una distribución coherente entre las consultas y los textos, lo cual valida que S-BERT genera embeddings adecuados para tareas de búsqueda semántica, incluso cuando se los proyecta en espacios reducidos.

Conclusión:

A lo largo del ejercicio se compararon diferentes modelos de embeddings y métricas de similitud para evaluar la capacidad de recuperar fragmentos semánticamente relevantes frente a una consulta. El modelo S-BERT demostró ser el más adecuado para esta tarea, ya que genera representaciones que capturan mejor el significado global de cada fragmento, permitiendo ubicar consultas dentro de regiones temáticamente coherentes.

Entre las métricas evaluadas, **Jaro-Winkler** se destacó al reforzar coincidencias estructurales y léxicas, lo que complementa muy bien la representación semántica de **S-BERT**. En cambio, métricas como **Coseno** o **Jaccard** resultaron menos consistentes, especialmente cuando el texto a comparar tenía estructuras diferentes aunque tratara el mismo tema.

En resumen, la combinación de **S-BERT con la distancia Jaro-Winkler** ofrece una solución robusta para tareas de búsqueda semántica, sobre todo cuando se trabaja con textos técnicos o estructurados, como es el caso del manual del juego Cascadia.

Ejercicio 3 – POS, NER y Búsqueda por Sustantivos

Objetivo: Extraer sustantivos desde un texto utilizando POS tagging, agrupar entidades nombradas correctamente y evaluar cómo se comportan distintas métricas de similitud frente a consultas con errores ortográficos, comparándolas contra los sustantivos extraídos.

Desarrollo:

1) En este caso, se eligió trabajar con un texto distinto al del ejercicio anterior para no repetir siempre el mismo corpus. Se utilizó el segundo archivo más largo: 'boardgamereview_review.txt'
   Este archivo contenía mucho ruido, ya que provenía de una extracción directa de una página web e incluía etiquetas HTML, navegación del sitio y expresiones irrelevantes que no formaban parte de la reseña real.
   Para resolver esto, se realizó una limpieza manual, identificando claramente el comienzo y el fin del texto de la review propiamente dicha, y eliminando fragmentos innecesarios. Esta depuración fue clave para poder trabajar con un contenido que realmente tuviera valor semántico

para el análisis posterior.

2) Una vez que el texto quedó limpio, se lo segmentó en oraciones individuales utilizando **pySBD**, ya que en este ejercicio se decidió trabajar a nivel oración (no por fragmentos como en el Ejercicio 2). La idea fue analizar cada oración como unidad mínima y extraer sus sustantivos de manera aislada.

3) A cada oración se le aplicó **POS tagging** con spaCy, seleccionando únicamente aquellas palabras etiquetadas como **NOUN**, **PROPN**. Esto permitió enfocarse únicamente en los sustantivos o frases sustantivas de cada oración, dejando afuera conectores o palabras funcionales que no aportaban información relevante.

4) También se aplicó **NER (Named Entity Recognition)** para identificar entidades nombradas dentro del texto. Sin embargo, al inspeccionar los resultados, se detectó un problema: varias entidades compuestas como “Board Game Review” o “Alderac Entertainment Group” estaban fragmentadas, apareciendo palabra por palabra en filas distintas. Esto dificultaba tanto la interpretación como el análisis semántico posterior, ya que cada una de las palabras representaba la misma entidad.
Para resolverlo, se desarrolló un proceso de agrupación donde se identificaron las entidades completas y se unificaron en una sola fila por entidad, quedando el dataset en un formato mucho más ordenado y útil.

5) A continuación, se cambió el enfoque del análisis. En lugar de trabajar con queries semánticas, se seleccionaron dos palabras **intencionalmente mal escritas**: "**Cascadie**" y "**Cascadis**" (variantes con errores de la palabra “Cascadia”) para simular errores tipográficos comunes en búsquedas reales.

   La idea fue comparar cómo se comportaban distintas métricas de distancia frente a estas palabras incorrectas, al ser comparadas contra todos los sustantivos extraídos previamente del texto.

