M71V MAESTRÍA EN GESTIÓN Y ANÁLISIS DE DATOS FINANCIEROS

M71V. 09 – MODELOS BASADOS EN DATOS NO ESTRUCTURADOS

TRABAJO FINAL

PROCESAMIENTO DE DATOS DE TEXTO

DOCENTE:

- **GARCÍA FRONTI, JAVIER IGNACIO**

ALUMNO:

- **VIZZO, MARIANO**

PRIMER TRIMESTRE 2024

ÍNDICE

1. INTRODUCCIÓN

1.1. Título del trabajo práctico

"Uso de K-Means, BERT y modelos predictivos para clasificar documentos relevantes para obtener un marco teórico en un Trabajo Final de Maestría que refiere a la predicción de decisiones de inversión de los usuarios de una wallet (perteneciente a una Fintech Argentina)"

1.2. Definición del problema

Muchos investigadores enfrentan el desafío de que sus conjuntos de datos contienen artículos que no son relevantes para su pregunta de investigación. Por ejemplo, si el objetivo es encontrar documentos relacionados con la predicción de decisiones de inversión de los usuarios de una wallet de Fintech, los investigadores deben lidiar con términos de búsqueda ambiguos.

Las palabras "*fintech*", "*inversión*", "*educación financiera*", "*inclusión financiera*" o "*aprendizaje automático*" conducen a muchos artículos relevantes para la investigación, pero también a artículos que no lo son (por ejemplo, artículos sobre educación financiera de estudiantes universitarios, aprendizaje de los estudiantes, etc.). Al utilizar técnicas avanzadas de procesamiento de lenguaje natural (NLP) y métodos de clasificación, este enfoque permite excluir estos artículos sin tener que combinar términos de búsqueda específicos. Además, el mismo código también se puede utilizar para eliminar o preferir ciertos géneros, como publicidad, noticias deportivas, etc.

Para abordar este problema, se utilizarán dos etapas principales:

- **Agrupación de documentos utilizando K-Means.**
- **Clasificación de relevancia utilizando BERT en conjunto de datos de texto con diferentes modelos predictivos de clasificación.**

Adicionalmente, se aplicará un **modelamiento de tópicos utilizando LDA.**

1.3. Objetivos

El objetivo general del trabajo práctico es aplicar técnicas avanzadas de procesamiento de lenguaje natural (NLP) y métodos de clasificación para identificar y clasificar documentos académicos y de investigación que sean relevantes para obtener un marco teórico en un Trabajo Final de Maestría que refiere a la predicción de decisiones de inversión de los usuarios de una wallet de una Fintech.
En aquellos casos que el documento académico o de investigación seleccionado se relacione o contribuya a mejorar el marco teórico del Trabajo Final de Maestría, será clasificado como "*Relevante*", en caso contrario, será clasificado como "*Irrelevante*".

2. DESCRIPCIÓN DEL CORPUS DE DOCUMENTOS

2.1. Recopilación de datos

Para analizar las tendencias tecnológicas en Fintech, se realizó una búsqueda de informes y documentos en formato PDF en varias fuentes confiables, como Google Scholar, ResearchGate, ScienceDirect, Deloitte Insights, PwC Publications, KPMG Insights, Banco Mundial y Fondo Monetario Internacional.

La metodología utilizada consiste en identificar palabras clave como "*inclusión financiera*", "*fintech y big data*", "*aprendizaje automático*", "*machine learning*", "*educación financiera*", etc. Luego, se realizan búsquedas en las plataformas mencionadas y se descargan los documentos pertinentes.
Para finalizar con la etapa de recopilación de los datos, se realiza una revisión de los documentos para asegurar su relevancia y calidad, sin embargo, se consideraron documentos no relevantes para el análisis.

2.1.1. Importación del corpus de documentos

Se realiza una extracción y organización del contenido de archivos PDF distribuidos en varios corpus almacenados en Google Drive. Inicialmente, se define una función llamada **`read_pdf`** que utiliza **`extract_text`** para leer el contenido de los archivos PDF especificados por su ruta, automatizando así la tarea de extracción de texto.
Luego, se define una lista de directorios (**`directorios_corpus)`** que contiene las rutas a los directorios donde se almacenan los archivos PDF de distintos corpus. Se crea un diccionario (**`pdf_texts`**) para almacenar el contenido extraído, estructurando los datos de tal manera que cada clave corresponde a un corpus y su valor es otro diccionario con los textos de los archivos PDF.
Para facilitar el análisis, los textos extraídos de cada corpus se convierten en DataFrames. Estos DataFrames se almacenan en un diccionario (**`dataframes_corpus`**), permitiendo un acceso organizado y estructurado a los datos. Posteriormente, se accede a los DataFrames de cada corpus por separado y se imprime una muestra de los primeros registros para verificar la correcta lectura y almacenamiento de los datos.
Finalmente, se concatenan todos los DataFrames individuales en un único DataFrame (**`df_original`**), que contiene los datos de todos los corpus combinados. Este DataFrame unificado facilita el análisis global de los documentos extraídos, proporcionando una estructura clara y manejable para futuros análisis en proyectos académicos y de investigación.

