

Nubimetrics Challenge

Análisis de Sentimiento en Comentarios

Introducción al Problema

El objetivo de este challenge fue crear un modelo que permita analizar reviews dejadas por clientes luego de que estos realizaran una compra en un Marketplace.

Estas reviews o comentarios contienen información muy valiosa sobre la reputación del producto y la marca en el mercado.

Por esa razón, el cliente en cuestión, “ELECTRIC COMPANY LLC”, solicitó un modelo que pueda identificar si el comentario expresa un sentimiento, negativo, neutro, o positivo. Para saber la reputación de su marca en el Marketplace.

NOTA: Este documento hace mención a como se enfrentó el problema y la manera de pensarlo, en el mismo se mencionan detalles claves a la hora de resolver la propuesta, pero, hay cosas que no se mencionan en esta documentación para que no se vuelva tan técnica.

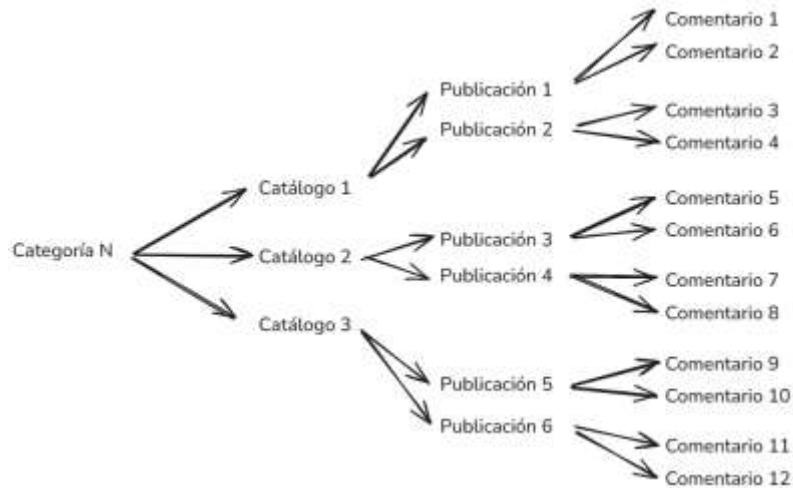
Para entrar en detalles técnicos, definiciones, validaciones o ver algo en mayor detalle se recomienda referirse a la notebook “**nb_comment_analyzer_training**”.

◆ Dataset de entrenamiento

El set de datos que se utilizó para entrenar y validar al modelo:

- Cuenta con **1435 registros** y **9 columnas**.
- Cada **comentario** corresponde a un registro.
- Las columnas son:
 - **id_categoria:** identificador único de la categoría.
 - **id_catalogo:** identificador único del catálogo.
 - **id_publicacion:** identificador único de la publicación.
 - **id_comentario:** identificador único del comentario.
 - **marca:** Marca del producto de la publicación.
 - **rate:** Puntaje de 1 a 5 otorgado por el comprador.
 - **valorization:** Apoyo de otros compradores al comentario.
 - **title:** Descripción breve de la conformidad del comprador.
 - **content:** Comentario realizado por el comprador.

- **Una publicación** puede tener **muchos comentarios**. Además, una publicación **no puede pertenecer** a **más de una categoría** ni a **más de un catálogo**.
- **Un catálogo** puede tener **muchas publicaciones**.
- **Una categoría** puede tener **muchos catálogos**.



📌 Análisis Exploratorio de Datos (EDA)

Durante el EDA, el enfoque fue entender cómo se distribuían los comentarios según las diferentes marcas y productos presentes en el Dataset.

El EDA tenía como objetivo descubrir patrones dentro de los datos, familiarizarme con los mismos para saber que esperar y cómo encarar la solución.

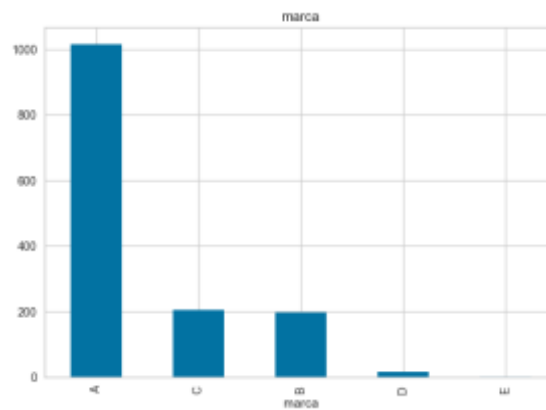
Esta etapa es clave para definir las estrategias de limpieza, preprocesamiento, feature engineering, modelado, etc.