6) Para cada una de las dos palabras con error, se calculó la similitud con todos los sustantivos utilizando las siguientes métricas:

   - **Similitud del coseno** (con TF-IDF)
   - **Jaccard**

- Dice
- Distancia de Levenshtein
- Jaro-Winkler

Resultados:

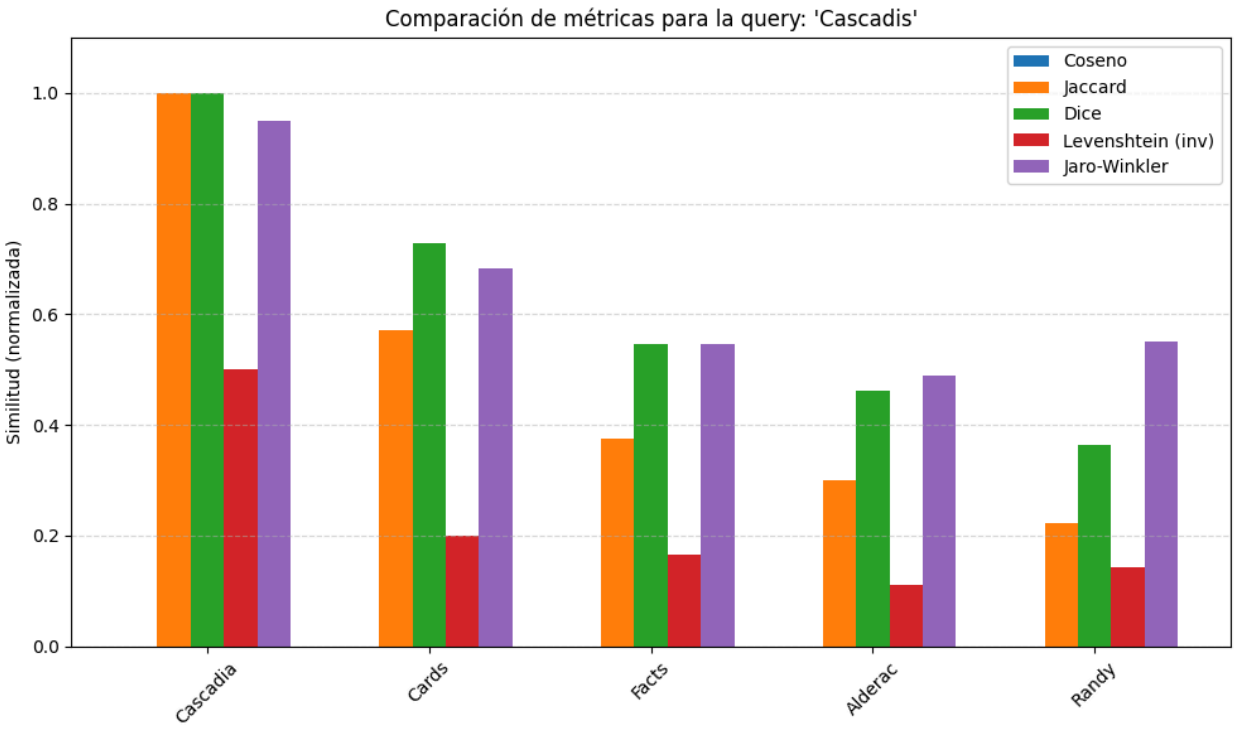
- En ambos casos, la **métrica del coseno** devolvió cero para todas las comparaciones. Esto es esperable, ya que TF-IDF trabaja con coincidencias exactas de tokens y no maneja errores ortográficos.