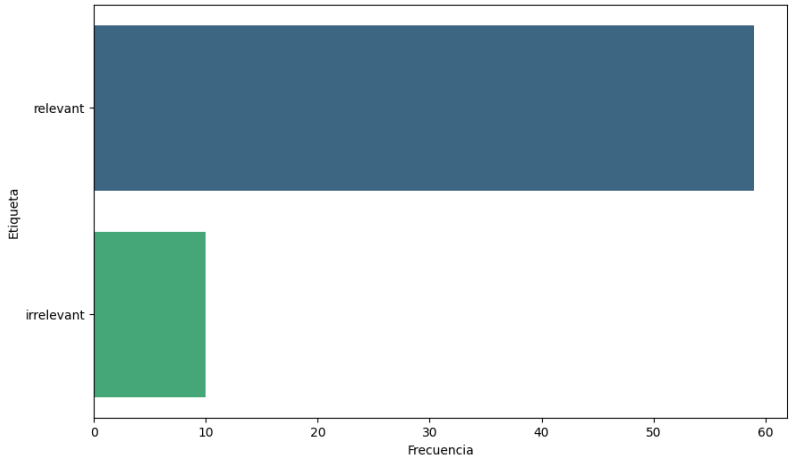
2.1.2. Interfaz de usuario interactiva

Se crea una interfaz simple utilizando **`ipywidgets`** para etiquetar documentos PDF como relevantes o irrelevantes. Primero, se añade una columna **`label`** al DataFrame unificado **`df`** para almacenar las etiquetas.
La función **`show_document(index)`** muestra el contenido del documento actual y los botones de etiquetado, permitiendo al usuario ver los primeros 2000 caracteres del texto del documento. La función **`label_document(label)`** asigna una etiqueta al documento actual y avanza al siguiente documento.
Se crean dos botones (**`relevant_button`** y **`irrelevant_button`**) para que el usuario pueda etiquetar los documentos. Estos botones se conectan a la función de etiquetado mediante **`on_click`**.
El primer documento se muestra junto con los botones de etiquetado al llamar a **`show_document(index)`**, iniciando el proceso de etiquetado interactivo. Finalmente, se visualiza el DataFrame con las etiquetas asignadas, proporcionando una visión completa de todos los documentos junto con su clasificación de relevancia.

2.2. Análisis exploratorio

2.2.1. Frecuencias de documentos

Gráfico Nº1: Frecuencia de documentos relevantes e irrelevantes

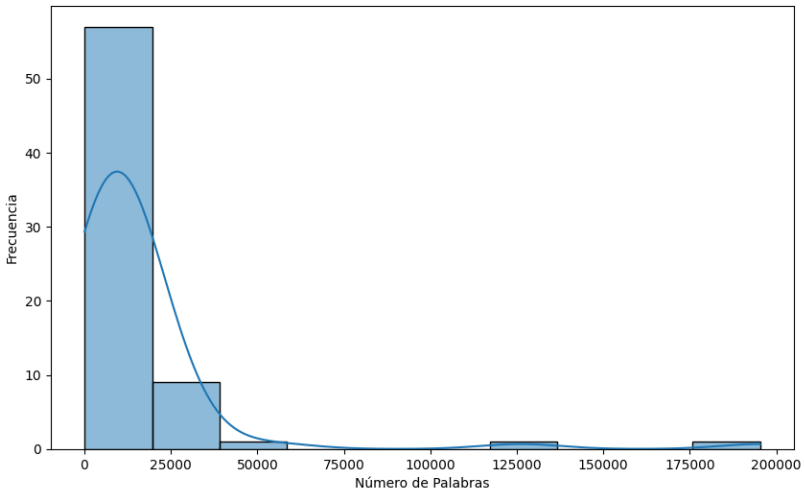


Fuente: elaboración propia en Python.

El gráfico Nº1 indica que el conjunto de datos se encuentra significativamente desbalanceado, sin embargo, se procede a continuar con procesamiento de datos y su posterior análisis.
Luego, en el apartado "*5.4. SMOTE: Técnica de balanceo de clases*", se corrige el desbalanceo y se comparan los resultados de los diferentes modelos predictivos, verificando que existen mejoras en las predicciones al aplicar la técnica mencionada.

2.2.2. Histograma de longitudes de textos

Gráfico Nº2: Histograma de longitudes de textos



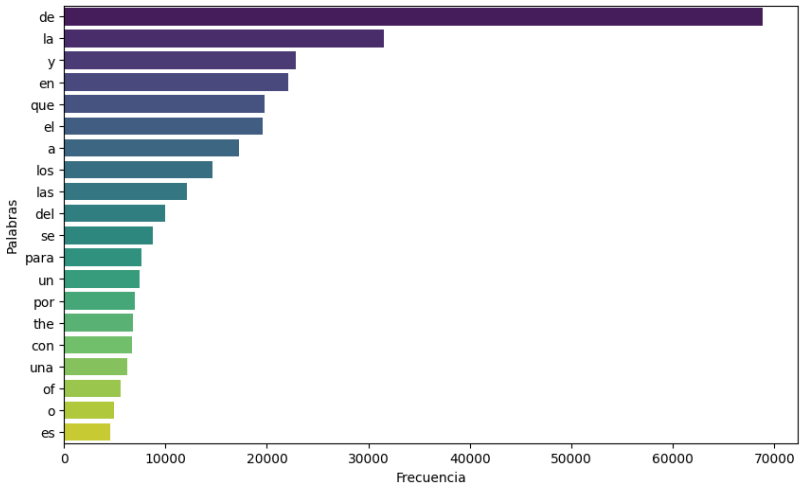
Fuente: elaboración propia en Python.

La interpretación del gráfico sugiere que la mayor parte de los textos del corpus tienen un número relativamente bajo de palabras. Esto se refleja en la alta frecuencia de textos con una longitud de hasta 25.000 palabras. Por otro lado, hay pocos textos que presentan una longitud extremadamente alta, llegando hasta 200.000 palabras en el eje X.