◆ EDA Inicial.

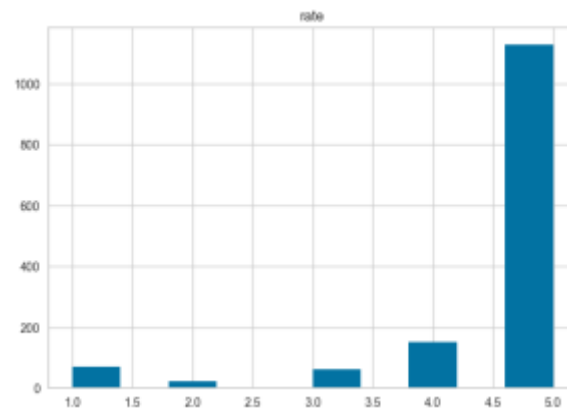
En un principio observamos la distribución de las distintas variables:

- Cantidad de categorías presentes = **32 categorías**.
- Cantidad de catálogos = **213 catálogos**.
- Cantidad de publicaciones = **352 publicaciones**.
- Cantidad de comentarios (coincide con la cantidad de registros) = **1435 comentarios**, de los cuales hay **16 nulos**.

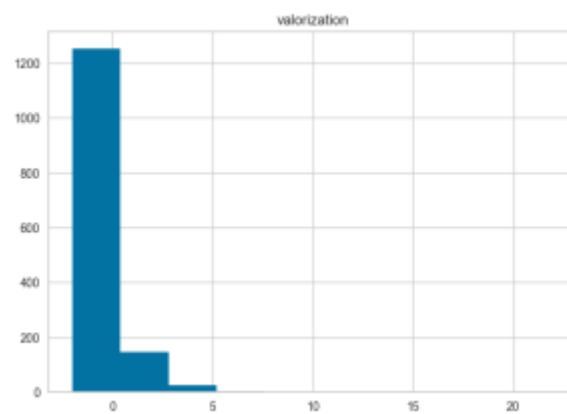
- Distribución de **comentarios por marca**:



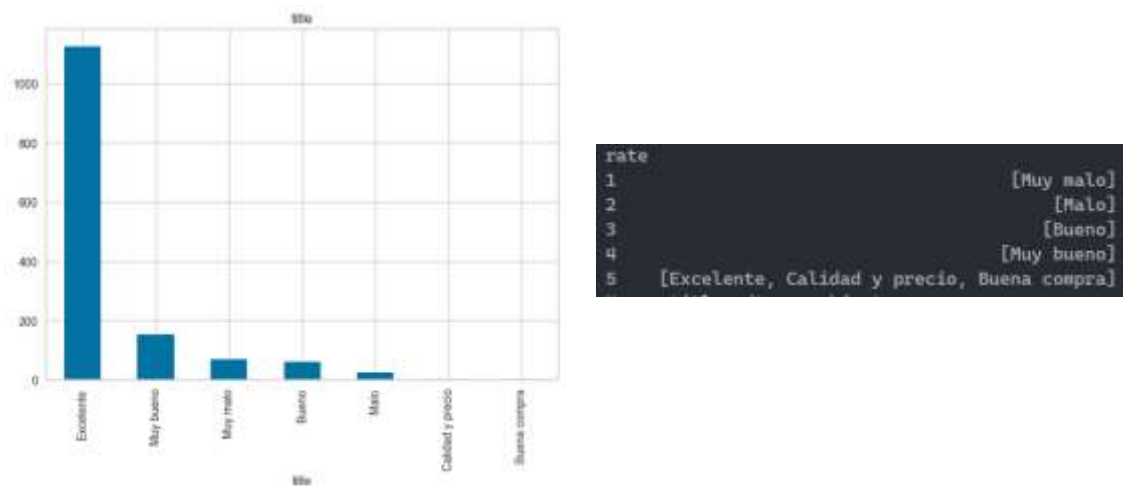
- Distribución de **rates**:



- Distribución de **valorización**:



- Distribución de **title** y como este se relaciona con los **rates**:



De aquí obtuvimos que:

- ✓ La mayoría de los productos tienen reviews positivas.
- ✓ Hay 16 comentarios que tienen el campo '**content**' nulo.
- ✓ La mayoría de las valorizaciones son nulas, y no influyen en el rate significativamente, a partir de esto se decidió **no utilizar** la variable '**valorization**'.
- ✓ La variable '**rate**' y '**title**' están altamente correlacionadas, se pueden ver que, mientras el rate es más alto mejor son las reviews de los clientes.
- ✓ La marca con **mayor** cantidad de comentarios es la **marca "A"**
- ✓ La marca con **menor** cantidad de comentarios es la **"E"**

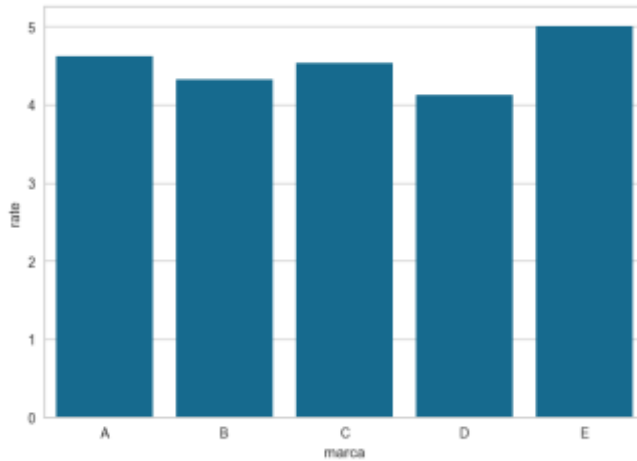
◆ Análisis Exploratorio.

Luego de explorar como las distintas variables se distribuían y comportaban de manera individual, continue estudiando como las distintas variables se comportaban entre sí, tratando de descubrir patrones dentro de los datos.

Para respaldar este análisis genere varios gráficos visuales:

✓ Rate promedio por marca.

Facilito ver cuales eral las marcas con reviews más positivas.

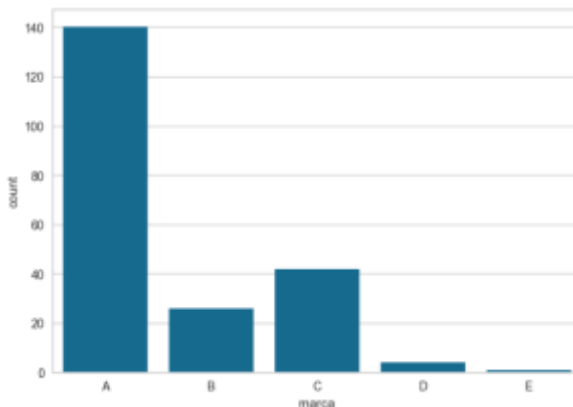


i El promedio de rates entre las distintas marcas es bastante similar.

i A pesar de lo mencionado, la **marca "E"** tiene el mayor promedio. Recordar que esta marca tiene **un único comentario**, el cual tiene un rate de 5, por esa razón esta marca es la de mayor rate.

✓ Cantidad de catálogos por marca.

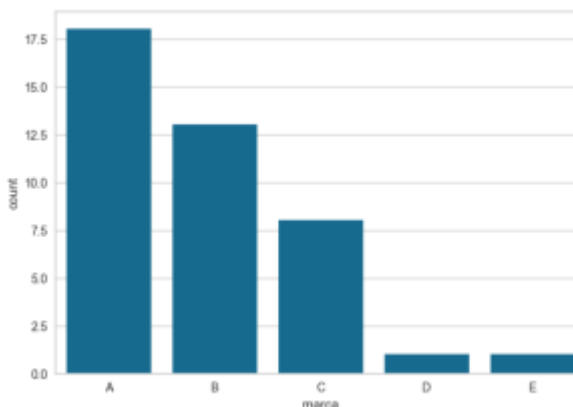
Permitió ver la presencia de las marcas en los distintos catálogos.



i La marca **"A"** es la que aparece en la mayoría de los catálogos.