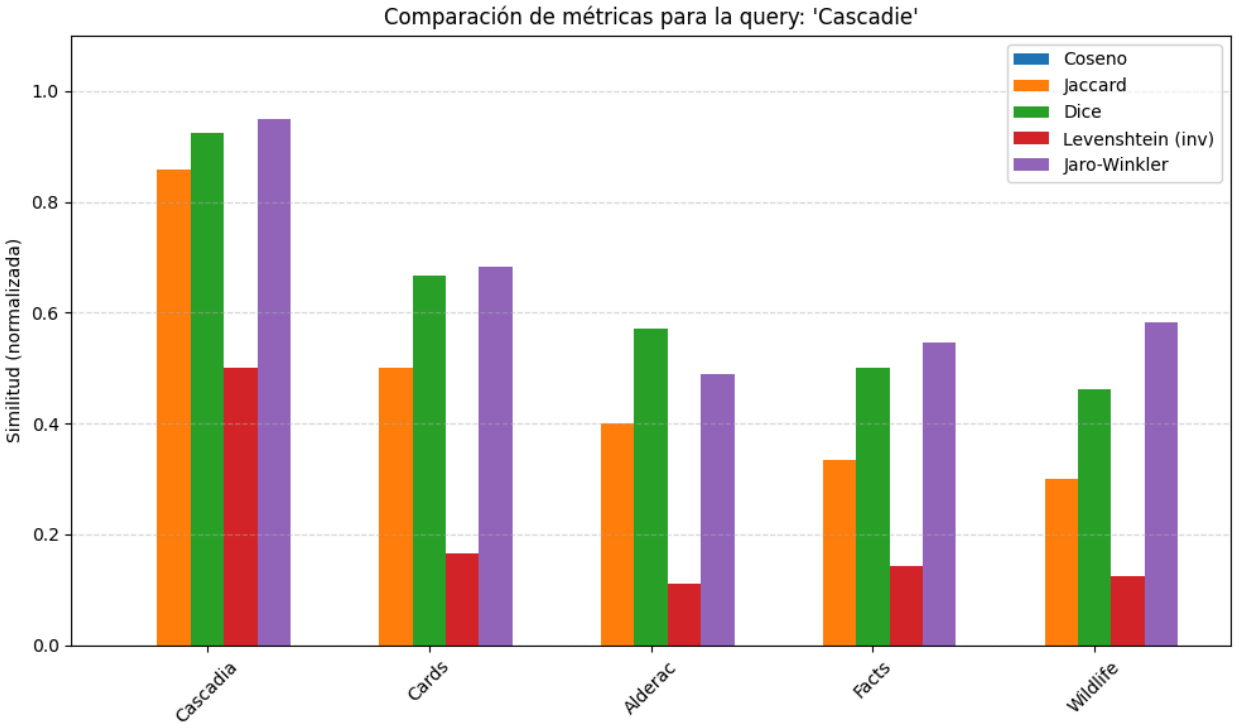
- Las métricas **Jaccard** y **Dice**, que operan a nivel de coincidencia de letras, devolvieron valores relativamente altos cuando había una gran superposición de caracteres. Por ejemplo, al comparar con “Cascadia”, "Cascadis" obtuvo 1.0 en ambas, mientras que "Cascadie" dio **0.8571** y **0.9231** respectivamente. Sin embargo, estas métricas no siempre reflejan bien la estructura o el orden de la palabra, por lo que pueden sobrestimar la similitud.

- La **distancia de Levenshtein** mostró buen rendimiento en ambos casos, devolviendo **una distancia de edición de 1** frente a “Cascadia”, lo que refleja correctamente que solo se requiere un cambio para corregir la palabra. Aun así, como devuelve una distancia y no una similitud, requiere interpretación adicional o normalización para hacerla comparable. Ésta distancia, cuanto más baja mejor.

- La métrica **Jaro-Winkler** fue la que mejor capturó tanto la similitud fonética como estructural. En ambos casos, "Cascadis" y "Cascadie" obtuvieron un **0.95 de similitud** con “Cascadia”, reconociendo correctamente que se trata de errores mínimos. Esta métrica pondera especialmente los **prefijos compartidos**, lo cual es ideal cuando los errores ocurren al final, como en este caso.