Una razón para esta distribución puede ser la variedad en el tipo de documentos presentes en el corpus. Los documentos varían ampliamente en longitud debido a la naturaleza de su contenido. Por ejemplo, algunos artículos son inherentemente cortos, como los artículos de opinión o resúmenes, mientras que otros pueden ser mucho más extensos, como reportes detallados, tesis, o incluso libros completos.

2.2.3. Frecuencias de palabras (sin eliminar stopwords)

Gráfico Nº3: Frecuencias de palabras de texto no procesado



Fuente: elaboración propia en Python.

El gráfico muestra la frecuencia de las palabras más comunes del corpus de documentos sin eliminar las stopwords (palabras comunes que no aportan mucho significado, como "*de*", "*la*", "*y*", "*en*", "*que*", etc.). La mayoría de las palabras más frecuentes son stopwords, lo cual es esperable en un análisis de texto no procesado.

3. CLUSTERING DE TEXTOS CON K-MEANS

3.1. Preprocesamiento de textos

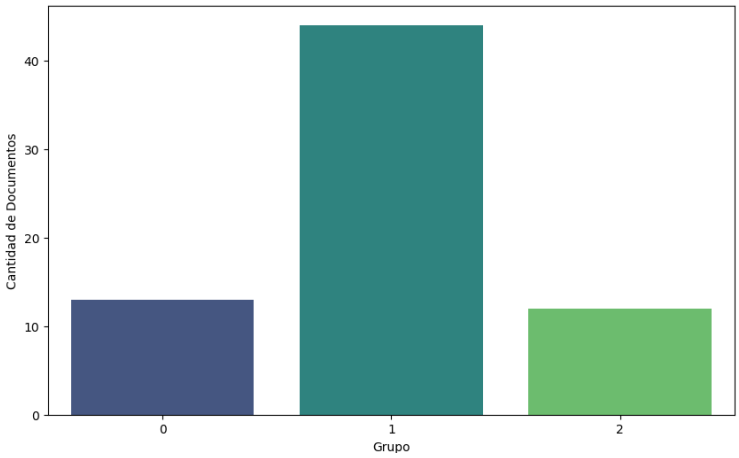
Primero, se define una lista predefinida de stopwords en español, que son palabras comunes que se eliminan del texto para centrarse en las palabras más significativas. Estas stopwords incluyen palabras como "*de*", "*la*", "*que*", y "*el*" que se mencionaron en el apartado anterior.
A continuación, se define una función **`preprocess_text`** que realiza varios pasos de limpieza de texto. Esta función convierte el texto a minúsculas para asegurar la uniformidad, elimina los signos de puntuación y separa el texto en tokens o palabras individuales. Luego, filtra estas palabras eliminando las stopwords predefinidas. Finalmente, las palabras restantes se unen nuevamente en una cadena de texto limpia.
El DataFrame **`df`** se actualiza añadiendo una nueva columna **`text_clean`**, donde se almacena el texto procesado. Este preprocesamiento facilita análisis posteriores al eliminar elementos no relevantes del texto original.
Además, se añade una columna **`length`** al DataFrame para almacenar la longitud del texto limpio en términos de número de palabras. Esto permite realizar análisis sobre la extensión de los documentos.

3.2. Vectorización de textos

Para representar el texto de manera numérica, se utiliza la técnica de vectorización TF-IDF (Term Frequency-Inverse Document Frequency). Esta técnica convierte el texto limpio en vectores numéricos, donde cada dimensión representa una palabra y su valor representa su importancia relativa en el documento. Se limita el número de características a 100.
Finalmente, se aplica el algoritmo de clustering KMeans para agrupar los textos en tres clusters distintos. Cada documento se asigna a uno de estos **`clusters`**, y el resultado se almacena en una nueva columna cluster en el DataFrame. Este agrupamiento permite identificar patrones y similitudes entre los documentos.

3.3. Cantidad de documentos por grupo

Gráfico Nº4: Cantidad de documentos por grupo

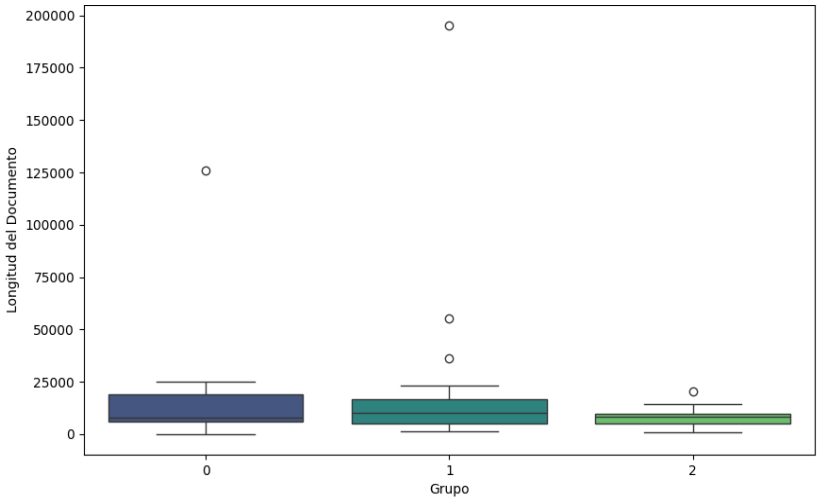


Fuente: elaboración propia en Python.

El gráfico muestra la distribución de la cantidad de documentos en cada uno de los tres grupos creados mediante el algoritmo de clustering KMeans. Observamos que el Grupo 1 contiene la mayor cantidad de documentos (44), mientras que los Grupos 0 (13) y 2 (12) tienen una cantidad significativamente menor. Esto sugiere que los documentos en el Grupo 1 comparten características más comunes que los otros grupos, resultando en una mayor agrupación.

