UNIVERSIDAD
NACIONAL
DE COLOMBIA

PROCESAMIENTO DEL LENGUAJE NATURAL

Práctica 1: Análisis de Sentimiento

Integrantes

MARIA CAMILA ZAPATA ARRUBLA
ANDRÉS ALEXIS GALVIS HERRERA
JUAN JOSÉ ZAPATA CADAVID
JUAN ESTEBAN MEJÍA ESPEJO

Docente

JAIME ALBERTO GUZMÁN LUNA

UNIVERSIDAD NACIONAL DE COLOMBIA
SEDE MEDELLÍN
FACULTAD DE MINAS

18 de Diciembre del 2024

Índice

Creación del Corpus

Fuente 1

Para este primer Corpus se decidió utilizar las reseñas de la página de Steam del videojuego **Life Is Strange: Double Exposure.** Se decidió utilizar este videojuego debido a la variedad de reseñas, tanto positivas como negativas, asegurando un balance en las clases.
Para la extracción de las reseñas se utilizó la biblioteca de ***Selenium*** de Python, debido a que el contenido carga dinámicamente a medida que se navega haciendo scroll-down en la página. Se tomaron las 200 primeras reseñas recuperadas y se guardaron en un archivo *.txt* El siguiente paso a realizar fue el etiquetado manual de las reseñas, donde cada miembro del equipo aportó en el proceso, encargándose de etiquetar 50 reseñas. Finalmente, se obtuvieron 112 reseñas negativas y 88 reseñas positivas, separadas en carpetas llamadas neg y pos, respectivamente.

Fuente 2

Se usó el corpus en español del Taller de Análisis Semántico TASS de la Sociedad Española para el Procesamiento del Lenguaje Natural, descargando los archivos *general-test-tagged-3l* y *general-train-tagged-3l*. Haciendo uso de la biblioteca ***XML*** de Python se leyeron los archivos y se accedió al contenido y a la polaridad, esta última se utilizó como filtro, asegurándose de solo tomar los tweets con polaridad positiva o negativa. Luego de filtrar los tweets por polaridad, se unieron los datos de train y de test para cada polaridad, para finalmente, de forma aleatoria, seleccionar una muestra del 20% del total de los datos, obteniendo así 3605 tweets de polaridad positiva y negativa, los cuales se llevaron a las carpetas de pos y neg, respectivamente, junto con las reseñas de la fuente 1.

Métricas del Corpus

Fuente 1

Reseñas Positivas

25%Q	50%Q	75%Q	Max	Min	Media	Desv. Estándar	Suma
0	4	26	351	0	20.72	37.87	19532

Reseñas Negativas

25%Q	50%Q	75%Q	Max	Min	Media	Desv. Estándar	Suma
0	7	38	391	0	27.25	42.44	30170

Fuente 2

Tweets Positivos

25%Q	50%Q	75%Q	Max	Min	Media	Desv. Estándar	Suma
12	17	20	31	0	16.21	5.76	58450

Tweets Negativos

25%Q	50%Q	75%Q	Max	Min	Media	Desv. Estándar	Suma
16	20	23	31	0	19.13	4.89	68980

Corpus Positivo

25%Q	50%Q	75%Q	Max	Min	Media	Desv. Estándar	Suma
9	16	21	351	0	17.15	18.09	78012

Corpus Negativo

25%Q	50%Q	75%Q	Max	Min	Media	Desv. Estándar	Suma
14	19	23	391	0	21.04	21.30	99150

Preprocesamiento

Una vez creado el Corpus, el siguiente paso fue aplicar varios preprocesamiento para limpiar y adecuar el texto, haciendo uso de la biblioteca de ***re*** de Python.

1. Limpiar Hashtags: Se identifican todos los hashtags buscando el símbolo #; se elimina el símbolo de almohadilla y se separan las palabras que componen el hashtag. Ejemplo: #MeGustaPython daría como resultado Me Gusta Python.
2. Conversión a minúsculas.
3. Eliminación de tildes.
4. Eliminación de números.
5. Eliminación de URLs: Se identifica si el texto comienza con http; si es así, se elimina.
6. Eliminación de retornos de carro.
7. Eliminación de HTML: Se eliminan todas las tags <>
8. Eliminación de signos de puntuación.
9. Eliminación de emoticones.
10. Se cambian las letras “q” por la palabra “que”.
11. Se cambian las menciones con @ a “usuario”.
12. Se eliminan espacios en blanco.
13. Se separan las reseñas de la fuente 1: Las reseñas se encuentran en un archivo *.txt*, separadas por un “Publicado el día de mes””; se separan por estas palabras clave con el texto “\n----------\n”, para facilitar su posterior uso.
14. Se guardan los textos procesados.