✓ Categorías por marca.

Permitió ver la presencia de las marcas en las distintas categorías.



i La marca **"A"** es la que aparece en la mayoría de las categorías, tiene sentido ya que es la que tiene más publicaciones.

i Se observa que la marca **"B"** aparece en mayor cantidad de categorías que la marca **"C"**, pero en base al análisis anterior se observa que esta ultima aparece en mayor cantidad de catálogos.

Esto indica que **un catalogo puede contener varias publicaciones de una misma marca**. Por esa razón, la marca “C” tiene mayor cantidad de comentarios que la marca “B”, pero la “B”, esta en mas categorías, lo que indica que es una marca menos comentada que la “C”.

En análisis exploratorio detallado aporte una comprensión profunda del comportamiento y expectativas de los usuarios, me dio unas bases sólidas para definir estrategias efectivas para enfrentar y validar los distintos modelos.

◆ WordClouds

Las WordCloud (Nube de palabras) miden la frecuencia de palabras mas comunes en los textos.

Es un análisis textual que permite detectar cuales son los términos mas representativos en los comentarios, lo que nos facilita la comprensión sobre cómo están constituidos los comentarios.

Este análisis se realiza con regularidad a lo largo de la NoteBook, ya que nos da una idea de que palabras son mas comunes en las distintas clases de comentarios (Negativo, Neutro, o Positivo) como en el ejemplo siguiente, en donde en base al target (la obtención del mismo se explica en la siguiente sección de esta documentación), vemos cuales son las palabras más comunes:



Generación de target

Ya que el target no estaba especificado en el Dataset, tuve que definirlo.

Para esto me apoye en la variable “**rate**”, dependiendo la valoración que le dio el usuario al producto.

NOTA: También podría haberme apoyado de la variable “**valorization**”, por ejemplo: un comentario con **rate = 4** pero con valorizaciones negativas podría ser categorizado como neutro. Pero como la mayoría de los valores de esa variable son **0**, se opto por dejarla fuera del Dataset.

Entonces, basándonos en el “**rate**” clasificamos a los comentarios como:

- **Clase 0** – Comentario **Negativo** – rates = [1, 2].
- **Clase 1** – Comentario **Neutro** – rates = [3].
- **Clase 2** – Comentario **Positivo** – rates = [4, 5].

Preprocesamiento y Limpieza del Texto

Esta etapa es extremadamente importante, ya que para que el modelo pueda aprender a clasificar a los comentarios, estos deben estar normalizados y con el menor ruido posible.

Para limpiar el texto y normalizarlo se aplicaron los siguientes pasos:

1. Se unificaron los campos “**title**” y “**content**”, ya que hacer esto nos aporta una valoración mas especifica del usuario y nos enriquece las reviews.
2. Todas las palabras se llevan a minúsculas, para homogenizar palabras (Ej.: Producto -> producto).
3. Eliminación de tildes, números, signos de puntuación, y caracteres especiales.
4. Reducimos múltiples caracteres a uno solo, esto incluye espacios múltiples también (Ej.: Holaaa -> hola).
5. Reducción de palabras a su raíz etimológica. Esto lo hice aplicando Lematización y Stemming:
 - a. **Lematización:** Devuelve una versión reducida de la palabra, es una palabra de la misma familia (Ej.: comprando | comprado -> comprar).
 - b. **Stemming:** Reduce la palabra a su raíz etimológica mediante la sustracción de sufijos y prefijos de las palabras, la raíz que queda (stem) muchas veces no es una palabra en sí, por lo tanto, en algunas ocasiones las palabras pueden perder significado (Ej.: comprando -> compr).

NOTA: El modelo mostro mucha mejor performance al **utilizar Stemming**, por esa razón se aplicó la técnica mencionada.

6. Eliminación de **stopwords** son palabras sin significado semántico (palabras poco informativas sobre el contenido del texto), como pronombres, preposiciones, artículos, etc.