3.4. Distribución de longitudes de documentos por grupo

Gráfico Nº5: Distribución de longitudes de documentos por grupo

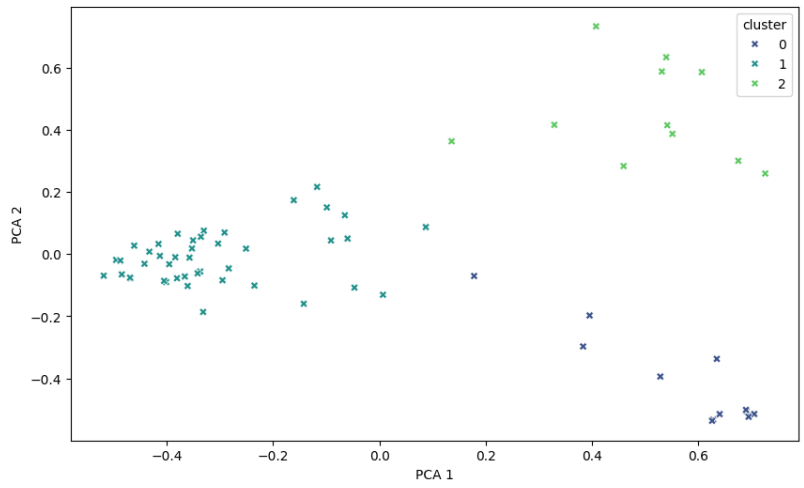


Fuente: elaboración propia en Python.

El gráfico Nº5 es un boxplot que ilustra la distribución de las longitudes de los documentos en palabras para cada grupo. La mayoría de los documentos en todos los grupos tienen longitudes relativamente bajas, alrededor de 25.000 palabras o menos. Sin embargo, hay algunos documentos atípicos en cada grupo que tienen longitudes extremadamente altas, llegando hasta 200.000 palabras. Estos outliers pueden representar reportes extensos, tesis o libros completos. Este gráfico también muestra que la mediana de las longitudes es similar entre los grupos, aunque el Grupo 2 parece tener una distribución de longitudes ligeramente más concentrada.

3.5. Visualización de clusters de documentos

Gráfico Nº6: Clusters de documentos



Fuente: elaboración propia en Python.

El gráfico Nº6 es una visualización mediante PCA que muestra una separación clara entre los tres clusters, lo que indica que el algoritmo de clustering ha identificado grupos de documentos con características distintivas.

Algunos clusters están más dispersos que otros, indicando que la variabilidad interna de los documentos en esos clusters es mayor. Por un lado, el grupo 1 (verde) está más concentrado, lo que sugiere que los documentos en este cluster son más similares entre sí, por otro lado, el grupo 0 (azul) y el grupo 2 (verde claro) están más dispersos, lo que sugiere mayor diversidad en el contenido de los documentos de estos clusters.
La alta variabilidad en la longitud de los documentos en ciertos clusters (especialmente el grupo 1) puede indicar la presencia de documentos de naturaleza diversa dentro de ese cluster. Ante esta situación, en el apartado *"6. TOPIC MODELING CON LDA"*, se realizará una subdivisión de estos clusters aplicando técnicas adicionales de refinamiento, en las cuales se verifica mejoras en la cohesión interna.