Extracción de Características

1. Tokenización: El texto se divide en unidades más pequeñas llamadas "tokens" (palabras o grupos de palabras). En el código, la función *tokenize_text* se encarga de convertir el texto en unigramas (palabras individuales) y bigramas (pares de palabras consecutivas). Este proceso es esencial para permitir que los algoritmos de aprendizaje trabajen con unidades significativas de texto.

2. Eliminación de Palabras Vacías (Stopwords): Las palabras vacías son términos comunes (como "el", "de", "la", etc.) que no aportan significado semántico relevante. La función *remove_stopwords* elimina estas palabras utilizando la lista de palabras vacías de NLTK. Esta eliminación reduce el ruido en los datos y mejora la calidad de las características extraídas.

3. Eliminación de Tokens de Baja Frecuencia:Las palabras o tokens que aparecen con poca frecuencia en el corpus pueden no aportar valor predictivo y aumentar la dimensionalidad del espacio de características. La función *remove_low_frequency_tokens* filtra los tokens que aparecen menos de un cierto umbral de frecuencia (por defecto, 3 veces). Este paso ayuda a reducir la dimensionalidad y mejora la eficiencia computacional.

4. Stemming (Reducción de Palabras a su Raíz): El stemming convierte las palabras a su forma base, eliminando sufijos y desinencias. La función *apply_stemming* aplica el Snowball Stemmer para reducir las palabras a sus raíces. Esto permite que palabras con diferentes formas gramaticales (como "correr", "corriendo" o "corrió") se consideren como la misma característica.

5. Unificación del Corpus: Los textos procesados se vuelven a unificar en un solo corpus. En el código, los textos procesados de tweets y reseñas se combinan para crear un corpus general que se guarda en un archivo *.csv* Este archivo puede utilizarse como entrada para modelos de clasificación, modelos de lenguaje, entre otros.

Ponderación de Características

A partir de las características extraídas en el paso anterior, se procede a ponderarlas (asignarles mayor o menor peso) usando las siguientes técnicas:

- **Frecuencia absoluta (Term Ocurrences o TO)**: Cada característica tiene un peso igual al número de veces que aparece en un documento dado del corpus. Se crea una función en Python llamada *calculate_to* para realizar este proceso.
- **Esquema TF-IDF (Term Frecuency-Inverse Document Frecuency)**: Esta ponderación otorga una mayor importancia a aquellas características que aparecen un mayor número de veces en el corpus, pero en pocos documentos del mismo.Se crea una función en Python llamada *calculate_tfidf* para realizar este proceso.

Luego de calcular la ponderación tanto con TO como con TF-IDF, se guardan en archivos *.npy* las matrices y nombres de las características.

Técnicas de Aprendizaje

Se evaluaron tres algoritmos: Regresión Logística, KNN y Árboles de Decisión; variando combinaciones de ponderación: TO o TF-IDF; y de uso o no uso de Stopwords, Stemming o Ambas. Las métricas a evaluar fueron: Precision, Recall y F1-Score.

Se definieron funciones para:

- Cargar los datos (el corpus procesado)
- Dividir los datos en entrenamiento y pruebas, con un 80% y 20% respectivamente.
- Entrenar el modelo de Regresión Logística, KNN y Árbol de Decisión.
- Guardar el modelo en un *.pkl*
- Realizar predicciones con el modelo y evaluar las métricas de Precision, Recall y F1-Score; esta evaluación se realiza usando Cross-Validation, con *K=10*
- Graficar los resultados y guardarlos en archivos *.png*

Se definieron las posibles combinaciones de Stopwords y Stemming:

Stopwords	Stemming
True	True
True	False
False	True
False	False

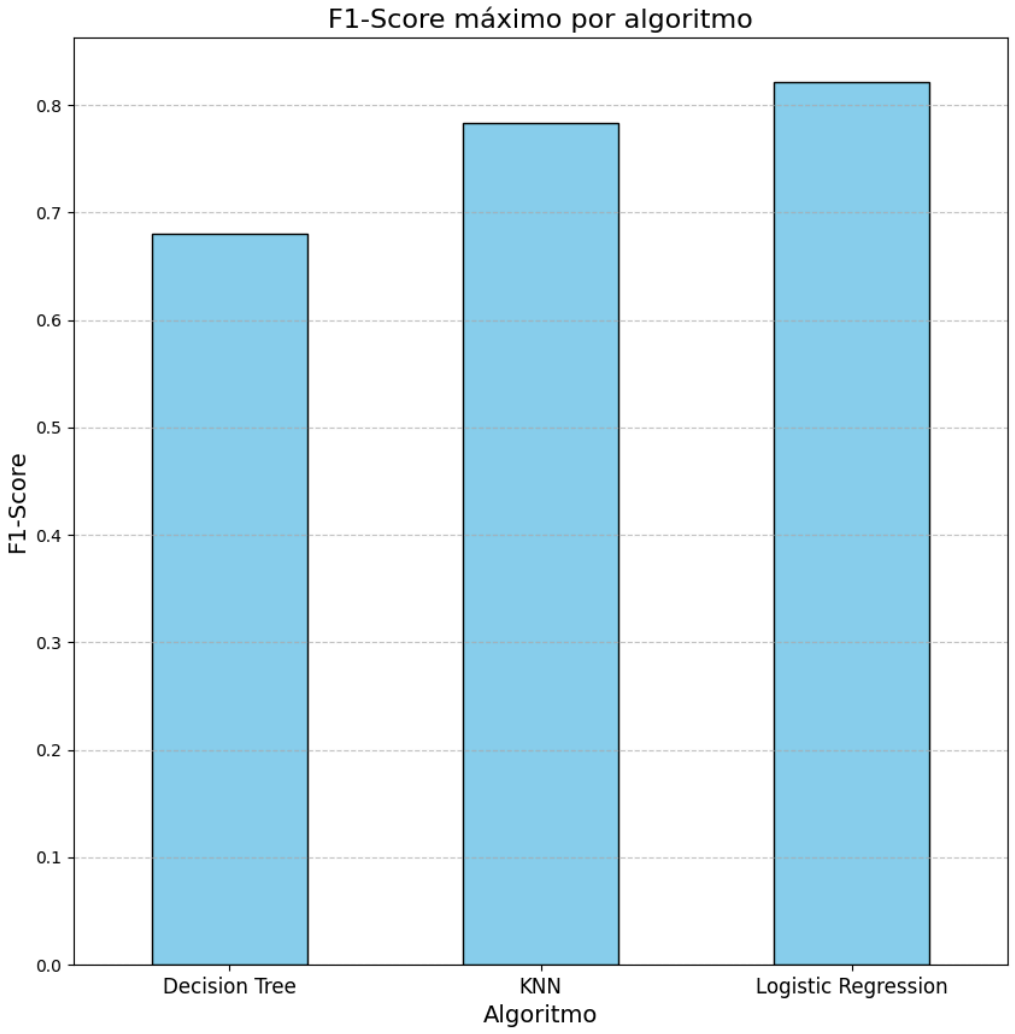
Se itera sobre cada pareja de posibles combinaciones y, usando las funciones listadas previamente, se entrenan los modelos de los 3 algoritmos, primero usando ponderación TO, y luego ponderación TF-IDF; se realizan predicciones y se calculan las métricas, las cuales se almacenan en una lista temporal para luego exportarse a un *.csv*. Se tienen entonces las siguientes combinaciones para cada uno de los tres algoritmos:

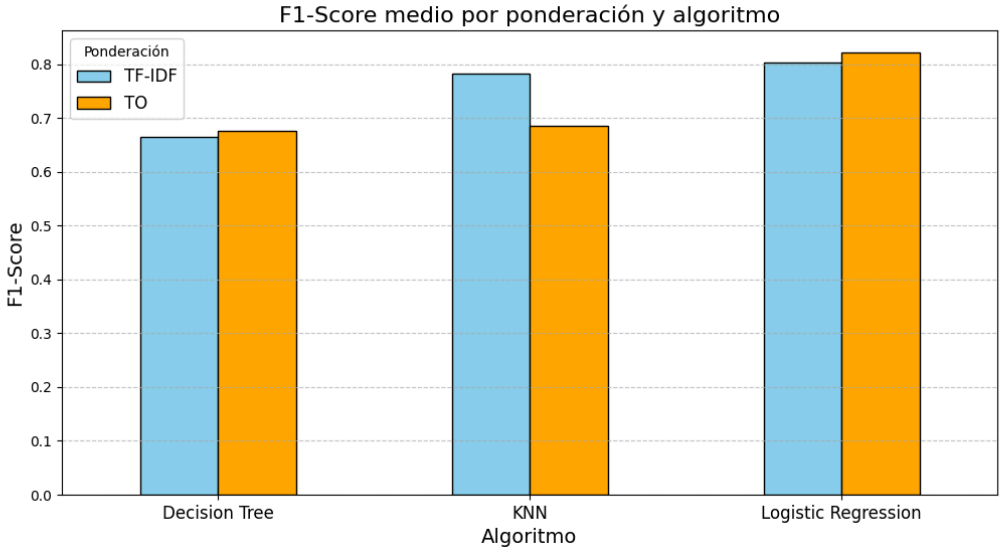
Ponderación	Stopwords	Stemming
TO	True	True
TO	True	False
TO	False	True
TO	False	False
TD-IDF	True	True
TD-IDF	True	False
TD-IDF	False	True
TD-IDF	False	False

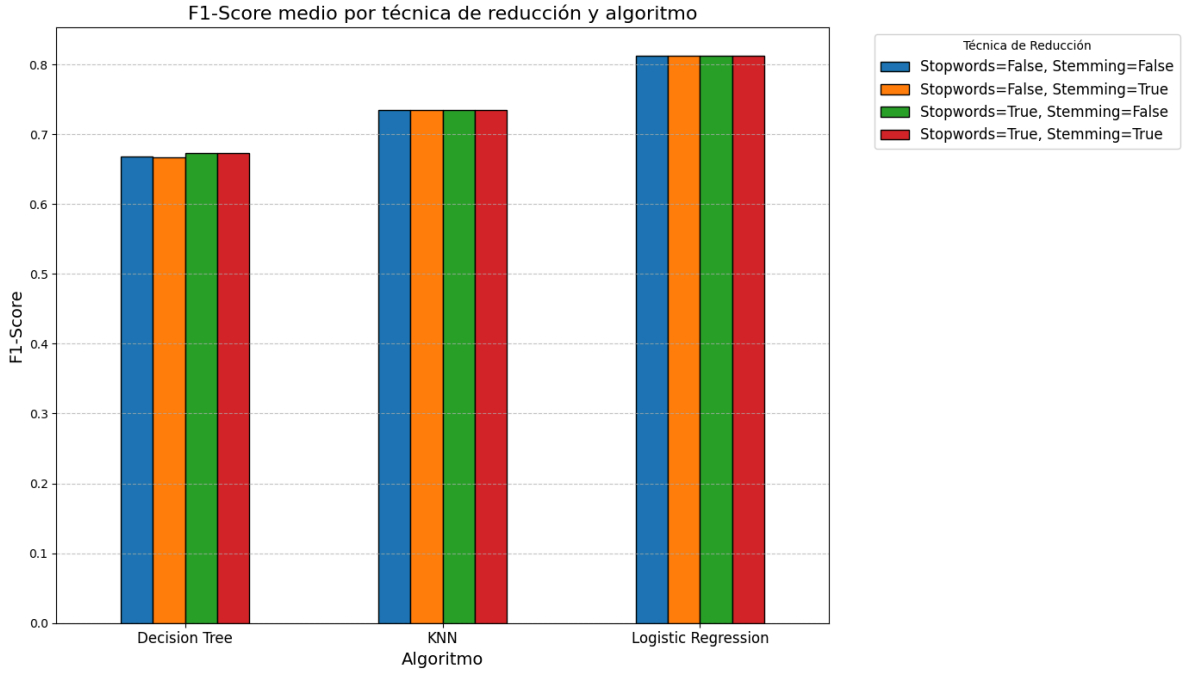
Finalmente, se obtuvieron los siguientes resultados:

				REGRESIÓN LOGÍSTICA			K VECINOS MÁS CERCANOS			ÁRBOLES DE DECISIÓN		
#	POND	STOPWORDS	STEMMING	PRECISION	RECALL	F1-SCORE	PRECISION	RECALL	F1-SCORE	PRECISION	RECALL	F1-SCORE
1	TO	True	True	0.8098	0.8352	0.8218	0.5888	0.8229	0.6857	0.6716	0.6826	0.6759
2	TO	True	False	0.8098	0.8352	0.8218	0.5888	0.8229	0.6857	0.6662	0.6826	0.6726
3	TO	False	True	0.8098	0.8352	0.8218	0.5888	0.8229	0.6857	0.6742	0.6881	0.6804
4	TO	False	False	0.8098	0.8352	0.8218	0.5888	0.8229	0.6857	0.6671	0.6868	0.6753
5	TF-IDF	True	True	0.8452	0.7666	0.8034	0.7649	0.805	0.7829	0.6562	0.6854	0.6691
6	TF-IDF	True	False	0.8452	0.7666	0.8034	0.7649	0.805	0.7829	0.662	0.6909	0.6744
7	TF-IDF	False	True	0.8452	0.7666	0.8034	0.7649	0.805	0.7829	0.639	0.6703	0.6531
8	TF-IDF	False	False	0.8452	0.7666	0.8034	0.7649	0.805	0.7829	0.654	0.6704	0.6613

A partir de dichos resultados se generaron las siguientes gráficas:




F1-Score medio por ponderación y algoritmo
Ponderación
TF-IDF
TO
F1-Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Decision Tree
KNN
Logistic Regression
Algoritmo


F1-Score medio por técnica de reducción y algoritmo
Técnica de Reducción
Stopwords=False, Stemming=False
Stopwords=False, Stemming=True
Stopwords=True, Stemming=False
Stopwords=True, Stemming=True
F1-Score
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Decision Tree
KNN
Logistic Regression
Algoritmo

Bonus: Uso de Modelos Preentrenados

Como ejercicio adicional se quiso evaluar el desempeño de un modelo preentrenado. Se tomó el modelo ***robertuito-sentiment-analysis***, alojado en HuggingFace. Se escogió este modelo ya que fue entrenado con más de 5000 tweets en español y está especializado en análisis de sentimiento.
Haciendo uso de la biblioteca ***Transformers*** de HuggingFace se consumió el modelo a través de su API; se usaron los mismos datos de prueba utilizados en los tres algoritmos anteriores para, hacer predicciones y calcular las métricas; además, se implementó manualmente Cross-Validation con *K=10*; también, se realizaron pruebas de las combinaciones de las diferentes técnicas de reducción, es decir, el uso de stopwords y stemming. Estos resultados se agregaron a una lista para luego ser exportados a un archivo *.csv*
Se obtuvieron los siguientes resultados:

Stopwords	Stemming	Precision	Recall	F1
True	True	0.58118318	0.66918717	0.62097154
True	False	0.55022326	0.5227911	0.53444886
False	True	0.62149098	0.59429237	0.60460353
False	False	0.59105861	0.45107578	0.510151

Conclusiones

A partir de las gráficas y resultados se evidencian los siguientes datos clave:

- El algoritmo con mejor desempeño general fue Regresión Logística, alcanzando un F1-Score máximo de 0.82; seguido de KNN, con un F1-Score máximo de 0.78
- El algoritmo con peor desempeño general fue el Árbol de Decisión, con un F1-Score máximo de 0.67
- Para la Regresión Logística, se obtuvieron mejores resultados usando la ponderación TO, aunque la diferencia no fue tan significativa.
- Para KNN, se obtuvieron mejores resultados usando la ponderación TF-IDF, con una diferencia significativa.
- Para Árboles de Decisión, se obtuvieron mejores resultados usando la ponderación TO, aunque la diferencia no fue significativa.
- Variar la técnica de reducción, para los algoritmos de Regresión Logística y KNN, no supuso ningún cambio en el resultado.
- Para Árboles de Decisión, se obtuvieron los mejores resultados con Stopwords y con Stemming, aunque la diferencia no fue significativa.
- Los elementos clave para un mejor desempeño fueron la elección correcta de algoritmo y método de ponderación.
- Para el modelo preentrenado, se obtuvieron los mejores resultados con Stemming.
- El uso de modelos preentrenados y más complejos no siempre es la mejor solución, ya que como muestran los resultados, se obtuvo un mejor desempeño con un algoritmo simple y más eficiente, como lo es la Regresión Logística.