Ejemplo de limpieza de texto:

Texto Original	Texto Limpio
Excelente Sale aire fresco... no frío... es algo ruidoso... aún nose cuanto gasta de luz por que tengo unos días con el. Me lo pongo directo a mi cama y me funciona bastante bien. Pero no refresca la habitación.	8unción8e sal fresc no ruidos aun nos cuant gast 8unc pong direct cam 8unción bastant bien per no refresc

Al aplicar estos pasos nos aseguramos que el modelo se entrenará con textos limpios, estandarizados y homogéneos, lo que mejora la calidad de las predicciones.

Vectorización de Texto

Encoding, Tokenización o Vectorización de texto, es una representación de cada comentario en función de las palabras que lo componen. Se representa al texto limpio como un vector en el espacio de palabras que conforman el vocabulario, en otras palabras, se divide al texto en palabras individuales o **tokens**.

Entonces, para vectorizar al texto se cuentan el numero de veces que aparece cada palabra en un comentario y se le asigna un peso correspondiente a dicha palabra en una matriz (o Dataset), así generamos nuestros inputs para el modelo de clasificación.

Con todas las palabras detectadas generamos un conjunto de palabras únicas, esto se llama **vocabulario**, y representa todas las palabras que el modelo “aprenderá”.

Para el problema propuesto se generaron 3 diferentes DataSets para entrenar distintos modelos de clasificación, cada uno de estos DataSets se generó teniendo en cuenta una técnica distinta de vectorización.

CountVectorizer

Representaremos a los comentarios como un conteo de ocurrencias de cada palabra.

La problemática de este enfoque es que simplemente estamos indicando cuantas veces aparece una palabra y no estaremos resaltando la importancia de la palabra, por ejemplo, hay palabras que aparecen en muchos comentarios y resultan entonces poco informativas.

CountVectorizer + Term Frequency Inverse Document Frequency (TF – IDF)

Apoyándonos en la transformación anterior aplicaremos también la transformación Term Frequency Inverse Document Frequency.

Esta transformación **compara el número de veces que una palabra aparece** en un comentario contra el **número de comentarios en la que la palabra aparece**.

Este enfoque le da menos importancia a las palabras que aparecen en muchos documentos y resalta palabras más representativas de cada comentario.

TF – IDF y Singular Value Decomposition

Una vez obtenida la transformación **TF-IDF** se le aplicará una reducción de la dimensionalidad utilizando **SVD**.


SDV consiste en encontrar combinaciones de palabras que resulten informativas y quedarse con las “mejores”. Es una transformación algebraica similar a PCA (Principal Component Analysis) de manera de que podamos describir el DF con un número de combinaciones menor al número de términos que había originalmente. Las dimensiones que se generan son combinaciones lineales de los términos.

Esta reducción de la dimensionalidad podría mejorar la performance de un clasificador.





En la notebook se hace un análisis adicional, en el cual obtenemos la cantidad de componentes principales en función de la varianza explicada por cada uno. Puede pasar que se generen más componentes de las necesarias, de esa manera por ahí, a los 200 componentes ya capturamos el 100% de la varianza, pero si seleccionamos una cantidad mayor de componentes, nos estaríamos quedando prácticamente con ruido.

Entrenamiento, validación y mejora del modelo ganador.

◆ PyCaret

 Para encontrar nuestro modelo baseline utilizare la librería [PYCARET](#), una librería que “simplifica y automatiza los flujos de trabajo en Machine Learning”.

PyCaret abstrae todo el proceso de entrenamiento de modelos clásicos: preprocesamiento, comparación entre algoritmos, validación cruzada, métricas, ajuste de hiperparámetros, y generación de reportes en pocas líneas de código, lo que nos permite:

-  Probar múltiples modelos de clasificación automáticamente.
-  Comparar su rendimiento con métricas clave (Accuracy, F1, Recall, etc.).
-  Obtener un baseline fuerte como punto de partida.
-  Seleccionar el mejor modelo para luego optimizarlo manualmente si lo deseamos.

El uso de PyCaret nos permite acelerar la etapa de comparación de modelos, centrándonos en el análisis de resultados.