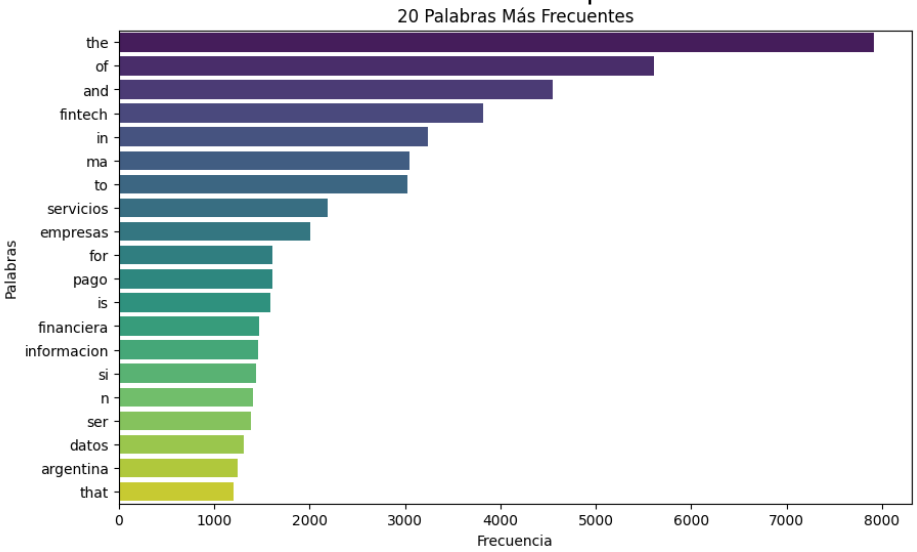
4. IMPLEMENTACIÓN DE BERT

4.1. Preprocesamiento de textos

Ahora, se utiliza BERT (Bidirectional Encoder Representations from Transformers) para obtener representaciones de alta calidad de los documentos, capturando el contexto y el significado de las palabras en el texto.
Inicialmente, se carga el DataFrame etiquetado desde un archivo CSV almacenado en Google Drive, lo que permite trabajar con una colección de documentos previamente categorizados. Se define una lista de stopwords en español utilizando la biblioteca **`nltk`**, lo que facilita la eliminación de palabras comunes que no aportan valor significativo al análisis del contenido.
El siguiente paso consiste en la inicialización de un lematizador de **`nltk`**, que reduce las palabras a su forma base o raíz, permitiendo un análisis más coherente y uniforme del texto. Para la limpieza del texto, se utiliza la función **`clean_text`**, que elimina caracteres especiales y dígitos, dejando solo letras y espacios en blanco. Esto asegura que el texto esté libre de elementos no alfabéticos que podrían interferir con el análisis.
La normalización del texto se lleva a cabo mediante la función **`normalize_text`**, que convierte el texto a minúsculas y elimina tildes, garantizando la uniformidad y la consistencia del formato del texto. Posteriormente, se aplica la función **`remove_stopwords_and_tokenize`**, que tokeniza el texto en palabras individuales y elimina las stopwords, reduciendo el ruido en los datos y enfocándose en las palabras más relevantes.
La lematización del texto se realiza con la función **`lemmatize_text`**, que convierte las palabras tokenizadas a su forma base, unificando variaciones de la misma palabra. Estas funciones de preprocesamiento se aplican secuencialmente a la columna de texto original del DataFrame, resultando en una nueva columna que contiene el texto limpio, normalizado, tokenizado y lematizado.
Finalmente, el DataFrame preprocesado se guarda en un nuevo archivo CSV, listo para ser utilizado en análisis posteriores, incluyendo la aplicación de modelos avanzados de procesamiento de lenguaje natural como BERT. Este exhaustivo preprocesamiento del texto es crucial para mejorar la precisión y eficiencia del análisis, permitiendo obtener mejores resultados en la tarea de modelado y clasificación de documentos.

4.2. Frecuencia de palabras

Gráfico N°7: Frecuencia de palabras



Fuente: elaboración propia en Python.

El gráfico revela la presencia de stopwords comunes en inglés, como "*the*", "*of*", "*and*", "*in*", "*to*", "*is*", "*for*", "*that*". Esto indica que algunas de estas palabras no se eliminaron completamente durante el preprocesamiento del texto. La omisión de estas stopwords podría afectar la calidad del análisis, ya que introducen ruido y no aportan valor significativo al entendimiento del contenido de los documentos.

Por otro lado, el gráfico también destaca términos relevantes específicos del dominio de Fintech. Palabras como "*fintech*", "*servicios*", "*empresas*", "*pago*", "*financiera*", "*informacion*", "*datos*" y "*argentina*" son frecuentes en los documentos. Estos términos son cruciales para el análisis del sector Fintech, ya que están directamente relacionados con los servicios y operaciones de las empresas Fintech, así como con los datos financieros y el contexto geográfico de Argentina.

4.3. Nube de palabras

Para un mayor entendimiento del conjunto de datos, en primer lugar, se realiza una nube de palabras con las 200 palabras más importantes.

Gráfico N°8: Nube de palabras

Fuente: elaboración propia en Python.

La imagen muestra las palabras más frecuentes en el texto lematizado de tus documentos. Las palabras más grandes y centradas son las que aparecen con mayor frecuencia en el texto. En esta nube de palabras, podemos observar que términos como "*si*", "*solo*", "*vez*", "*cada*", *"forma", "tambien" "manera"* y "*ello*" son muy comunes. Sin embargo, estas palabras no aportan mucho valor en términos de análisis de contenido porque son palabras vacías o stopwords.
Además, se visualizan palabras con errores o innecesarias para el análisis, como, por ejemplo; *"trave", "ma", "pai", "caso", "b" "ii"*, etc.
Luego, se realizan mejoras con la incorporación de stopwords personalizadas, lematización y stemming y ajustes en la visualización, obteniendo la imagen mejorada que se muestra en el apartado siguiente.

4.4. Nube de palabras mejorada

Gráfico Nº9: Nube de palabras mejorada

Fuente: elaboración propia en Python.

La nube de palabras mejorada destaca varias palabras dominantes como "*fintech*", "*empresa*", "*cliente*", "*proceso*", "*sistema*", "*informacion*" y "*usuario*". Estas palabras reflejan los temas centrales del corpus de documentos relacionados con el sector Fintech, señalando las áreas de mayor interés y frecuencia en los textos analizados.
En cuanto a los temas relevantes, términos como "*empresa*" y "*cliente*" sugieren un enfoque significativo en la interacción entre las empresas Fintech y sus clientes. Esto indica que muchos documentos se centran en cómo estas empresas gestionan y atienden a su base de clientes.
Además, palabras como "*proceso*", "*sistema*", "*plataforma*" y "*servicio*" indican que los documentos discuten extensamente sobre los procesos operativos y los sistemas tecnológicos utilizados en la industria Fintech. Esto resalta la importancia de la infraestructura tecnológica y los servicios que soportan las operaciones diarias de las empresas Fintech.
La tecnología y el desarrollo continuo son temas prominentes en el corpus, como se observa en las palabras "*tecnologia*", "*desarrollo*" e "*innovacion*". Estos términos subrayan el papel crucial de la innovación tecnológica en el avance y la competitividad del sector Fintech.
El aspecto financiero y el funcionamiento del mercado también son temas centrales, reflejados en palabras como "*financiera*", "*mercado*", "*credito*" e "*inversion*". Estos términos destacan las discusiones sobre el entorno financiero y las actividades de inversión que caracterizan a la industria Fintech.
La inclusión financiera y la regulación son otros temas importantes en el corpus. La frase "*inclusion financiera*" sugiere que la expansión del acceso a servicios financieros es un tema significativo. Asimismo, términos como

"*regulacion*", "*normativa*" y "*ley*" indican que hay un considerable enfoque en el marco regulatorio que afecta a las operaciones de las empresas Fintech.
Finalmente, otros términos relevantes como "*usuario*", "*proyecto*", "*riesgo*", "*pago*" y "*servicio*" muestran la diversidad de temas tratados en los documentos. Estos incluyen desde la gestión de riesgos hasta las soluciones de pago, reflejando la amplia gama de intereses y áreas de estudio en el campo de Fintech.