A continuación se muestran **dos gráficos comparativos** donde se visualizan las **cinco palabras más similares** según la métrica **Jaro-Winkler**, (que definimos que es la mejor en términos reales de errores mínimos), tanto para “Cascadis” como para “Cascadie”. Estas visualizaciones permiten contrastar directamente qué palabras fueron consideradas cercanas por cada métrica, y cómo varían los valores obtenidos en cada una.





Conclusión:

Este ejercicio sirvió para evaluar qué tan bien responden diferentes métricas

de similitud textual cuando se enfrentan a **palabras con errores ortográficos leves**, algo muy común en sistemas reales de búsqueda o input manual.

La extracción previa de sustantivos y la consolidación de entidades permitieron tener un corpus limpio y enfocado para realizar las comparaciones. Luego, al introducir variantes incorrectas de "Cascadia", se comprobó que:

- **Jaro-Winkler** es la métrica más adecuada para este tipo de tareas: reconoce correctamente similitudes fonéticas, errores de tipeo al final de las palabras y da valores altos cuando hay coincidencia de prefijos, lo que tiene mucho sentido en nombres propios o términos específicos.

- **Levenshtein** también mostró buen desempeño, aunque requiere normalización para ser comparado fácilmente con otras métricas.

- **Dice** y **Jaccard** aportaron valores razonables, pero no siempre distinguen entre estructuras reales y coincidencias accidentales de caracteres.

- **Coseno** no es útil en estos casos si no se aplica algún tipo de preprocesamiento que permita tolerar errores.

En resumen, **la métrica más robusta y coherente para detectar errores ortográficos leves fue Jaro-Winkler**, y debería considerarse como primera opción en tareas de similitud textual sobre vocabulario específico o nombres propios.

Ejercicio 4 – Detección de Idioma

Objetivo: Detectar el idioma de los textos de los archivos cargados y registrar esta información.

Desarrollo:

Para este ejercicio se utilizó la biblioteca **langdetect**, que permite identificar el idioma predominante de un texto. Se aplicó el proceso a todos los archivos disponibles en la carpeta de "información".

Se cargó cada archivo como texto plano y se aplicó la función de detección de idioma.

1) Para tener una idea más confiable, se usó langdetect.detect_langs() en lugar de detect() a secas. Esto devolvió una lista con los idiomas más probables y su porcentaje de confianza, lo cual sirvió para verificar qué tan "segura" era la predicción del modelo.

2) Con esa información, se construyó un DataFrame que contiene:

    - El nombre del archivo.
    - El idioma detectado (más probable).
    - El porcentaje de confianza.

Resultados:

En todos los casos, la detección automática arrojó como resultado "en" (inglés) como idioma principal, con niveles de confianza muy altos (casi todos por encima de 0.99). Esto tiene sentido, ya que todos los textos provienen de sitios web, blogs o manuales relacionados con el juego Cascadia, y están escritos en inglés.

Además, se realizó una verificación manual rápida del contenido de cada archivo, y se confirmó que todos efectivamente están escritos en inglés. No se detectaron casos de ambigüedad ni fragmentos relevantes en otros idiomas que pudieran confundir al detector.

Conclusión:

La detección automática del idioma fue exitosa y consistente en todos los archivos. El uso de langdetect.detect_langs() permitió trabajar con valores de confianza y validar que el detector no solo acierta, sino que además lo hace con seguridad.

Ejercicio 5 – Análisis de Sentimientos y Búsqueda Semántica Filtrada

Objetivo: Clasificar las reseñas según sentimiento automáticamente y crear un sistema de búsqueda semántica que además permita filtrar por sentimiento.