◆ Entrenamiento

Se entrenaron 3 modelos distintos sobre data sets generados con distintas técnicas de vectorización (ver sección anterior). El objetivo era evaluar cómo se comportaban los modelos frente a representaciones mas complejas de los textos, desde el conteo de ocurrencias hasta la reducción de dimensionalidad.

El foco estuvo principalmente en la capacidad del modelo de distinguir correctamente los comentarios negativos, por sobre los neutros y positivos, ya que la clase 0 es la mas importante desde el punto de vista de la reputación y acción comercial.

- ✓ La **técnica** que mostró mejores **resultados** fue **Count-Vectorizer**, a pesar de ser la mas simple fue la que mejor logro representar los comentarios.
- ✓ El **modelo** ganador fue **LightGBM**.

Una vez seleccionado el modelo y la técnica de procesamiento, se realizaron ajustes de hiperparámetros y validaciones cruzadas adicionales para mejorar la generalización del modelo ganador.

◆ Validación

Para validar el rendimiento y la capacidad de generalización del modelo se aplicaron diversas estrategias.

🔍 Métricas generales:

Se presta atención a las métricas clásicas para clasificación:

- **Precisión:** Que tan preciso es el clasificador para predecir instancias positivas.
- **Recall:** Proporción de positivos correctamente predichos.
- **F1-Score:** Medida armónica entre Recall y precisión.

⚠ La métrica que se intenta **maximizar** será **F1-Score**, ya que nos interesa un equilibrio entre Precisión y Recall, ya que no podemos aceptar falsos negativos, y debemos estar seguro de los verdaderos positivos.

🔍 Matriz de confusión:

Veremos visualmente como fueron las predicciones, que clases están siendo correctamente clasificadas y cuales están siendo confundidas.

Aquí veremos si hay alguna clase que cueste clasificar más que otra. Algo que se observo es que la clase 1 (comentarios neutros) fue la más difícil de predecir correctamente.

🔍 Curva ROC y AUC:

Evaluaremos la relación entre **Specificity** o False Positive Rate (que tan específico es el clasificador al predecir instancias negativas) y **Recall** o True Positive Rate. Veremos cómo se comporta el modelo al clasificar en distintos umbrales de probabilidad.

Lo que se busca es que la curva ROC este lo más pegada posible al vértice izquierdo superior, esto indica que el modelo tiene un buen nivel de generalización.

Por otra parte, el AUC (Area Under the Curve), resume en una sola métrica la relación que muestra el gráfico ROC, y sus valores se interpretan:

- $AUC = 1$ -> Modelo Perfecto.
- $AUC = 0.5$ -> Modelo Aleatorio.
- $AUC < 0.5$ -> Modelo Malo.

Curva Precisión-Recall:

Evaluaremos la relación entre Precisión y Recall para distintos umbrales de decisión, de manera similar que la curva ROC.

Esta curva es útil cuando existe un desbalance de clases, como en nuestro caso. Ya que se enfoca en la relación entre Falsos Positivos y Verdaderos Positivos.

Lo ideal es obtener una curva lo más cercanas posibles al punto (1,1). Esto indica que el modelo tiene una alta Precisión y un alto Recall.

Feature Importante y explicabilidad del modelo.

Trataremos de interpretar como el modelo hace las predicciones mediante el análisis de las variables más importantes.

Validaremos con dos enfoques:

- **Feature Importance tradicional**

Visualizaremos los pesos que le asigno el modelo a cada variable.

De Esta manera veremos cuales son las variables que más peso tienen a la hora de realizar las predicciones.

- **SHAP**

SHAP es una librería que permite darle explicabilidad al modelo, se logra una explicación más detallada y precisa de la importancia de las variables.

Usando SHAP podremos ver y entender como contribuye cada feature en la predicción de una clase específica.