4.5. Embedding usando BERT

Se define una función **`get_bert_embeddings`** que tokeniza el texto, lo pasa por el modelo BERT, y obtiene la última capa oculta de representaciones. La función promedia estas representaciones para obtener una única representación vectorial del texto. Esta función se aplica a cada texto en el DataFrame, almacenando los embeddings resultantes en una lista.
Los embeddings se convierten en un DataFrame, al cual se añade una columna de etiquetas originales del texto. Esto facilita la manipulación y análisis posterior de los datos.
El DataFrame de embeddings se guarda en un archivo CSV para ser utilizado por los modelos predictivos de clasificación que se aplicarán en el siguiente apartado.

5. ENTRENAMIENTO DE MODELOS PREDICTIVOS DE CLASIFICACIÓN

5.1. Definición de los modelos

En este apartado se describe el proceso de entrenamiento y evaluación de varios modelos de machine learning utilizando embeddings generados por BERT para la clasificación de textos. El objetivo es comparar el rendimiento de diferentes algoritmos y seleccionar el más adecuado para la tarea.
Inicialmente, el DataFrame de embeddings se divide en características (**`X`**) y etiquetas (**`y`**). Se utiliza la función **`train_test_split`** para dividir los datos en conjuntos de entrenamiento y prueba, con un 80% de los datos destinados al entrenamiento y un 20% a la prueba.
Se definen cinco modelos de clasificación: *Regresión Logística, SVM (Máquinas de Vectores de Soporte), Random Forest, Gradient Boosting y KNN (K-Nearest Neighbors).*
Cada modelo se entrena utilizando el conjunto de datos de entrenamiento (**`X_train, y_train`**). Posteriormente, se predicen las etiquetas para el conjunto de prueba (**`X_test`**), y se evalúan los resultados mediante varias métricas: precisión, exactitud, recall y F1 score. Estas métricas proporcionan una visión completa del rendimiento de cada modelo, considerando tanto la capacidad de clasificación correcta como el equilibrio entre precisión y recall.
Además de las métricas, se genera y muestra la matriz de confusión para cada modelo, lo que permite visualizar las predicciones correctas e incorrectas. Esto ayuda a identificar patrones específicos de errores y la distribución de las predicciones.
Por último, se imprime un informe de clasificación detallado para cada modelo, proporcionando una comparación clara de su rendimiento en términos de precisión, recall y F1 score para cada clase (irrelevante y relevante).

5.2. Optimización de los modelos

Su aplicación se puede visualizar en el código en Python adjunto.

5.3. Resultados de los modelos

Tabla Nº1: Métricas de errores de los modelos

Model	Accuracy	Precision	Recall	F1 Score
Regresión Logística	1.000000	1.000000	1.000000	1.000000
SVM	0.928571	0.862245	0.928571	0.894180
Random Forest	1.000000	1.000000	1.000000	1.000000
Gradient Boosting	0.928571	0.964286	0.928571	0.939048
KNN	1.000000	1.000000	1.000000	1.000000

Fuente: elaboración propia en Python.

Observando la tabla se afirma que los modelos sin errores son *Regresión Logística, Random Forest y KNN*. Estos modelos han clasificado perfectamente todos los documentos, logrando una precisión, recall y f1-score de 1.00 para ambas clases.
Por otro lado, los modelos que cometieron errores fueron *SVM* que no ha clasificado correctamente ningún documento irrelevante, lo que resulta en una precisión y recall de 0.00 para esta clase, y *Gradient Boosting cometió* errores en la clasificación de documentos relevantes, mostrando una menor precisión y f1-score para esta clase en comparación con otros modelos.
Estos modelos pueden ser mejorado utilizando técnicas adicionales de preprocesamiento que se explicarán en el apartado siguiente.

5.4. SMOTE: Técnica de balanceo de clases

En el apartado *"2.2. Análisis exploratorio"* se resaltó la existencia de desbalanceo de clases de manera muy significativa. Ante esta situación, se aplica la técnica de SMOTE a la **clase minoritaria,** que son los documentos "*Irrelevantes*".
Luego se reentrenaron los modelos y se observó una mejora de estos.

Tabla Nº2: Métricas de errores de los modelos

Model	Accuracy	Precision	Recall	F1 Score
Regresión Logística	1.000000	1.000000	1.000000	1.000000
SVM	1.000000	1.000000	1.000000	1.000000
Random Forest	1.000000	1.000000	1.000000	1.000000
Gradient Boosting	1.000000	1.000000	1.000000	1.000000
KNN	0.928571	0.964286	0.928571	0.939048

Fuente: elaboración propia en Python.

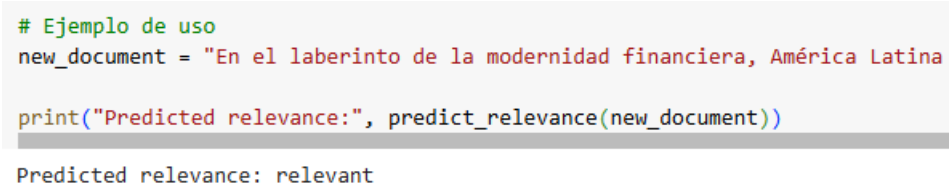
Observando la tabla se afirma que el modelo *SVM* mostró una mejora significativa, pasando de no clasificar correctamente ningún documento irrelevante a clasificar perfectamente todos los documentos. El modelo *Gradient Boosting* mejoró de manera considerable, logrando una precisión, recall y f1-score de 1.00 para ambas clases después de aplicar SMOTE.
Los modelos de *Regresión Logística, Random Forest y Gradient Boosting* ya mostraban un buen rendimiento antes de aplicar SMOTE y lograron una clasificación perfecta después de aplicar SMOTE.