Desarrollo:

1) En este ejercicio se decidió no usar las reviews que venían en el dataset original, ya que estaban mal extraídas: mezclaban HTML, encabezados del sitio, menús, y no era posible distinguir las reseñas reales. Dado que no se podía rescatar mucho texto útil de esos archivos, se optó por reextraer directamente las reseñas desde la página oficial de BoardGameGeek (BGG) para el juego Cascadia.
   Como ayuda, se utilizó el código que ya estaba realizado para el juego Pradera en la primer instancia del TP, con una única modificación: Se incorporó una lógica para dividir las reseñas por el separador `'-----'`, lo cual permitió separar fácilmente una reseña de otra. Esto también ayudó a poder **detectar el idioma por reseña individual**, en lugar de hacerlo por archivo completo.
   Enlace al código: Reviews Casacadia.ipynb
   Enlace al archivo resultante:
   https://drive.google.com/file/d/16-kRFNwswWWDbZl2DKdWYOpA9BSQSvU0/view?usp=drive_link

2) Como modelo de análisis de sentimiento se utilizó `nlptown/bert-base-multilingual-uncased-sentiment`, el cual devuelve una predicción en formato de estrellas del 1 al 5.
   Para poder convertir eso a una clasificación más útil, se definió la siguiente regla de interpretación:
   - **1 o 2 estrellas** → Sentimiento **negativo**
   - **3 estrellas** → Sentimiento **neutral**
   - **4 o 5 estrellas** → Sentimiento **positivo**

3) Dado que este modelo admite como máximo 512 tokens por entrada, pero algunas reseñas superaban ese límite, se diseñó una función para **dividir las reseñas en oraciones** (usando `nltk.sent_tokenize`), analizar cada oración por separado, y luego calcular:
   - La etiqueta más común (modo) entre las oraciones.
   - El **score promedio** de confianza.

De esta forma, se logra un análisis robusto incluso para reseñas largas.

4) El resultado de todo el proceso fue un **DataFrame final con las siguientes columnas**:

    - `titulo`: Título de la reseña (extraído del scraping).
    - `reseña`: Texto completo.
    - `sentimiento`: Positivo, neutro o negativo (interpretado).
    - `valor_crudo`: Etiqueta original del modelo (ej. "4 stars").
    - `score`: Confianza promedio del análisis (entre 0 y 1).

5) Además, se implementó una **función de búsqueda semántica** basada en embeddings generados con **S-BERT** (que ya se ha demostrado que funciona bien en oraciones), que permite realizar las siguientes acciones:

    - Ingresar una **frase de búsqueda** (ej: "great family game").
    - Aplicar un **filtro opcional por sentimiento**.
    - Recuperar las **tres reseñas más similares** (según similitud coseno), devolviendo título, sentimiento y score de similitud.

De esta forma, el sistema combina dos capas: **interpretación semántica del texto** y **filtrado emocional**, lo cual lo hace más flexible y útil para distintos tipos de análisis.

Conclusión:

El análisis de sentimientos aplicado a las reseñas extraídas mostró un resultado interesante: todas fueron clasificadas como positivas o neutras. Esto refleja una realidad del juego Cascadia: la mayoría de las personas tiene una opinión favorable, y no hay críticas duras o experiencias negativas registradas en las reseñas analizadas. Es un buen indicador de la recepción general del juego por parte de la comunidad.

En cuanto al sistema de búsqueda por similitud semántica, se observó que los valores de similitud no fueron muy altos (el máximo fue 0.37). Sin embargo, al revisar los resultados manualmente, se vio que las reseñas devueltas sí están relacionadas de manera coherente con la frase ingresada, incluso si el score es bajo. Esto indica que el modelo logra captar la relación semántica aunque los textos sean breves, variados o estén redactados de forma informal.

Ejercicio 6 – Clasificación Semántica de Preguntas

Objetivo: Crear un set de preguntas clasificadas en tres categorías: **Información**, **Relaciones** y **Estadísticas**, y entrenar modelos para predecir su categoría automáticamente.

Desarrollo:

1) Para este ejercicio se generó un dataset artificial de 3 preguntas, divididas en partes iguales entre tres categorías predefinidas:

    - **Información**: preguntas que se responden a partir de descripciones o textos informativos del juego (por ejemplo, el manual o presentaciones).
    - **Relaciones**: preguntas que involucran vínculos entre elementos, como asociaciones entre animales, hábitats o cartas.
    - **Estadísticas**: preguntas que se basan en valores, puntuaciones o estructuras numéricas.