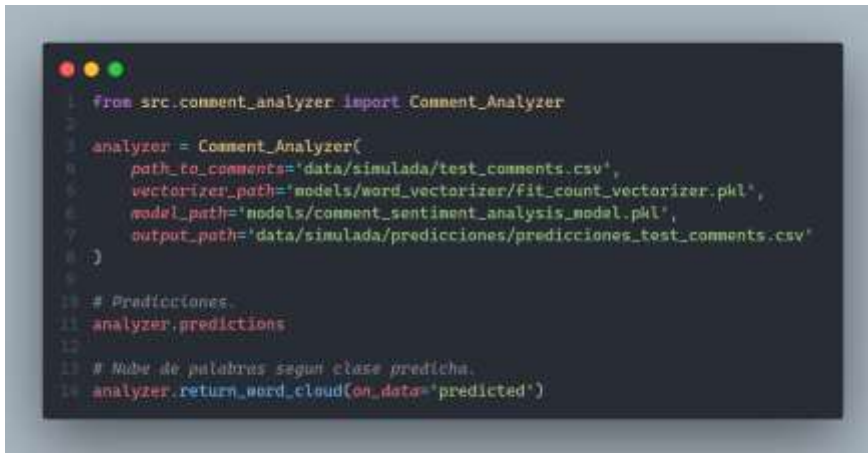
Esto es útil para justificar las elecciones del modelo, o identificar variables que tienen mayor impacto en la clasificación. Además, podremos detectar si alguna variable introduce sesgos o ruido.

Pipeline – Comment_Analyzer

Para encapsular toda la solución y facilitar la utilización del modelo entrenado para permitir la aplicación del mismo en entornos reales o productivos, se creó un pipeline.

Este pipeline o clase permite ejecutar todo el flujo completo de análisis de nuevos comentarios, lo que nos asegura, replicabilidad, organización y facilidad de uso para nuevos comentarios.

Ejemplo de uso:



```
1 from src.comment_analyzer import Comment_Analyzer
2
3 analyzer = Comment_Analyzer(
4     path_to_comments='data/simulada/test_comments.csv',
5     vectorizer_path='models/word_vectorizer/fit_count_vectorizer.pkl',
6     model_path='models/comment_sentiment_analysis_model.pkl',
7     output_path='data/simulada/predicciones/predicciones_test_comments.csv'
8 )
9
10 # Predicciones.
11 analyzer.predictions
12
13 # Nube de palabras segun clase predicha.
14 analyzer.return_word_cloud(on_data='predicted')
```

◆ Argumentos:

- *path_to_comments*: Ruta al archivo CSV que contiene los comentarios a analizar.
- *vectorizer_path*: Ruta al vectorizador entrenado en la notebook "nb_comment_analyzer_training".
- *model_path*: Ruta al modelo entrenado en la notebook "nb_comment_analyzer_training".
- *Output_path*: Ruta donde se guardarán las predicciones.

◆ Atributos de clase:

- *self.raw_data*: Datos crudos cargados del archivo CSV. Devuelve un `pd.DataFrame`.
- *self.clean_data*: Comentarios limpios y procesados. Devuelve un `pd.DataFrame`.
- *self.processed_data*: Matriz que contiene las ocurrencias de las palabras, en otras palabras, el texto vectorizado, listo para predecir. Devuelve un `pd.DataFrame`.
- *self.predictions*: Predicciones del modelo para cada comentario en el CSV. Devuelve un `Array`.

◆ Métodos externos:

- *return_word_cloud(on_data='predicted')*: Dependiendo del parametro ingresado genera una WordCloud para:
 - *'raw'*: Texto original.
 - *'clean'*: Texto limpio.
 - *'predicted'*: Texto segmentado por clase predicha.

◆ **Métodos privados:**

- `__load_data()`: Carga el archivo CSV.
- `__validate_data(raw_data)`: Valida y limpia nulos, reemplaza strings vacíos, y asegura integridad mínima para el análisis.
- `__load_stop_words()`: Carga las stopwords en español utilizando nltk.
- `__clean_data()`: Aplica preprocesamiento completo sobre los comentarios:
- `__load_vectorizer(vectorizer_path)`: Carga el vectorizador entrenado (CountVectorizer).
- `__load_model(model_path)`: Carga el modelo entrenado (LightGBM).
- `__preprocess_data()`: Transforma el campo text_label (campo interno de la clase) a una representación numérica mediante el vectorizador.
- `__predict_comments()`: Aplica el modelo cargado a los datos procesados para obtener las predicciones (Clase 0, 1 o 2).
- `__save_predictions(output_path)`: Guarda las predicciones en un nuevo archivo .csv con una columna adicional 'predictions'.