La aplicación de SMOTE ha demostrado ser una técnica efectiva para mejorar el rendimiento de los modelos de machine learning en conjuntos de datos desbalanceados. La mejora en precisión, recall y f1-score en varios modelos destaca la importancia de trabajar con conjuntos balanceados para obtener evaluaciones más precisas y justas, y para mejorar la capacidad de generalización de los modelos.

5.5. Evaluación con nuevos textos

En esta sección se evalúa si los modelos funcionan correctamente. Para ello, se consideran dos ejemplos que se obtuvieron de Google a través de realizar búsquedas de las siguientes palabras clave: *"inclusión financiera"* y *"aprendizaje"*.

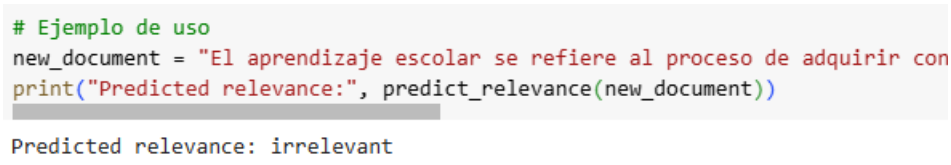
Figura Nº1: Ejemplo de texto relevante




```
new_document = "En el laberinto de la modernidad financiera, América Latina emerge como un escenario vibrante, donde la danza de la tecnología financiera (FinTech) promete guiar a sus habitantes hacia un nuevo amanecer de inclusión y prosperidad."
```

Fuente: elaboración propia en Python.

Figura Nº2: Ejemplo de texto irrelevante




```
new_document = "El aprendizaje escolar se refiere al proceso de adquirir conocimientos, habilidades y competencias dentro de un entorno educativo formal, como una escuela, instituto o universidad. Es un componente esencial de la educación formal y proporciona a los estudiantes las bases necesarias para su desarrollo académico y personal. En este contexto, los alumnos interactúan con contenidos curriculares, profesores y compañeros, construyendo significados y comprendiendo conceptos clave. El aprendizaje escolar no solo implica la adquisición de información, sino también la aplicación activa de ese conocimiento en situaciones relevantes. Así, se busca que los estudiantes desarrollen habilidades críticas, creativas y analíticas que les permitan enfrentar desafíos tanto dentro como fuera del aula. Si tienes alguna pregunta específica sobre el aprendizaje escolar, no dudes en preguntar"
```

Fuente: elaboración propia en Python.

En ambos ejemplos, las predicciones clasifican correctamente ambos textos, por lo tanto, el preprocesamiento de los datos de texto, el modelamiento y las optimizaciones efectuadas son correctas para el corpus objeto de análisis.

6. TOPIC MODELING CON LDA

El modelamiento de tópicos con LDA es una técnica de aprendizaje no supervisado que tiene como objetivo identificar patrones temáticos en un corpus de textos. En lugar de etiquetar manualmente cada documento con temas específicos, *topic modeling* automáticamente organiza, comprende y etiqueta una colección de documentos.

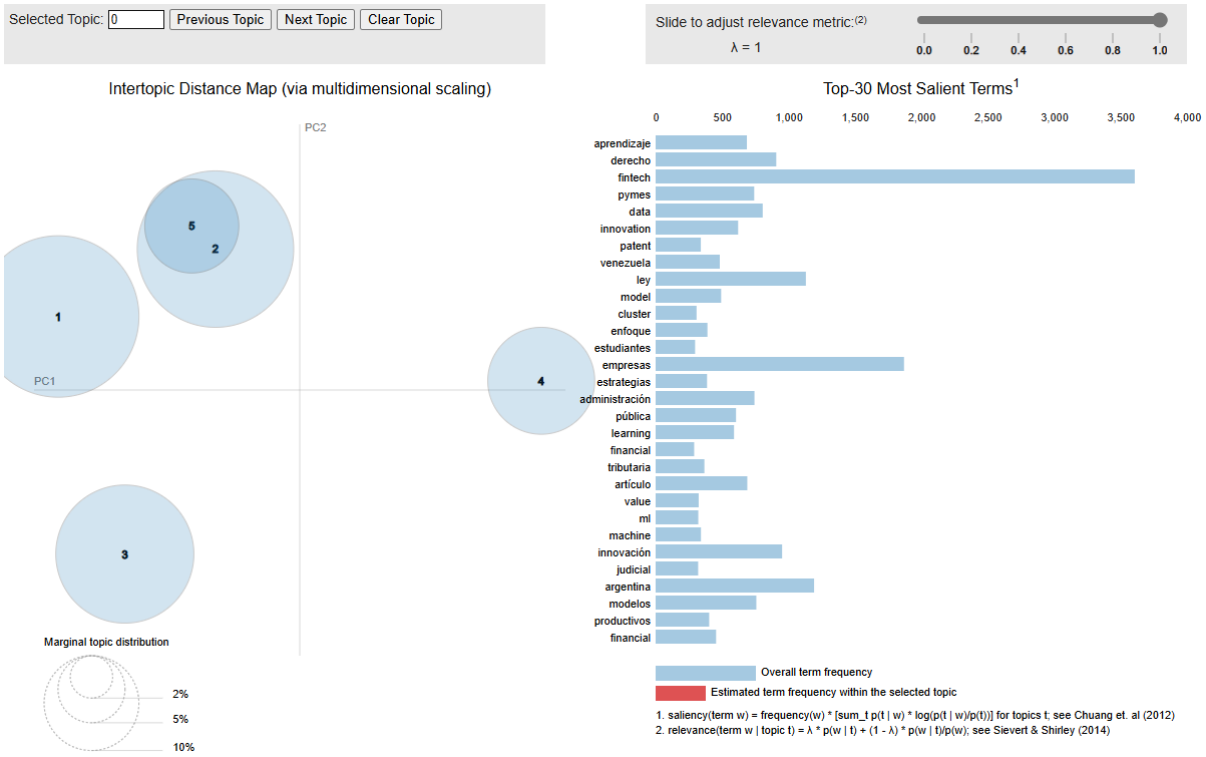
En primer lugar, se realizaron pruebas iniciales considerando 3 temáticas (de acuerdo con el número de clusters definidos en el apartado *"3. CLUSTERING DE TEXTOS CON K-MEANS")* y el resultado de la coherencia fue de 0.4717.