Cada pregunta fue diseñada tomando como referencia los contenidos presentes en las diferentes fuentes de datos disponibles en el drive del juego Cascadia.

2) Se creó un DataFrame con las columnas:
    - **pregunta**: texto de la pregunta.
    - **clase**: categoría asignada (0: información, 1: relación, 2: estadística).

3) Se dividieron los datos en **conjunto de entrenamiento y prueba (80-20 respectivamente)** para poder evaluar el rendimiento de los modelos sobre datos no vistos.

4) Se definió una función para probar diferentes combinaciones de **vectorizadores** y **modelos de clasificación**:

    - **Vectorizadores utilizados**:
        - **TF-IDF**: representa cada pregunta como una bolsa de palabras, ponderando la importancia de cada término según su frecuencia. No captura contexto ni semántica, pero es

rápido, simple y sabe capturar patrones repetitivos o tareas estructuradas.
    - **FastText**: trabaja con **sub-palabras** y genera embeddings que pueden manejar bien errores ortográficos, variaciones de palabras o formas no vistas. Es un modelo robusto, especialmente útil en textos cortos.
    - **S-BERT**: diseñado específicamente para generar embeddings semánticos de oraciones completas. Se incluyó para evaluar si la riqueza semántica que ofrece podía aportar una mejora real frente a métodos más simples.

- Modelos:
    - **Random Forest:** Elegido por su robustez frente a ruido y su capacidad para modelar relaciones no lineales.
    - **Logistic Regression**: Se comporta muy bien en espacios linealmente separables y es fácil de interpretar.
    - **KNN**: Este modelo no entrena como tal, pero clasifica en base a **la cercanía en el espacio vectorial**. Se incluyó justamente para contrastar qué tan bien se organizaban las representaciones vectoriales

Cada combinación fue entrenada y evaluada utilizando métricas de **accuracy** y **F1 macro**, pero como nuestro dataset está balanceado no queremos darle más importancia a una clase u otra nos vamos a centrar en el valor del accuracy

Resultados*:*

	Vectorizer	Model	Accuracy	F1_macro
0	FastText	RandomForest	0.941176	0.939683
1	SBERT	LogisticRegression	0.901961	0.900354
2	SBERT	RandomForest	0.901961	0.900354
3	TF-IDF	LogisticRegression	0.882353	0.880846
4	TF-IDF	RandomForest	0.862745	0.861862
5	TF-IDF	KNN	0.862745	0.858406
6	FastText	LogisticRegression	0.862745	0.860091
7	FastText	KNN	0.862745	0.853782
8	SBERT	KNN	0.823529	0.810270

Como se puede ver, la mejor combinación fue FastText + Random Forest, alcanzando un accuracy del 94.1%, lo cual indica un desempeño muy alto.

Evaluación con preguntas nuevas:

Para validar la capacidad de generalización del modelo fuera del dataset original, se probaron seis preguntas nuevas formuladas manualmente y no presentes en el entrenamiento. De estas seis, cinco fueron clasificadas correctamente, lo que representa un 83.3% de acierto en esta muestra externa.

El único error se dio en una pregunta que comenzaba con “How many...”, la cual fue etiquetada como estadística por el modelo, cuando en realidad era una pregunta informativa sobre una regla del juego. Este caso sugiere que el modelo, aunque robusto, puede dejarse llevar por palabras clave si el contexto no es del todo claro.

Conclusión:

El clasificador entrenado logró distinguir con muy buen nivel de precisión entre los tres tipos de preguntas, tanto en el set de prueba como en ejemplos nuevos.
La combinación de FastText como vectorizador y Random Forest como clasificador se destacó no solo por sus métricas, sino también por su

capacidad para capturar patrones semánticos incluso en preguntas formuladas de distintas maneras.

Este tipo de sistema puede ser muy útil como componente previo en un asistente automático o en un buscador inteligente, ya que permite enrutar cada consulta hacia el recurso o documento más relevante, optimizando tiempos de respuesta y precisión en la búsqueda de información.