Luego se realizaron pruebas de mejoras, probando diferentes números óptimos de tópicos y el mejor resultado se logró con 5 temáticas cuyo resultado de coherencia aumentó a 0.6068.

El gráfico Nº10 se interpreta de la siguiente manera:

- *Tema 2:*
  1. Palabras clave: Las palabras clave más relevantes incluyen "*derecho*", "*innovación*", "*administración*", "*artículo*", "*pública*", "*ley*", entre otras.
  2. El Tema 2 parece estar relacionado con temas legales y de administración pública. Palabras como "ley", "derecho" y "pública" sugieren un enfoque en aspectos legales y administrativos de las Fintech o áreas relacionadas.
- *Tema 1, 3, 4 y 5:* Los otros círculos representan otros temas. El análisis de los términos más relevantes para cada uno de estos temas ayuda a identificar el enfoque de cada tema.

Gráfico Nº10: Mapa de distancias entre temas



Fuente: elaboración propia en Python.

7. CONCLUSIONES

En este trabajo se ha demostrado la efectividad de aplicar técnicas avanzadas de procesamiento de lenguaje natural (NLP) y métodos de clasificación para identificar y categorizar documentos relevantes en el ámbito

académico o de investigación, más precisamente en relación con la temática *Fintech*. A lo largo del proyecto, se realizaron diversas etapas que permitieron alcanzar los objetivos planteados de manera exitosa.

El preprocesamiento de texto fue un paso crucial en el cual se aplicaron técnicas de limpieza, normalización, tokenización y lematización. Estos procesos fueron fundamentales para eliminar el ruido y asegurar que los datos estuvieran en un formato adecuado para el análisis posterior. La eliminación de stopwords y la conversión de las palabras a su forma base permitieron una representación más coherente y uniforme de los textos.

Posteriormente, se empleó el algoritmo K-Means para realizar un clustering de los textos. Esta técnica permitió agrupar los documentos en clusters, identificando patrones y similitudes entre ellos. La visualización de los clusters mediante PCA reveló una separación clara entre los diferentes grupos, facilitando una mejor comprensión de la estructura y variabilidad de los textos en el corpus.

La implementación de BERT para la obtención de embeddings de alta calidad representó un avance significativo en la representación de los textos. BERT capturó el contexto y el significado de las palabras, proporcionando una base sólida para el análisis y la clasificación. Estos embeddings mejoraron la precisión y eficiencia de los modelos predictivos aplicados posteriormente.

En la etapa de entrenamiento de modelos predictivos, se evaluaron varios algoritmos de clasificación, incluyendo *Regresión Logística, SVM, Random Forest, Gradient Boosting* y *KNN*. Los resultados mostraron que algunos modelos, como *Regresión Logística y Random Forest*, lograron una clasificación perfecta, mientras que otros, como *SVM y Gradient Boosting*, mejoraron significativamente tras aplicar técnicas de balanceo de clases como SMOTE. La evaluación mediante métricas de precisión, recall y F1 score, junto con las matrices de confusión, permitió identificar patrones de errores y ajustar los modelos para obtener un rendimiento óptimo. Aunque los modelos desarrollados han mostrado resultados prometedores, su desempeño puede mejorar con una mayor cantidad de documentos, dado que lo utilizado en este trabajo fue a modo ejemplificativo.

Además, el código implementado en Python puede utilizarse para datos de textos que se obtengan de otras fuentes, como scraping de web, APIs o cualquier otra fuente que se considere necesaria. Esto amplía las posibilidades de aplicación de las técnicas desarrolladas en el trabajo práctico, permitiendo un análisis más amplio y diverso de diferentes tipos de documentos.

Para concluir, se aplicó la técnica de LDA para el modelado de tópicos, lo cual ayudó a identificar temas subyacentes en el corpus. Este análisis temático permitió organizar los documentos de manera más coherente y descubrir patrones importantes en los datos.

8. LINK DE COLAB

https://colab.research.google.com/drive/1VLmh1i3eXQyDlaSXcNVP4W4yF_rmdqaC#scrollTo=UchQPNthDN9J

9. REFERENCIAS

- Hovy, D. (2020). Text analysis in Python for social scientists: Discovery and exploration. Cambridge University Press. Recuperado del Aula Virtual de la Universidad de Buenos Aires.
- Repositorio material:https://github.com/dirkhovy/text_analysis_for_social_science/tree/main Recuperado del Aula Virtual de la Universidad de Buenos Aires.
- Repositorio material:https://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/ Recuperado del Aula Virtual de la Universidad de Buenos Aires.
- Repositorio material: https://github.com/rivaquiroga/analisis-de-texto-python-2023 Recuperado del Aula Virtual de la Universidad de Buenos Aires.